Herausragende Masterarbeiten

Autor*in

Quang Hai Nguyen

Studiengang

Software Engineering for Embedded Systems, M.Eng.

Masterarbeitstitel

Evaluation and development of the bridging application between ISO 15118 and OCPP 2.0.1 protocols

RPTU

Distance and Independent Studies Center
DISC

Rheinland-Pfälzische Technische Universität
Kaiserslautern-Landau

Distance Study Program
Software Engineering for Embedded Systems

Master's Thesis

Evaluation and development of the bridging application between ISO 15118 and OCPP 2.0.1 protocols

Provided by
Nguyen, Quang Hai

First supervisor: Prof. Dr.-Ing. Peter Liggesmeyer
Second supervisor: Ms. Zai Müller-Zhang

Declaration

Ich versichere, dass ich diese Masterarbeit selbstständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt und die den benutzten Quellen wörtlich oder inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Place, date	Signature
Hannover, 31.03.2023	NGUYEN, QUANG HAI

Abstract

The increase in the number of electric vehicles(EVs) has undoubtedly put stress on the local power grid because these systems were designed without anticipating the charging needs of electric vehicles. To overcome this problem, Smart Charging is introduced to allow the Charging Stations Management System(CSMS) to load-balance the charging needs of the electric vehicles during peak hours. In addition, it allows the EVs to return their energy to the system when needed. Smart Charging uses the de facto standards ISO 15118 and OCPP to enable the CSMS to control the charging profiles of the EVs. Since these protocols are specified by different organizations, their compatibility must be analyzed to ensure their interoperability.

In the first part, this thesis aims to apply a theoretical analysis method to analyze the compatibility between ISO 15118 and OCPP. This method uses the Symbolic Transition System to model the interactions between the protocols. Then, the state transitions and message exchanges of the models are analyzed using the flooding algorithm. The result of this analysis is a compatibility matrix, which illustrates the degrees of compatibility between the states of the protocols. Based on the results, this thesis concludes that ISO 15118 and OCPP are compatible. However, their compatibility is not perfect because of data type incompatibility between messages. The reason is that ISO 15118 uses domain data types for its parameters, while OCPP uses generic data types to increase its interoperability with other protocols.

The second part of this thesis describes the concept and design of the application to bridge the communication between ISO 15118 and OCPP. The application also demonstrates how to overcome the problems found in the compatibility analysis using facade patterns. In addition, the development of the bridging application highlights several issues that have arisen in practice. The first issue is, due to the large memory footprint of the messages, the OCPP stack is not suitable for running on small embedded systems without extreme optimization. Second, using JSON, a human-readable format, to encode the OCPP messages is unnecessary because most of the messages are processed by machines. In addition, the OCPP application is highly complex due to the nested conditions involved in sending and receiving OCPP messages. Finally, both the JSON and EXI data formats require serializers (parsers) to encode (decode) the messages, adding to the complexity of the system.

Table of Contents

Abbreviations

AC	Alternate Current
CP	Control Pilot
CSMS	Charging Station Management System
DC	Direct Current
EIM	External Identification Means
EV	Electric Vehicle
EVCC	Electric Vehicle Communication Controller
EVSE	Electric Vehicle Supply Equipment
EXI	Efficient XML Interchange format
IEC	International Electrotechnical Commission
JSON	JavaScript Object Notation
HPGP	HomePlug Green PHY
IPv6	Internet Protocol version 6
ISO	International Organization for Standardization
OCA	Open Charge Alliance
OCPP	Open Charge Point Protocol
OEM	Original Equipment Manufacture
OSI	Open System Interconnection
PLC	Power Line Communication
PWM	Pulse Width Modulation
RPC	Remote Procedure Call
SECC	Supply Equipment Communication Controller
SLAC	Signal Level Attenuation Characterization

TCP	Transmission Controller Protocol
TLS	Transport Layer Security
UDP	User Datagram Protocol
V2G	Vehicle-2-Grid
V2GTP	Vehicle-2-Grid Transfer Protocol
STS	Symbolic Transition System
XML	Extensible Markup Language
XSD	XML Schema Definition

List of Tables

List of Figures

Listings

1 Introduction

According to The Federal Government, 2020, seven to ten million Electric Vehicle (EV)s will be registered in Germany by 2030, responsible for twenty-three Tera-Watt-hour of charging demand (Bermejo et al., 2021). Even though the energy demand is enormous, it only accounts for an eight percent increase. However, charging millions of EVs will cause massive stress on the local energy distribution system during peak times. Engel et al., 2018 indicates that the peak circuit load can increase by thirty percent with twenty-five percent of EV penetration. Therefore, Smart charging is introduced to cope with the issue by employing several strategies to smooth out the load during peak time. AMPECO, n.d. lists three primary techniques for smart charging:

- Load shifting: the EVs are charged slower during peak time and faster during off-peak time.
- Peak shaving: the system automatically controls the charging process so that the charging power does not exceed its capability.
- Dynamic load balancing: all EVs are charged within the maximum charging limit and according to their capacity.

With bidirectional energy transfer enabled, the EVs can even feed the energy from their battery back to the grid to support the grid during peak hours. To enable Smart Charging, the back-end system must know the charging status of the charging stations, and EVs connecting to the grid. Furthermore, the back-end system must control the charging stations to influence the energy delivered to the EVs, which optimizes the charging process and the grid's load. Therefore, Smart Charging requires a significant amount of data, such as charging parameters, metering values, EVs status, and charging stations status, to be exchanged among EVs, Electric Vehicle Supply Equipment (EVSE), and Charging Station Management System (CSMS).

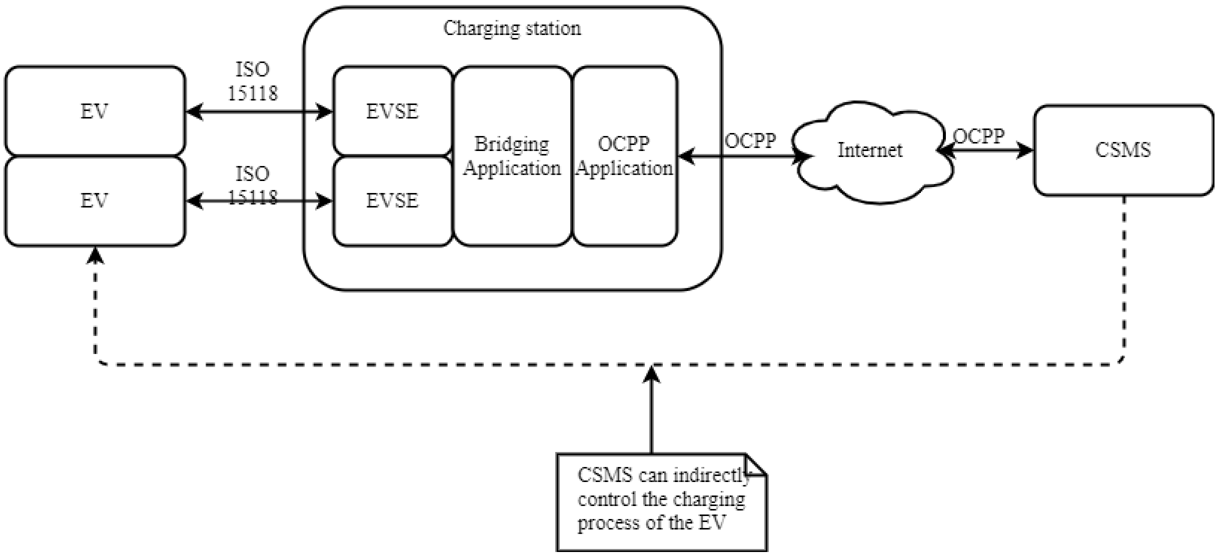


Figure 1: Overview of EV charging system

Various communication standards are proposed to build up such a scenario, for example, IEC 61851, ISO 15118, and CHAdeMO for communication between EV and EVSE; Open Charge Point Protocol (OCPP), IEC 63110, OpenADR, and EEBUS for the communication between the EVSE and back-end systems (Neaimeh and Andersen, 2020). Nevertheless, the de facto standards in the industry are ISO 15118 and OCPP 2.0.1. ISO 15118 defines the communication standard between the EVs and the EVSEs, which allows the EVs to exchange data, such as charging parameters, EV's status, EVSE's status, metering values, and authorization information, with the EVSEs. On the other hand, OCPP specifies the communication standard between EVSEs and CSMS, providing the methods for CSMS to manage, monitor and control EVSEs. Consequently, to implement a complete communication chain between the EVs and CSMS, the protocol adopters must build a bridging application to receive ISO 15118 messages, transform them into OCPP messages, transmit the OCPP messages and vice versa (Figure 1).

1.1 The Challenges of Charging Protocols Integration

The challenges of implementing Smart Charging are that each protocol is defined by different organizations. ISO 15118 standard is a joint work between the International Organization for Standardization (ISO) and the International Electrotechnical Commission (IEC). Meanwhile, the OCPP is defined by the Open Charge Alliance (OCA). Although both protocols are TCP/IP based (Figure 5 and 8), they have different requirements regarding message format, data structure, timing, and message exchange sequences. Chapter seven of ISO, 2019 specifies the use cases and the interactions between the primary actors (EV or EVSE) and the secondary actors (CSMS, Energy management systems, charging station operators) during a charging session. However, those use cases do not describe their interactions in detail. Open Charge Alliance, 2020b specifies more in detail (compared to ISO, 2019) about its support for ISO 15118 in various OCPP use cases (Table 3). Those use cases are presented by message sequences, illustrating which OCPP messages are used in cooperating with ISO 15118 messages. These message sequences nonetheless lack the detail of which data and data types are exchanged between the protocols.

Various studies on ISO15118 and OCPP have been carried out. For instance, Kern, 2021 investigated the privacy and security of ISO 15118 and OCPP ; Klapwijk and Driessen, 2017 addresses which charging protocols are suitable for which functionalities based on their interoperability, maturity, market adoption, and openness; Wellisch et al., 2015 proposes the smart charging capable Alternate Current (AC) charging station based on ISO 15118 and OCPP; Schmutzler et al., 2013 provides a review on the ISO 15118 standard and how OCPP can leverage the smart charging. However, there is little work to investigate whether ISO 15118 and OCPP are compatible and the challenges when developing an application to bridge these protocols.

Both the ISO 15118 and OCPP standards will be increasingly adopted due to the popularity of EVs. Therefore, the standards need to be highly interoperable so that different protocol implementations from different vendors can work together. As a result, a protocol compatibility analysis must be performed to measure the interoperability of ISO 15118 and OCPP. The analysis must answer the questions of how well ISO 15118 and OCPP are compatible. Protocol compatibility means that

- The interactions between the protocols do not cause one or all of them to enter a deadlock state.
- The data exchanged between the protocols is similar in structure and data type. This similarity is important because it reduces the effort of data transformation, i.e. the developer can take the data from one protocol and send it immediately via the other protocol, which reduces the complexity of the software implementation.

In practice, the bridging application must also be implemented in order to verify how much impact the incompatibility has during development and how much effort the developers have to invest to implement such an application. In addition, the implementation of the bridging application provides additional incompatibilities that were not anticipated in the compatibility analysis.

1.2 The Objectives of the Thesis

The objectives of the thesis are to apply the method proposed by Ouederni, M., Salaün, G., & Pimentel, E. (2010). *Measuring the compatibility of service interaction protocols - technical report iti 4-10* to analyze the compatibility between OCPP and ISO 15118. Then, a Bridging application is developed, based on SEVENSTAX GmbH ISO 15118 stack and OCPP stack, to verify the compatibility of the protocol in practice. The outcomes of the thesis are:

- The compatibility between ISO 15118 and OCPP is analyzed based on the theoretical methodology. The analysis answers the question of how good the interoperability between ISO 15118 and OCPP is and what are the incompatibilities between them. Based on the analysis methodology, a prototype tool is developed to calculate the degree of compatibility of the protocols.

- The problems that EVSE manufacturers encounter in practice when integrating these protocols into their systems and how to overcome them; and the recommendations for the development of the bridging application between ISO15118 and OCPP protocols.

1.3 The Structure of the Thesis

The rest of the thesis is divided into four chapters: Background, Protocols Compatibility Analysis, Development a Protocols Bridging Application, and Conclusion and Future Work.

The Background chapter gives an overview of the state of the art in protocol compatibility analysis and explains why the method proposed by Ouederni et al., 2010 is suitable for analyzing the interaction between ISO 15118 and OCPP. In addition, this chapter describes how this analysis method works and how the result is calculated. Next, this chapter describes the structures of ISO 15118 and OCPP and their software implementation. Finally, this chapter explains how ISO 15118 and OCPP interact, which is the input for protocol analysis.

The Protocols Compatibility Analysis chapter describes how to model the interaction between ISO 15118 and OCPP and how to calculate the compatibility between them. Then, based on the result, the compatibility between the ISO 15118 and OCPP is concluded. Any incompatibility found in this chapter will be resolved in the following chapter.

The Development a Protocols Bridging Application chapter describes the architecture and design of the Bridging application and how this application addresses the incompatibilities found in the theoretical analysis. In addition, this chapter identifies any pitfalls not identified by the theoretical method and proposes solutions.

The Conclusion and Future Work chapter concludes with a discussion of how well the two protocols are compatible, the challenges that EVSE vendors may face in implementing these protocols, proposed solutions to overcome the challenges, and future implementations.

2 Background

This chapter provides the background theory necessary for the protocol compatibility analysis and Bridging application development. Firstly, it reviews the state of the art in protocols compatibility analysis, explains why Ouederni et al., 2010 is a suitable method for analyzing OCPP and ISO 15118, and describes the procedure for analyzing protocol compatibility using the mentioned method. Secondly, It describes the purpose, network layers, message construction and use cases of ISO 15118 and OCPP to provide an insight into both standards. In addition to the protocol explanation, the OCPP section also highlights the similarities and differences between itself and ISO 15118. Thirdly, this chapter briefly mentions the implementations of OCPP and ISO 15118 by SEVENTAX GmbH. Finally, this chapter explains the interaction and the data exchange between the protocols during a charging session using a message sequence diagram.

2.1 State-of-the-Art Compatibility Analysis

The analysis of protocols or services interaction is a well-known problem in the service-oriented-computing area, where the services' interoperability is analyzed. In other words, the requests of one service are served by its counterpart and vice versa so that both services can evolve to the next states. In services interactions, the state transitions and message exchange are defined as the service's protocol. Therefore, services are compatible if their underlying protocols are compatible. Starting from here, the term protocol and service are used interchanged. The existing analysis methods employ one of the modeling systems, such as Petri-Net and its variants (Yang et al., 2009, Martens, 2003, and Aalst et al., 2009), π-calculus (Wu et al., 2009), or state machine (Gao and Wei, 2011 and Elabd et al., 2009), to describe the behaviors of the services. Then, those methods utilize formalized algorithms and automation tools to check the compatibility of the services based on their models.

Martens, 2003 proposed the method for analyzing the services by modeling their behaviors using Petri-Net. Then the method described the process of analyzing the service compatibility by composing the models and using the compatibility notions such as syntactical and semantical compatibility. The outcome of the analysis is to answer whether the services are compatible and usable.

Gao and Wei, 2011 introduced the method for checking the compatibility of the service protocols, in which the service interactions are modeled using a finite state machine. This method introduces the context-awareness concept, which specifies that a transition occurs only when the message and the constrained context are matched. Then, the compatibility analysis is performed based on services product automata using a transversing approach and observable compatibility analysis. The outcome of this method is to answer whether the service protocols are compatible.

Elabd et al., 2009 proposed the compatibility analysis based on modeling the interaction between services using state machines. This method introduces the time restriction concept, meaning the state transitions require compatible messages under a specific interval. Similar to Gao, this method utilizes the observable compatibility and states transversal of the protocols product automata to analyze the compatibility of the protocol.

Wu et al., 2009 proposed the analysis approach using the π-calculus to model the protocol interactions. This method also utilizes the observable compatibility and state transversal to analyze the compatibility. Furthermore, this method introduces the algorithm to calculate the compatibility degree instead of only giving the Boolean answer if the service is compatible. Finally, this method proposes a tool to perform the analysis automatically.

Yang et al., 2009 approach employs the Colored Petri-Net to model the service behaviors. The model is analyzed using the Correct Process compatibility notion. Besides checking the states and the messages, this method considered the data type of the messages. The outcome of the analysis is to answer whether the protocols are compatible.

Aalst et al., 2009 provided a method to address the challenges in the services' interaction, namely services exposition, service refinement and replacement, and service adaption, using Petri-Net. Although the paper considered the behaviors of the services and messages exchange, it focused on the service replacement and adaption and did not give a clear indicator to determine if the services are compatible.

Even though the aforementioned methods are promising candidates to analyze the interaction between ISO 15118 and OCPP, they lack the aspects to analyze the charging protocols specifically. Martens, 2003, Gao and Wei, 2011, Elabd et al., 2009, Wu et al., 2009 and Aalst et al., 2009 analyze the compatibility regarding the state transitions and message exchange, but they do not take the parameters and parameters data types into consideration. Yang et al., 2009 considers only the data type of the message on a high level but does not evaluate the message parameters. Martens, 2003, Gao and Wei, 2011, Elabd et al., 2009, Yang et al., 2009, Aalst et al., 2009 return the result of the compatibility analysis with the Boolean answer-true or false, which is too straightforward for

ISO 15118 and OCPP. OCPP claims to support ISO 15118, so they are compatible to a certain degree. However, those protocols are mismatched in detail due to being specified by different organizations. Therefore, the analysis method must return a more elaborated result than a Boolean answer. During the review, Ouederni et al., 2010 was found as a suitable candidate for analyzing ISO 15118 and OCPP.

Ouederni et al., 2010 proposed a method to measure the compatibility of protocols using the Symbolic Transition System, which describes the system using states, transitions, labels (messages), and list of parameters. This method evaluates compatibility using unspecified reception and unidirectional complementarity notions, which are based on the analysis of message compatibility. The compatibility of the messages is evaluated on the basis of the parameters and their data types. The result of the analysis is a compatibility matrix in which the compatibility is scored between zero and one, where zero means incompatible and one means perfectly compatible. The following section describes in detail the compatibility analysis method proposed by Ouederni et al., 2010. In addition, some formulae are modified to meet the requirements for analyzing ISO 15118 and OCPP.

2.2 Theory of Compatibility Analysis

2.2.1 Modeling the Protocol Interaction

Ouederni et al., 2010 proposed the model of the protocol interaction using the Symbolic Transition System. The Symbolic Transition System is a tuple (A, S, I, F, T) where:

- A: set of labels associated with the transitions.
- S: is the set of states
- I: the initial state
- F: the non-empty set of the final state
- T: transition relation

Ouederni et al., 2010 specified that a label is either τ, for internal action or a tuple of (m, d, pl), where:

- m is the message name.
- d is the direction of communication (! for transmission and ? for reception).
- pl is the list of parameters and their data type.

Figure 2 demonstrates the interaction between the engineer service and the database service based on the Symbolic Transition System. At the initial states, the engineer service transmits the *register* message to the database service and moves to the next state (c_1). The *register* message has only one parameter, id, with data type int. Similarly, the database service can advance to the next state (s_1) when it receives the *register* message from the engineer service. Then, the database service transmits the *ack* message and advances into the final state (s_2). At the same time, the engineer service evolves to the final state when it receives the *ack* message.

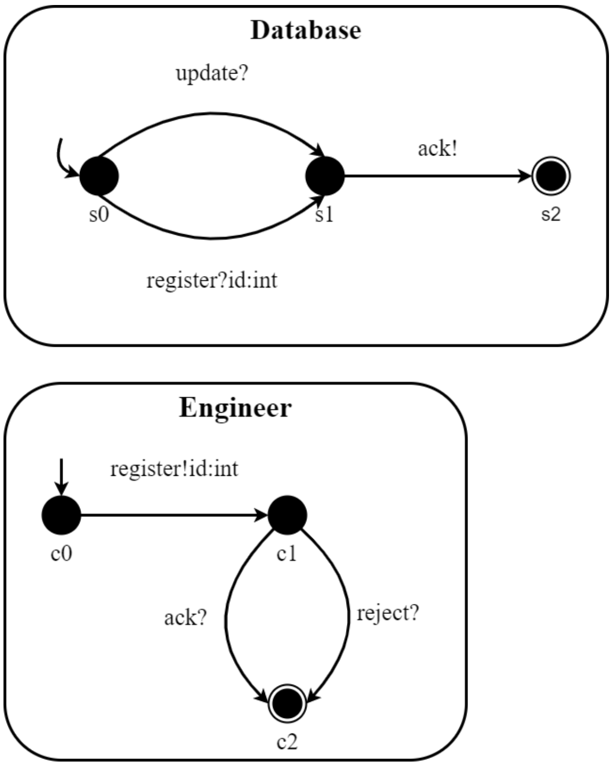


Figure 2: Example of the interaction between the engineer and database services. Adapted from Ouederni et al., 2010

Additionally, according to the aforementioned definitions, it can be deduced from Figure 2 that:

$$A_s = \{register?id : int, update?, ack!\}, S_s = \{s_0, s_1, s_2\}, I_s = \{s_0\}, F_s = \{s_2\},$$

$$T_s = \{(s_0, update!, s_1), (s_1, ack!, s_2), (s_0, register?id : int, s_1)\}$$

$$A_c = \{register!id : int, reject?, ack?\}, S_c = \{c_0, c_1, c_2\}, I_c = \{c_0\}, F_c = \{c_2\},$$

$$T_c = \{(c_0, register!id : int, c_1), (c_1, reject?, c_2), (c_1, ack?, c_2)\}$$

2.2.2 Compatibility Notion

Ouederni et al., 2010 defined two compatibility notions to measure the protocol compatibility: Unspecified Receptions and Unidirectional Complementary. The definitions of these notions are following:

Unspecified Receptions (Bidirectional Compatibility): Two services are compatible with this notion:

- If they are deadlock-free at their initial global state.
- If one service sends a message at the reachable state, its partners must receive that message and both services advance to a compatible state.

Unidirectional Complementary: Two services are compatible with this notion:

- If there is one service (complementer) receives (sends respectively) all messages that its partner (complemented) sends (receives respectively) at all reachable states.
- Both services must be free from deadlock in all reachable state

From the aforementioned definitions, the Unspecified Receptions requires that one service must support all transmissions from its counterpart and vice versa, and this behavior ensures that no transmissions are left un-handled. The second notion - Unidirectional Complementary implies that only one service (complementer) must support all the transmissions and receptions from its counterpart (complemented). This notion is useful to specify the interaction of the client-server model, where the server must support all transmissions and receptions from the client. Those definitions mention three terms: deadlock freedom, compatible states, and reachable states, which are defined by Ouederni et al., 2010 as follows:

Deadlock freedom: Two protocols are considered freedom at a state pair (s_1, s_2) if and only if both states are final states or these protocols are deadlock-free in each state reachable from the current state pair.

Reachable states: reachable states are the state pairs that the protocol can reach from a current state pair by synchronization on compatible labels, which are the tuples of (m, d, pl), or internal transition τ. The compatible label notion is explained in the Static Compatibility section.

State Compatibility: A state pair (s_1, s_2) are compatible whether the message sent (received respectively) by one protocol are state s1 is received (sent respectively) by another protocol at state s2, and both protocol advance to the compatible states. If one protocol cannot interact with the other protocol's action, then both protocols must advance to a state (s_1, s_2'), where the action is supported.

Section 2.5 shows that both ISO 15118 and OCPP are deployed in the same target (the charging station), and their interactions are triggered internally by the function calls instead of message exchanging via a physical bus. Therefore, the transmissions (or interactions) from ISO 15118 are always supported by the OCPP and vice versa. This behavior reassembles the characteristic of the Unspecified Receptions notion; hence, the Unspecified Receptions notion is applied to measure the compatibility between the protocols. Starting from this point, any formulas and calculations presented in the thesis are considered the Unspecified Receptions notion.

2.2.3 The Process of Measuring Protocol Compatibility

Ouederni et al., 2010 proposed the process (presented in Figure 3) to measure protocol compatibility, in which static compatibility is evaluated by analyzing the state nature, messages parameters, and messages labels. Then, the behavioral compatibility is calculated based on Bidirectional Propagation or Unidirectional Propagation, which utilizes observational compatibility, which, in turn, requires static compatibility as input. Finally, the state compatibility of each state of the protocol is calculated based on the state nature and the behavioral compatibility. The state compatibility is calculated in a defined iteration, and results are presented in the matrix $S_1 \times S_2$, where S_1 and S_2 are the set of states of both protocols. The compatibility matrix can be used to identify the mismatches.

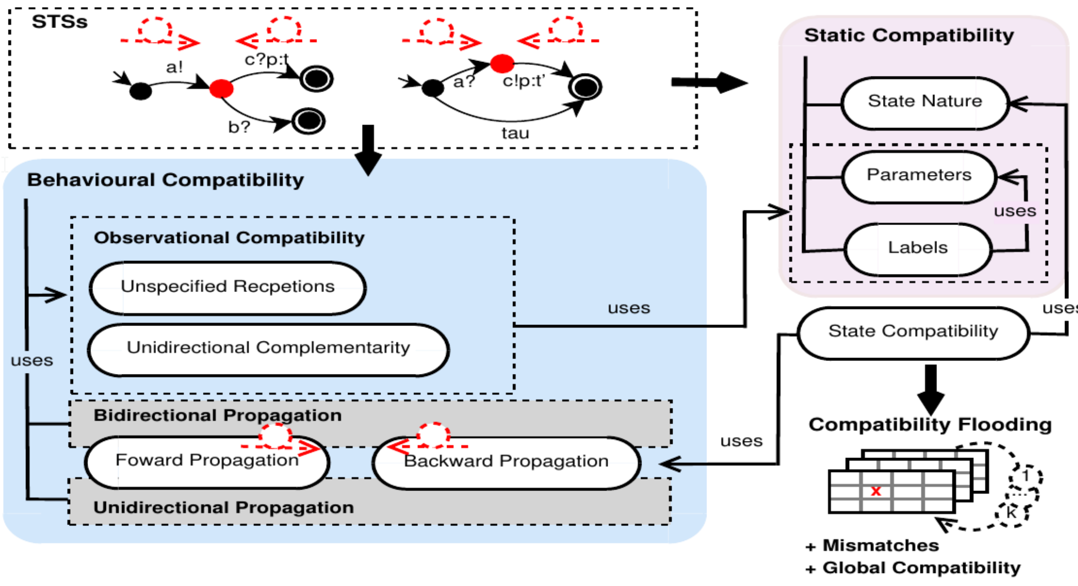


Figure 3: The process of measuring the protocol compatibility. Extracted from Ouederni et al., 2010

In the next section, the method of calculating the compatibility is presented, where the observational compatibility is based on the Unspecified Receptions notion.

2.2.3.1 Static Compatibility

Static compatibility is measured based on state natures, labels, and exchange parameters. Ouederni et al., 2010 specified the calculations as follows:

State Nature. The function $nat(s_1, s_2)$ return 1 if state s_1 and s_2 have the exact nature, i.e., both s_1 and s_2 are initial states, final states, or none of them; otherwise, it returns 0. As an example, in Figure 2, $nat(s_0, c_0) = nat(s_2, c_2) = nat(s_1, c_1) = 1$ and $nat(s_0, c_1) = 0$

Parameters. Parameter compatibility between two parameters list pl_1 and pl_2 is measured by the function $par\_comp(pl_1, pl_2)$, which is the average of the number compatibility $number(pl_1, pl_2)$, order compatibility $order(pl_1, pl_2)$, and type compatibility $type(pl_1, pl_2)$.

$$par\_comp(pl_1, pl_2) = \frac{number(pl_1, pl_2) + order(pl_1 and pl_2) + type(pl_1, pl_2)}{3}$$

Where:

$$number(pl_1, pl_2) = 1 - \frac{abs(\|pl_1\| - \|pl_2\|)}{max(\|pl_1\|, \|pl_2\|)}$$

$$order(pl_1, pl_2) = \frac{\|unorderedTypes(pl_1, pl_2)\|)}{\|sharedTypes(pl_1, pl_2)\|)}$$

The function $unorderedTypes(pl_1, pl_2)$ returns the set of parameters that are not in order. The function $sharedTypes(pl_1, pl_2)$ returns the set of shared types between pl_1 and pl_2.

$$type(pl_1, pl_2) = 1 - \frac{\|unsharedTypes(pl_1, pl_2)\|}{\|pl_1\| + \|pl_2\|}$$

The function $unsharedTypes(pl_1, pl_2)$ returns the set of types that are not shared between pl_1 and pl_2.

Because the interactions between ISO 15118 and OCPP are internal function calls, the number compatibility and order compatibility do not contribute to parameter compatibility but the parameters data types because the messages and parameters are defined by different organizations. Hence, the number and order compatibility are set to 1, and the parameter compatibility function is reduced as follows:

$$par\_comp(pl_1, pl_2) = \frac{2 + type(pl_1, pl_2)}{3} = 1 - \frac{\|unsharedTypes(pl_1, pl_2)\|}{3(\|pl_1\| + \|pl_2\|)}$$

Labels. The function $lab\_comp(l_1, l_2)$ returns 0 if both labels have the same direction. Otherwise, it returns:

$$lab\_comp(l_1, l_2) = \frac{sem\_comp(l_1, l_2) + par\_comp(pl_1, pl_2)}{2}$$

Where $sem\_comp(l_1, l_2)$ measures the semantic compatibility between the message name m_1 and m_2. Ouederni et al., 2010 noted that semantic compatibility could be measured by using Manning and Schütze, 1999 or Pedersen et al., 2004. In ISO 15118 and OCPP use case, the function $sem\_comp(l_1, l_2)$ always returns 1 because ISO 15118 and OCPP use function calls to exchange messages. Therefore, the $lab\_comp(l_1, l_2)$ is reduced as follows:

$$lab\_comp(l_1, l_2) = \frac{1 + par\_comp(pl_1, pl_2)}{2} = 1 - \frac{\|unsharedTypes(pl_1, pl_2)\|}{6(\|pl_1\| + \|pl_2\|)} \tag{1}$$

2.2.3.2 Behavioral Compatibility

Assume there are two protocols describing the following tuples:

$$STS_i = (A_i, S_i, I_i, F_i, T_i)$$

$$STS_j = (A_j, S_j, I_j, F_j, T_j)$$

Ouederni et al., 2010 defined the behavioral compatibility of the two protocols are calcu-

lated based on a flooding algorithm, which returns the compatibility degree as a matrix of $COMP^k_{CN,D}$ where each entry $COMP^k_{CN,D}[s_i, s_j]$ stands for the compatibility of state (s_i, s_j) at the iteration k^{th} with the compatibility notion CN in the direction D (bidirectional or unidirectional).
In the initial iteration, the compatibility matrix is considered to be perfectly matched, meaning:

$$COMP^0_{UR,\leftrightarrow}[s_i, s_j] = 1$$

The $COMP^k_{UR,\leftrightarrow}$ is calculated based on two functions, which are observational compatibility $obs\_comp^k_{UR,\leftrightarrow}$ and state compatibility $state\_comp^k_{UR,\leftrightarrow}$ (Ouederni et al., 2010). The function $state\_comp^k_{UR,\leftrightarrow}$ is dependent on two functions, namely $fw\_propag^k_{UR,\leftrightarrow}$ and $bw\_propag^k_{UR,\leftrightarrow}$. In other to calculate the aforementioned functions, Ouederni et al., 2010 specified the following notations:

- $E(s, T) = \{t \in T | t = (s, (m, !, pl), s')\}$ denotes the emission transition T from state s to state s' with the message m and parameter list pl.
- $R(s, T) = \{t \in T | t = (s, (m, ?, pl), s')\}$ denotes the reception transition T from state s to state s' with the message m and parameter list pl.
- $Fw(s, T) = E(s, T) \cup R(s, T)$ denotes the forward transition from state s to state s'.
- $tau(s, T) = \{t \in T | t = (s, \tau, s')\}$ denotes the internal transition T from state s to state s'.

Then, Ouederni et al., 2010 defined the function $sum^k_{UR,\leftrightarrow}((s_i, s_j), T_i, T_j)$, calculating the sum of the best compatibility degree of the forward neighbors of the state s_i related to those of s_j, as follows:

$$sum^k_{UR,\leftrightarrow}((s_i, s_j), T_i, T_j) = \sum_{(s_i, l_i, s'_i) \in T_i} max_{(s_j, l_j, s'_j) \in T_j}(lab\_comp(l_i, l_j) * COMP^{k-1}_{UR,\leftrightarrow}[s'_i, s'_j]) \tag{2}$$

if $\|Fw(s_i, T_i)\| \neq \varnothing$ and $\|Fw(s_j, T_j)\| \neq \varnothing$, otherwise $sum^k_{CN,D}((s_i, s_j), T_i, T_j) = 0$
The observational compatibility (Ouederni et al., 2010) with respect to the Unspecified Receptions notion is calculated as follows:
If there is a perfect match,

$$obs\_comp^k_{UR,\leftrightarrow}(s_1, s_2) = 1$$

If there is deadlock,

$$obs\_comp^{k}_{UR,\leftrightarrow}(s_1, s_2) = 0$$

Otherwise,

$$obs\_comp^{k}_{UR,\leftrightarrow}(s_1, s_2) = \frac{sum^{k}_{UR,\leftrightarrow}((s_1, s_2), E(s_1, T_1), R(s_2, T_2)) + sum^{k}_{UR,\leftrightarrow}((s_2, s_1), E(s_2, T_2), R(s_1, T_1))}{\|E(s_1, T_1)\| + \|E(s_2, T_2)\|} \tag{3}$$

The forward propagation (Ouederni et al., 2010) at the state (s_1, s_2) can be calculated as follows:

$$fw\_propag^{k}_{UR,\leftrightarrow}(s_1, s_2) = \frac{d\_fw\_propag^{k}_{UR,\leftrightarrow}(s_1, s_2) + d\_fw\_propag^{k}_{UR,\leftrightarrow}(s_2, s_1)}{2} \tag{4}$$

Where $\forall(i, j) \in \{1, 2\}$ & $i \neq j$:
if $tau(s_i, Ti) \neq \varnothing$ and $\|Fw(s_i, T_i)\| = \varnothing$,

$$d\_fw\_propag^{k}_{UR,\leftrightarrow}(s_i, s_j) = \frac{\sum_{(s_i,\tau,s'_i)\in T_i} fw\_propag^{k}_{UR,\leftrightarrow}(s'_i, s_j)}{\|tau(s_i, Ti)\|} \tag{5}$$

otherwise,

$$d\_fw\_propag^{k}_{UR,\leftrightarrow}(s_i, s_j) = \frac{\sum_{(s_i,\tau,s'_i)\in T_i} fw\_propag^{k}_{UR,\leftrightarrow}(s'_i, s_j) + obs\_comp^{k}_{UR,\leftrightarrow}(s_i, s_j)}{\|tau(s_i, Ti)\| + 1} \tag{6}$$

The backward propagation is calculated in the same manner as the forward propagation but using incoming transitions instead of outgoing transitions.
Finally, the state compatibility and the compatibility of state state (s_i, s_j) at the iteration k^{th} are specified by Ouederni et al., 2010 as follows:

$$state\_comp^{k}_{UR,\leftrightarrow}(s_i, s_j) = \frac{w_1 * fw\_propag^{k}_{UR,\leftrightarrow}(s_i, s_j) + w_2 * bw\_propag^{k}_{UR,\leftrightarrow}(s_i, s_j) + w_3 * nat(s_u, s_j)}{w_1 + w_2 + w_3} \tag{7}$$

where Ouederni et al., 2010 noted that:

- w_1: the number of best matching found among the outgoing transition labels in states s_i and s_j.
- w_2: the number of best matching found among the incoming transition labels in

states s_i and s_j.

- w_3: equal 0 if there is at least one state with outgoing or incoming τ transitions, and such that both forward and backward compatibilities are equal to 1. Otherwise, w_3 is set to 1.

The compatibility degree between two states is the average of the state compatibility between them and their previous compatibility degree (Ouederni et al., 2010):

$$COMP^{k}_{UR,D}[s_i, s_j] = \frac{COMP^{k-1}_{UR,\leftrightarrow}[s_i, sj] + state\_comp^{k}_{UR,\leftrightarrow}(s_i, s_j)}{2} \tag{8}$$

Formula 8 is applied to calculate the compatibility degree among the states of the two services, resulting in the compatibility matrix. Ouederni et al., 2010 specifies that the compatibility is computed in an iterative process and that process is terminated when the Euclidean difference $\epsilon_k = \parallel COMP^{k}_{UR,\leftrightarrow} - COMP^{k-1}_{UR,\leftrightarrow} \parallel$ of matrices $COMP^{k}_{UR,\leftrightarrow}$ and $COMP^{k-1}_{UR,\leftrightarrow}$ converges or until a defined k iteration is reached.

Before analyzing the interactions between ISO 15118 and OCPP, the following sections describe the protocol structures, their message formats, their use cases, their software implementations, and their interactions.

2.3 ISO 15118

ISO 15118 standard specifies an IP-based and digital communication protocol between the Electric Vehicle Communication Controller (EVCC) and the Supply Equipment Communication Controller (SECC) in the EV and the EVSE respectively. ISO 15118 is designed to provide a seamless method to authenticate, authorize, charge EVs, and bill their drivers without further interaction from the drivers. Additionally, ISO 15118 enables Smart Charging, which allows bidirectional energy to be transferred between the EVs and the power grid. With smart charging, energy can be transferred from the grid to the EVs based on their demands and the grid's capacity. The energy can likewise be transferred from the EVs back to the grid to support the grid during peak times.

Before the communication using ISO 15118 takes place, it employs an analog Pulse Width Modulation (PWM) signal protocol, defined by the IEC 61851 standard, to initiate the communication between the EV and EVSE via the Control Pilot (CP) pin of the charging plug (Figure 4).

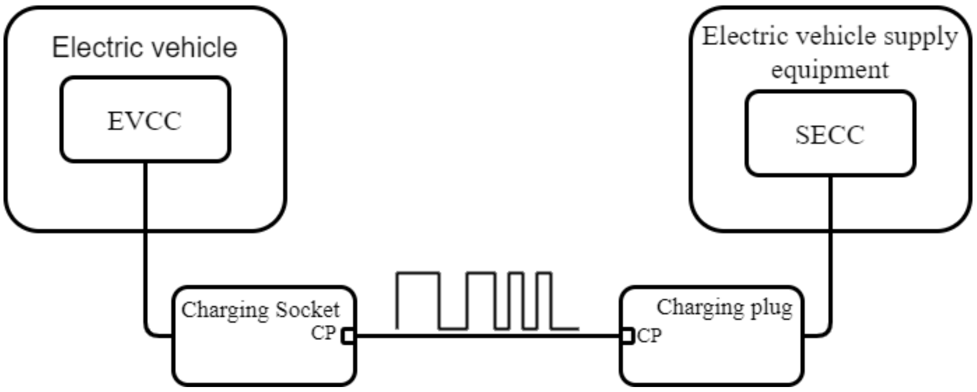


Figure 4: PWM communication based on IEC 61851

IEC 61581 specifies the charging states and the output power of a charging session by controlling the duty cycle and the amplitude of the PWM signal. The EVCC controls the states of the charging session by changing the signal's amplitude. Meanwhile, by modifying the signal's duty cycle, the SECC notifies the maximum current it can provide to the EVs. The states of the communication are described in Table 1.

Table 1: States of PWM communication according to IEC 61851 (DIN, 2019)

State A (+12V)	EV is not connected to the charging station.
State B (+9V)	EV is connected to the charging station and not ready for charging.
State C (+6V)	EV is connected to the charging station and ready for charging without the ventilator requested.
State D (+3V)	EV is connected to the charging station and ready for charging, with a ventilator requested.
State E (0V)	Problem with the power grid.
State F (-12V)	The charging station is not available.

By implementing only IEC 61851 communication, the EVs can be fully charged with the maximum current provided by the charging station. However, the purpose of a seamless charging protocol and enabling smart charging is defeated because only minimal data, i.e., charging states and maximum charging current, is shared between EVs and EVSEs. Therefore, under the following conditions:

- Both EV and EVSE support ISO 15118.
- IEC 61851 communication is at state B in Table 1.
- The duty cycle of the PWM signal is set to five percent by the EVCC.

The communication between the EV and EVSE is switched to ISO 15118. After the transition, the PWM signal is replaced by the Power Line Communication (PLC) protocol and the ISO 15118 protocol stack is initiated. The communication switches from analog to digital, IP-based form (Figure 5). The detail of the ISO 15118 protocol stack is described in the next section.

2.3.1 Characteristics of ISO 15118

Figure 5 describes ISO 15118 according to the OSI model. The Physical and Data link layer is managed by Signal Level Attenuation Characterization (SLAC) and HomePlug Green PHY (HPGP) protocol. The HPGP protocol controls the PLC communication between EVCC and SECC. The SLAC protocol defines the mechanism to ensure that the EV communicates to the correct charging station. The Network and Transport layer of the OSI model utilize the popular Internet Protocol version 6 (IPv6), Transmission Controller Protocol (TCP), User Datagram Protocol (UDP), and Transport Layer Security (TLS) protocols. The Session layer is occupied by the Vehicle-2-Grid Transfer Protocol (V2GTP) protocol, which manages the messages exchange session between EVCC and SECC. In the Presentation layer, the Efficient XML Interchange format (EXI) format is implemented to encode and decode the messages exchanged by the Application layer. EXI is the binary Extensible Markup Language (XML) format developed by WC3, which is an effort to optimize the performance and reduce the memory footprint of the XML format, to make it suitable for embedded systems.

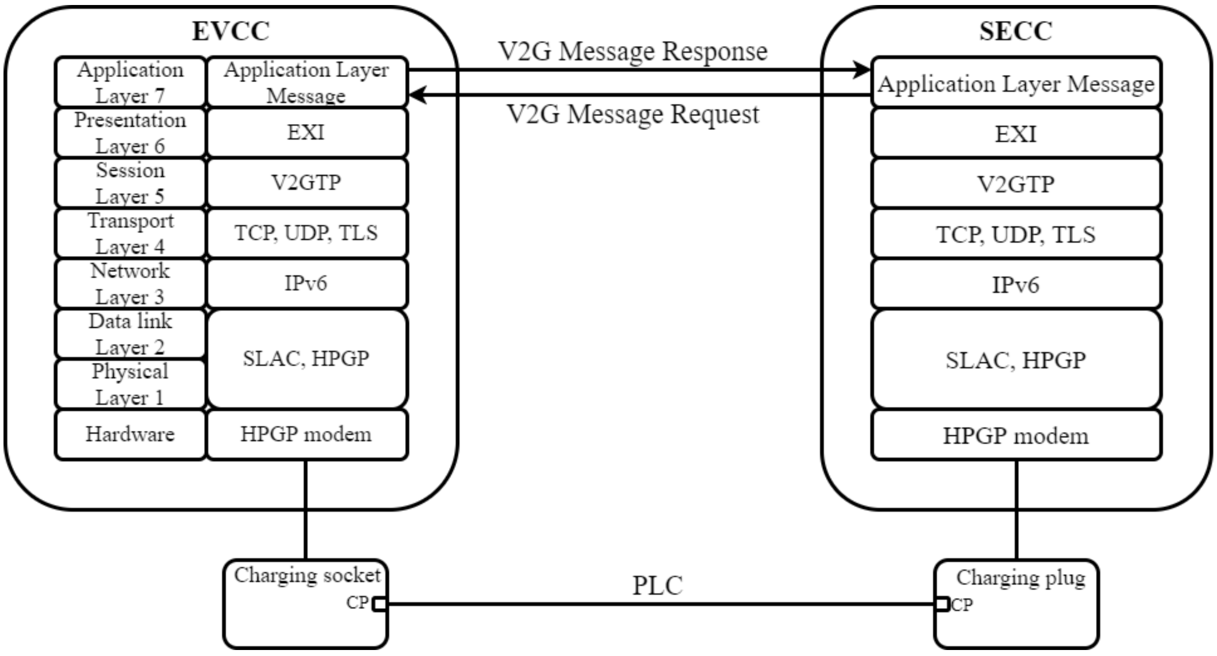


Figure 5: ISO 15118 protocol according to OSI model

Finally, the Application layer is responsible for exchanging messages with its counterpart. The Application layer works as the client-server messaging model, where EVCC is the client sending requests to the server and SECC is the server responding to the client's requests.

2.3.2 ISO 15118 Message

ISO, 2014 specifies that each ISO 15118 message has two parts: the header and the body (Figure 6). The ISO 15118 header provides the general information of the message. Each ISO 15118 header has minimum one element – the Session ID. The Session ID has maximum eight bytes and is presented in hexadecimal format. Its purpose is to identify the ISO 15118 message's communication session. This value is established during the communication setup phase, and every ISO 15118 message header must use this ID until the end of the charging session. When errors occur during message decoding on the EVSE side, the Notification element is added to the header to notify its counterpart about the errors. Both the Session ID and the Notification is defined in detail by the ISO 15118 XML Schema Definition (XSD) document. If a ISO 15118 message is required to be signed, the Signature element, specified by WC3, 2002, is included in the header. The body of the ISO 15118 message consists of the one or multiples elements that hold the parameters related to a specific ISO 15118 message defined in the ISO 15118 XSD document. For example, ISO, 2014 specifies that the SessionSetupReq message's body holds only one element called EVCCID, containing the ID of the EVCC.

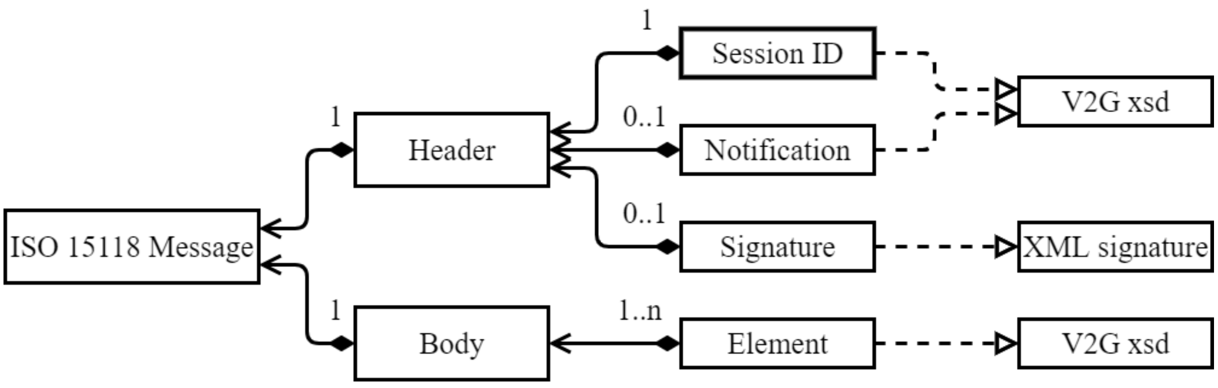


Figure 6: ISO 15118 message model

2.3.3 ISO 15118 Message Sequences

A charging session is divided into eight sequences by ISO, 2019, and within an individual sequence, the V2G messages exchanged between EVCC and SECC are defined. Because DC charging requires more complex control than AC charging, in some use cases, there is a difference regarding the messages between them. For example, in the sequence Target setting and charge scheduling, DC charging requires two extra message pairs, *CableCheck-Request/Response* and *PreChargeRequest/Response*, to check for the status of the connector and the charging parameters before the energy delivery. Furthermore, in DC charging, before ending the charging session, the message pair *WeldingDetectionRequest/Response* is used to check the status of the connector before the connector is unlocked from the EV. Table 2 lists the eight sequences, the messages in each sequence, and the difference between AC and Direct Current (DC) charging regarding message usage.

Table 2: ISO 15118 message sequences (ISO, 2014)

Sequence name	DC charging	AC charging
Start of communication	No messages are exchanged, IEC 61851 hands over the communication to ISO 15118	
Communication setup	supportedProtocolApplicationReq/Res SessionSetupReq/Res	
Certificate handling	ServiceDiscoveryReq/Res PaymentServiceSelectionReq/Res CertificateInstallationReq/Res(Optional)	
Identification, authentication, authorization	PaymentDetailReq/Res AuthorizationReq/Res	
Target setting and charge scheduling	ChargeParameterDiscoveryReq/Res	
	CableCheckReq/Res	N/A
	PreChargeReq/Res	N/A
	PowerDeliveryReq/Res	
Charge controlling and rescheduling	CurrentDemandReq/Res	ChargingStatusReq/Res
	MeteringReceiptReq/Res	
Value-added services	ServiceDetailReq/Res (Optional)	
End of the charging process	PowerDeliveryReq/Res	
	WeldingDetectionReq/Res	N/A
	SessionStopReq/Res	

For the details of the charging session, please also refer to the sequence diagram in Appendix A.

2.3.4 SEVENSTAX's ISO 15118 Implementation

In the context of this thesis, the construction of the SEVENSTAX's ISO 15118 stack is examined to give the audience an abstract view of how the ISO 15118 stack can be implemented. The overview lays the foundation for the bridging application development section. SEVENSTAX's ISO 15118 stack is developed in C programming language and it is intended to be deployed in the embedded systems. The stack relies on TCP/IP network stack and the PLC chip driver, which are also developed in-house. SEVENSTAX's V2G stack supports both EVSE and EV modes, but only the EVSE configuration is in the scope of the thesis.

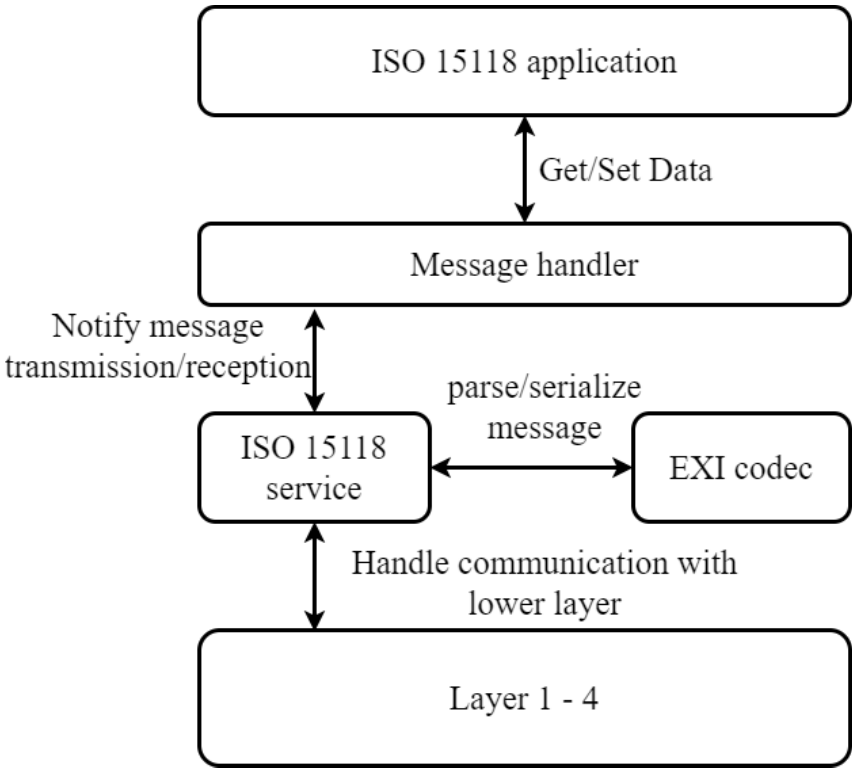


Figure 7: SEVENSTAX's V2G stack

The SEVENSTAX's ISO 15118 stack is divided into three components: ISO 15118 service, Message handler, and EXI codec (Figure 7). The ISO 15118 service is responsible for handling the connections with its counter part, and it also distributes C-structure messages to the upper layers and EXI messages to the lower layer. The EXI codec is accountable for parsing the ISO 15118 message to C-structure, which can be processed by the upper layer. Additionally, the EXI codec converts the messages in C-structure from the upper layer to the corresponding ISO 15118 message in EXI format, which is going to be sent away by the V2G service. The Message handler's responsibilities are getting and setting the ISO 15118 messages' parameters from or to the ISO 15118 application. The ISO 15118 application is intended to be implemented by the ISO 15118 stack developers, and this application must provide the data to or process the data from the Message handler according to the message sequence (Figure 32) of the charging session.

2.4 OCPP 2.0.1

OCPP is the de facto standard for communication between the charging station and the CSMS. The current version of OCPP is 2.0.1. The purposes of OCPP are to define an open and secured communication protocol over the internet, to support data exchange between charging stations and backend systems. With the support of ISO15118, OCPP can enable smart charging capability by managing and controlling the charging profile of the charging stations based on drivers' demand and the electric grid's situation.

2.4.1 Characteristics of OCPP

Similar to ISO 15118, OCPP is also an IP-based protocol, which relies on TCP as the transport protocol and TLS for authentication and encrpyted communication (Figure 8). The first difference between ISO 15118 and OCPP is that ISO 15118 uses PLC and SLAC protocol for Physical and Data link layer, while OCPP is based entirely on Ethernet communication.

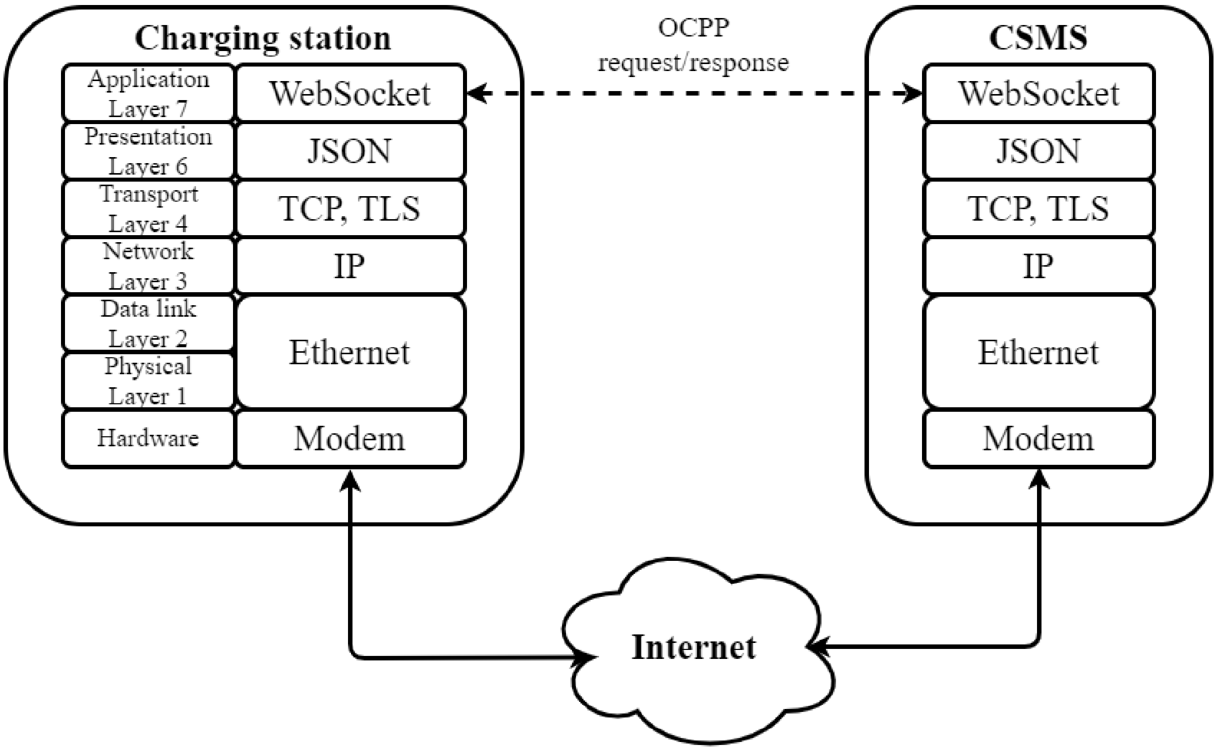


Figure 8: OCPP protocol stack based on OSI model

Unlike ISO 15118, which uses a proprietary protocol and EXI for exchanging data on the application layer, OCPP uses the well-known WebSocket protocol as an application protocol and JavaScript Object Notation (JSON) format to encode OCPP messages. Open Charge Alliance, 2020d specifies OCPP functions as a server-client communication protocol, where the charging station is the client, and CSMS is the server. Due to the nature of WebSocket, which is a full duplex communication, the client and the server can simultaneously send data to other parties. Consequently, to create the request and response behavior, Open Charge Alliance, 2020d defines a Remote Procedure Call (RPC) framework to coordinate the messages exchange. The fundamentals of the RPC are described as follows:

- An entity (charging station or CSMS) sends a request to its counterpart and waits for a response or an error message.
- While waiting for the response, that entity is not allowed to send a new request until the response arrives or a timeout interval elapses

Another distinction between ISO 15118 and OCPP is their messages. The V2G message is binary based, while the OCPP message is transferred using plain-text JSON. The structure of the OCPP message is discussed in the following subsection.

2.4.2 OCPP Messages

Open Charge Alliance, 2020d specifies three OCPP message types: CALL (request), CALLRESULT (response), and CALLERROR (Figure 9). Each message type can be identified by its Message Type ID, which is defined by Open Charge Alliance, 2020d as follows:

- The CALL has ID 2
- The CALLRESULT has ID 3
- The CALLERROR has ID 4

A unique Message ID is assigned to every CALL (request) so that its CALLRESULT (response) can be identified later. Therefore, if the CALL and the CALLRESULT belong to the same request/response, they must have the exact Message ID. The Action element only exists in the CALL message and has the string value indicating the payload of the message. The payload of the message is defined by Open Charge Alliance, 2020c. The Action element is omitted from the CALLRESULT because the CALL and CALLRESULT are always in pairs and can be identified using the Message ID. Therefore, including the Action element in the CALL is already sufficient. For instance, Figure 34 illustrates the Boot Notification CALL and CALLRESULT, including their payload's content.

When errors occur during communication, such as network connection, service availability, or message decoding, the recipient can respond to the sender with the CALLERROR. Open Charge Alliance, 2020d specifies that the CALLERROR consists of five elements: Message Type ID, Message ID, Error code, Error Description, and Error detail. CALLERROR's Message Type ID has the value four (4), and its Message ID must, again, be the same as the Message ID of the CALL message so that the receiver can match the CALLERROR to the CALL. The Error description and Error detail are optional elements allowing the receivers to describe the error in detail. Listing 1 is an example of a CALLERROR message notifying the recipient that SetDisplayMessageRequest is not supported.

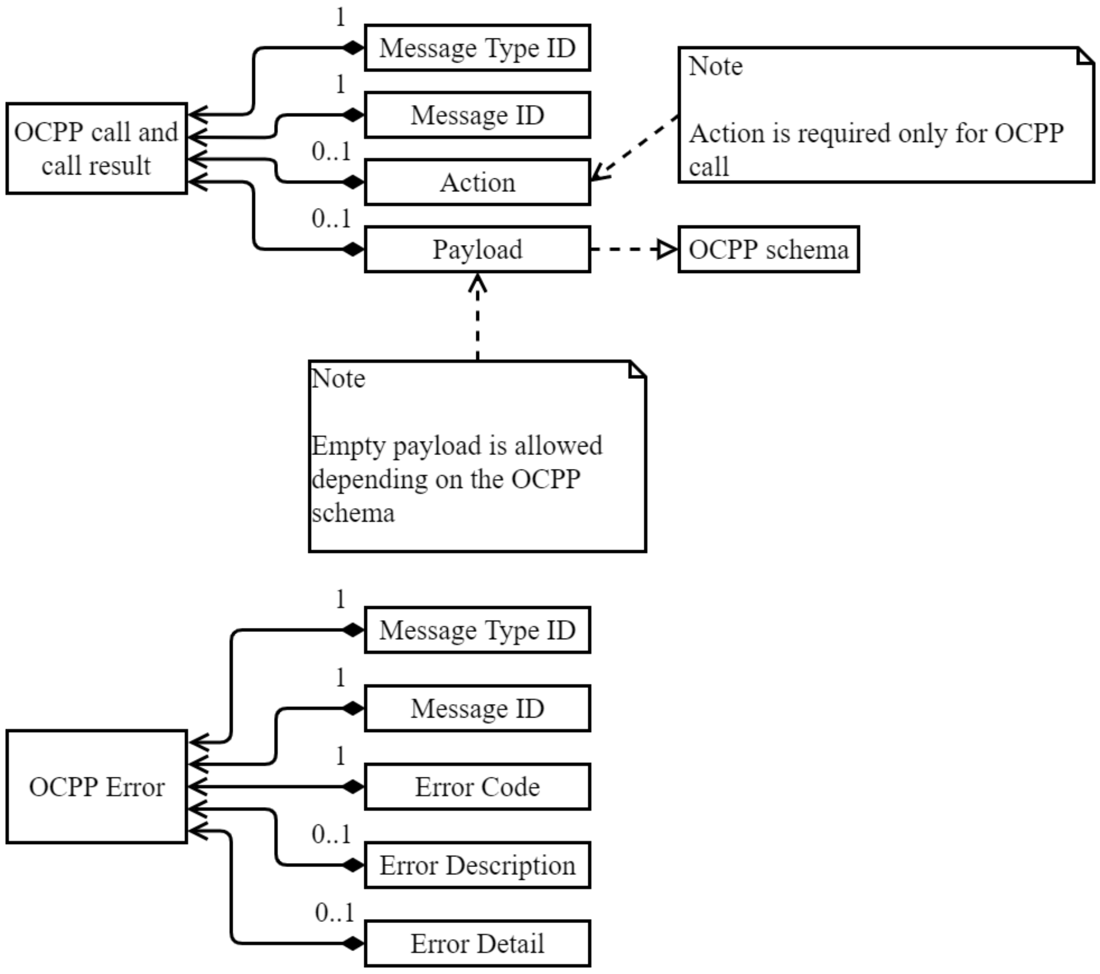


Figure 9: Message model of OCPP messages

2.4.3 OCPP Use Cases

Open Charge Alliance, 2020a groups the OCPP messages into use cases, and the use cases are grouped into sixteen blocks based on functionalities. Open Charge Alliance, 2020b illustrates each use case with a sequence diagram to indicate the direction of communication and the type of messages being exchanged. Additionally, each use case is accompanied by a list of requirements to define the expected behaviors. For example, Figure 33 illustrates the use case B03 – Cold Boot Charging Station – Rejected
Since Open Charge Alliance, 2020a) specifies various use cases, within the scope of this thesis, only the use cases related to ISO 15118 are listed and discussed. Table 3 lists the use cases related to ISO 15118.

Table 3: OCPP use cases related to ISO 15118, Open Charge Alliance, 2020b

Functional block	Use case ID	Use case name
Authorization	C07	Authorization using Contracts Certificate
	C08	Authorization at EVSE using ISO 15118 EIM
Transaction	E15	End of the charging process
Remote Control	F04	Remote Stop ISO 15118 Charging from CSMS
Meter Values	J03	Signed Meter Values
Smart Charging	K15	Charging with load leveling based on High-Level Communication
	K16	Renegotiation initiated by CSMS
	K17	Renegotiation initiated by EV
ISO15118 CertificateManagement	M01	Certificate installation EV
	M02	Certificate Update EV
	M03	Retrieve a list of available certificates from a Charging Station
	M04	Delete a specific certificate from a Charging Station
	M05	Install the CA certificate in a Charging Station
	M06	Get V2G Charging Station Certificate status

2.4.4 SEVENSTAX's OCPP Implementation

In the scope of this thesis, only the OCPP client is discussed because a bridging application is built in the later chapter based upon its interaction with the ISO 15118 stack. As with the ISO 15118 stack, the OCPP stack is developed using C-programming language and intended to be used in embedded systems. The stack comprises three components: the OCPP Service, the OCPP Controller, and the OCPP Message codec (Figure 10). The Controller component manages the WebSocket connection to the CSMS and the message routing between the Service component and lower network layers. Upon receiving or transmitting an OCPP message, the Controller component invokes the Message codec component to transform the OCPP message into a corresponding format. When receiving an OCPP message, the Message codec parses this JSON message into the C-structure, then the C-structure data is forwarded to the Service by the Controller. Before the Controller component sends any message to the CSMS, the Message codec component serializes the C-structure data to JSON format. The Service component is responsible for notifying the OCPP Application about the arrived messages and preparing the resources to serialize the transmitted messages. The OCPP Application is intended to be developed by OCPP stack users. It processes the messages from the Service component and responds with a reply according to the use cases defined by Open Charge Alliance, 2020b.

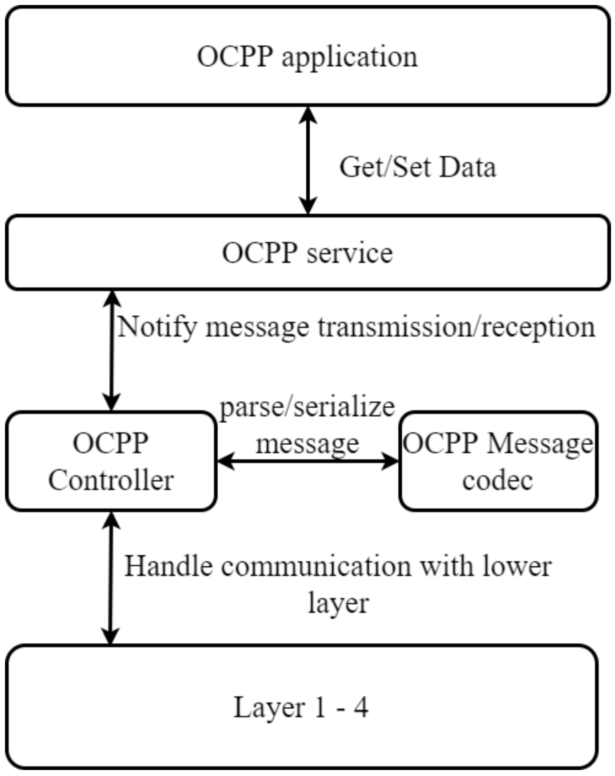


Figure 10: SEVENSTAX's OCPP stack

2.5 Interaction Between ISO15118 and OCPP 2.0.1

In this chapter, a combined sequence diagram of ISO 15118 and OCPP, derived from Sections 2.3.3 and 2.4.3, is presented to illustrate the message exchange between both protocols. Due to various scenarios supported by ISO 15118 and OCPP, the following points are discussed:

- The conducted (wired) communication of ISO, 2014 is analyzed.
- ISO 15118-20 is not analyzed because it is still under construction.
- It is assumed that the EV uses a contract certificate, which is not yet installed in the vehicle, as a payment method. Therefore, the EV must request the contract certificate from the charging station.
- During the charging session, the charging profile renegotiation may be triggered by either the EV or CSMS. Both re-negotiations are similar in terms of messages exchanged, only the triggering source is different. Therefore, only the later is shown and analyzed.
- From OCPP, only the use cases related to ISO 15118 are presented and evaluated.

The following notation for addressing the messages and their elements is introduced to keep the information unified, readable, and easy to look up.

- The message's name is written using Pascal case, which means the words in its name are compounded, and the first letter of each word is written in uppercase, for example, BootNotificationRequest or CertificateInstallationRequest.
- The elements in each message are also denoted using the Pascal case. Because an element can contain other sub-elements, to denote a sub-element, the following notation is applied: Element.SubElelement.SubSubElement, for example, EVChargeParameter.EAmount.
- The element's data type is presented as follows: Element.SubElelement:datatype, for example, EVChargeParameter.EAmount:PhysicalValueType
- The value of an element is written in italics, for example, *Occupied*.

Figure 11 shows the deployment of the charging-protocol-related software components in the EV, Charging station, and CSMS. . In the scope of the thesis, only the those components are of interest. Other components, e.g., the GUI, metering, or card reader, are omitted from the diagram.

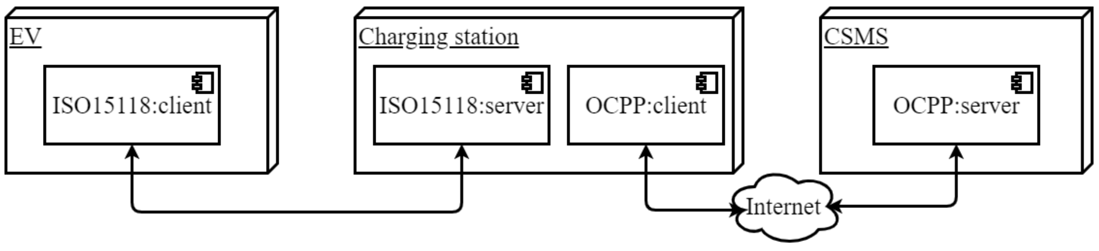


Figure 11: Deployment diagram of ISO 15118 and OCPP software component

In the EV entity, the ISO 15118 component acts as an ISO 15118 client, which sends ISO 15118 requests to its counterpart - the ISO 15118 server - in the Charging station. The charging station hosts two charging protocol components, an ISO 15118 component acting as a server and an OCPP component acting as a client. The ISO 15118 server handles the V2G requests from the EV and controls the charging process according to the required charging parameters. On the other hand, the OCPP client reports the status of the charging station and forwards the data from the ISO 15118 client to the CSMS. The CSMS manages its charging station using the OCPP server component. In the case of Smart Changing, the CSMS controls the power delivered to the EVs by sending OCPP messages containing the new charging profiles to the designated charging station. The four mentioned components are the four actors of the sequence diagram, which is discussed in the following sections.

2.5.1 Communication Setup

As shown in Figure 12, after booting up, the OCPP client in the charging station sends the BootNotificationRequest containing the boot reason and charging station information to the server. The OCPP server responds with the message containing the status *Accepted*, the server's current time, and the heartbeat's interval. When receiving a status other than *Accepted*, the client retries to send the request after an interval specified by the heartbeat's interval. After the BootNotification message, the OCPP client sends the status of its connectors, which are the electrical outlets providing the energy to the EV, to the server with the StatusNotificationRequest. The OCPP server acknowledges the request with the StatusNotificationResponse. From this point, the client periodically sends the HeartbeatRequest to the server to indicate the liveliness of the connection. The server confirms the request with a response containing the server's timestamp which can be used as a source for time synchronization between the client and the server. The interval between two Heartbeat requests is specified in the BootNotificationResponse. When a connector is connected to an EV, the charging station sends another StatusNotificationRequest containing the status *Occupied* to the server to indicate an EV connected to the charging station. At this point, the EV is performing the IEC 61581 communication with the EVSE.

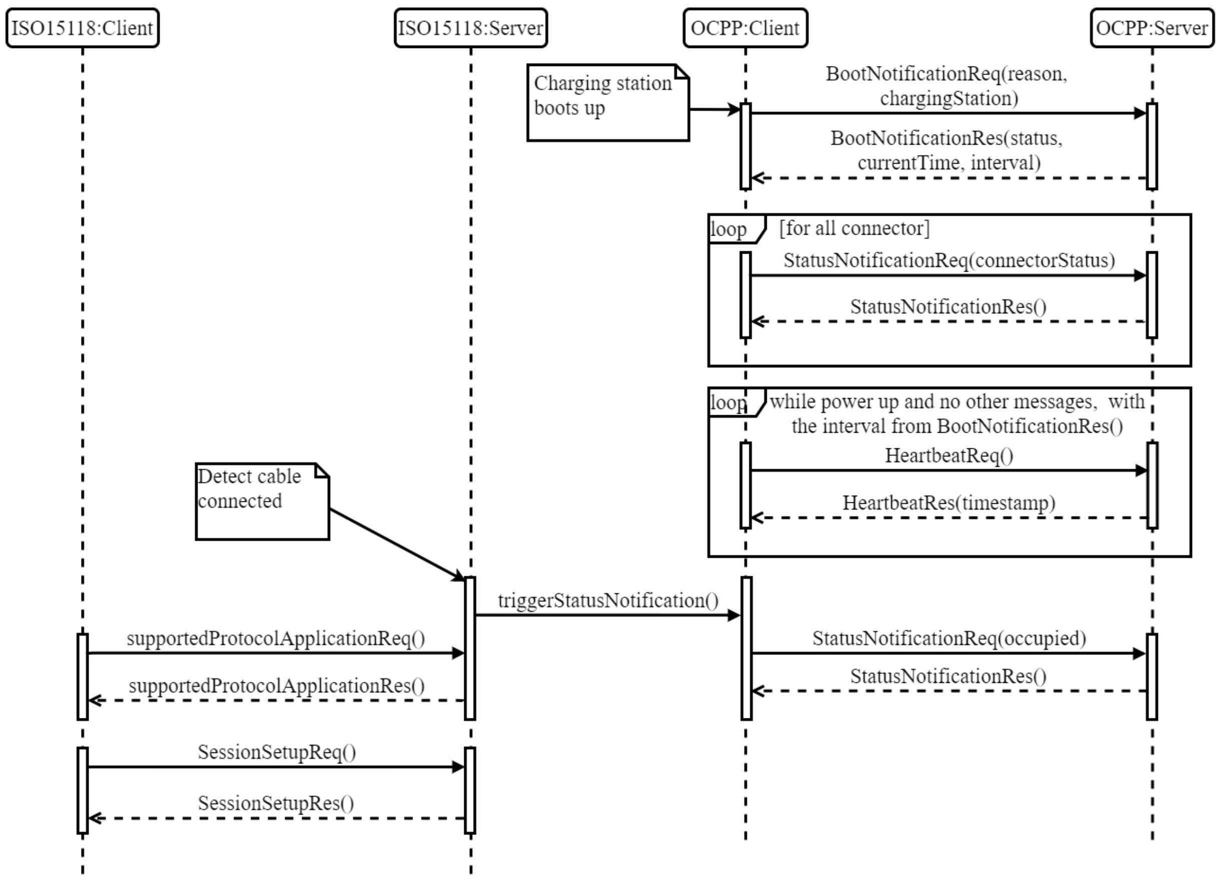

ISO15118:Client
ISO15118:Server
OCPP:Client
OCPP:Server
Charging station boots up
BootNotificationReq(reason, chargingStation)
BootNotificationRes(status, currentTime, interval)
loop [for all connector]
StatusNotificationReq(connectorStatus)
StatusNotificationRes()
loop while power up and no other messages, with the interval from BootNotificationRes()
HeartbeatReq()
HeartbeatRes(timestamp)
Detect cable connected
triggerStatusNotification()
supportedProtocolApplicationReq()
supportedProtocolApplicationRes()
StatusNotificationReq(occupied)
StatusNotificationRes()
SessionSetupReq()
SessionSetupRes()

Figure 12: The communication setup phase

After the communication is switched to ISO 15118, the ISO 15118 client on the EV sends the SupportedProtocolApplicationRequest to the server located in the charging station. The SupportedProtocolApplicationRequest specifies which types of protocol are supported by the client. The server replies to the client with the SupportedProtocolApplicationResponse containing the most suitable protocol it can support and the ResponseCode OK_SuccessfulNegotiation. If the client and the server agree upon the protocol, they establish the communication session. As mentioned in Section 2.3.2, each ISO 15118 message requires an ID to identify the communication session, but for the SessionSetupRequest message, its header contains the SessionID with the value *0x0000000000000000*, and its body contains the MAC address of the EVCC. The response header contains a SessionID, which is used for the rest of the communication session. Additionally, the response message's body contains the ResponseCode OK_NewSessionEstablished and the EVSEID. After the communication is established, the EV and the charging station perform the authorization and authentication. Up to this point, no charging parameters have been exchanged between ISO 15118 and OCPP protocol.

2.5.2 Identification, Authorization, and Authentication

After the communication is established, the EV sends the ServiceDiscoveryRequest to the charging station to query for the services provided by that charging station. The ServiceDiscoveryRequest has two optional elements: ServiceScope and ServiceCategory, which are used by the EV to limit the types of services in the response message. The charging station processes the ServiceDiscoveryRequest and responds with the ServiceDiscoveryResponse holding the following parameters containing the provided services: ResponseCode, PaymentOptionList, ChargeServiceList, and ServiceList. The EV may send the ServiceDetailRequest to inquire about the detail of the services provided in the ServiceDiscoveryResponse. The charging station responds to the EV with a ServiceDetailResponse, including the information about the inquired services. With the information from the previous messages, the EV chooses the payment service by sending the PaymentServiceSelectionRequest to the server. In this message, the EV specifies which type of service it wants to use for payment, e.g., using a contract certificate or external method. After processing the request, the station acknowledges the EV with the PaymentServiceSelectionResponds.

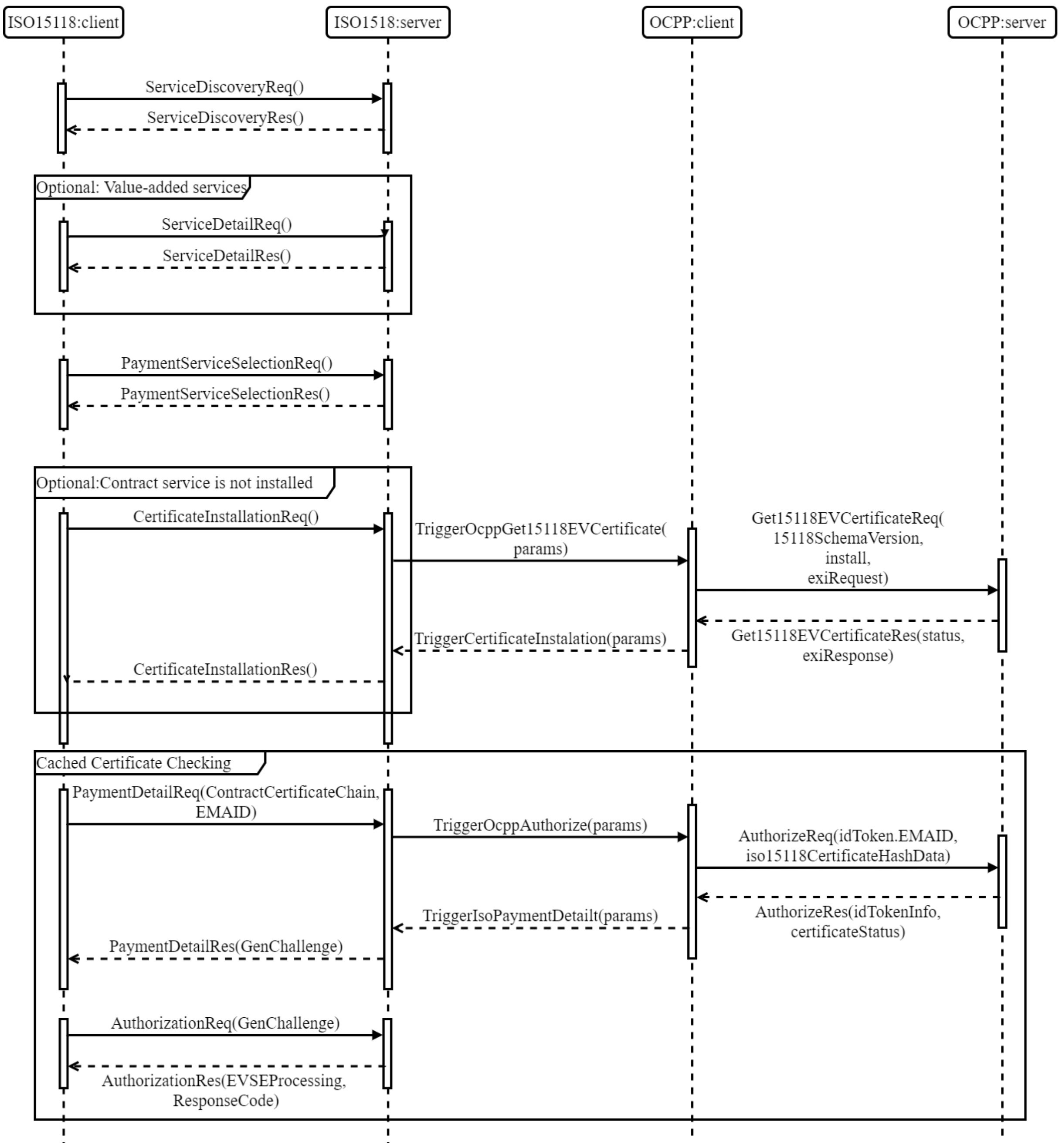

ISO15118:client
ISO1518:server
OCPP:client
OCPP:server
ServiceDiscoveryReq()
ServiceDiscoveryRes()
Optional: Value-added services
ServiceDetailReq()
ServiceDetailRes()
PaymentServiceSelectionReq()
PaymentServiceSelectionRes()
Optional:Contract service is not installed
CertificateInstallationReq()
TriggerOcppGet15118EVCertificate(params)
Get15118EVCertificateReq(15118SchemaVersion, install, exiRequest)
Get15118EVCertificateRes(status, exiResponse)
TriggerCertificateInstalation(params)
CertificateInstallationRes()
Cached Certificate Checking
PaymentDetailReq(ContractCertificateChain, EMAID)
TriggerOcppAuthorize(params)
AuthorizeReq(idToken.EMAID, iso15118CertificateHashData)
AuthorizeRes(idTokenInfo, certificateStatus)
TriggerIsoPaymentDetailt(params)
PaymentDetailRes(GenChallenge)
AuthorizationReq(GenChallenge)
AuthorizationRes(EVSEProcessing, ResponseCode)

Figure 13: Identification, authentication, and authorization phase

Because this thesis assumes that the EV uses the contract certificate as the payment method and the contract certificate is not present in the vehicle, the EV sends the CertificateInstallationRequest message to the charging station to issue the installation of the contract certificate into the EV. The CertificateInstallationRequest's payload contains is signed with the private key of the Original Equipment Manufacture (OEM) Provisioning Certificate so that it can be verified for the genuineness by the server. The body of this message consists of the OEM Provisioning Certificate, whose public key is employed to verify the genuineness of the request. In addition to the OEM Provisioning Certificate, the

CertificateInstallationRequest also includes the ListOfRootCertificateIDs containing the IDs of the root certificate installed in the EV. Upon receiving the CertificateInstallation-Request, the charging station sends the OCPP GetEV15118CertificateRequest message to the CSMS. This OCPP request contains three elements: the Iso15118SchemaVersion, the Action holding value *Install*, and the ExiRequest holding the complete Certificate-InstallationRequest message. The CSMS processes the request and replies with the GetEV15118CertificateResponse containing the Status of the operation (*Accepted*) and the ExiResponse. In case of not receiving the status *Accepted*, paymen using a contract certificate is not possible and the charging station must use a fallback solution, for example, authorization using External Identification Means (EIM). As soon as the charging station receives the OCPP response, it sends CertificateInstallationResponse, which is the ExiResponse element of the GetEV15118CertificateResponse, to the EV. The Certificate-InstallationResponse consists of the ResponseCode, the SAProvisioningCertificateChain, ContractSignatureCertChain, ContractSignatureEncryptedPrivateKey, DHpublickey, and the eMAID. Table 4 explains the meaning of the elements mentioned above.

Table 4: Element of the CertificateInstallationResponse

ResponseCode	Indicating the acknowledgment of the status of the message.
SAProvisioningCertificateChain	The certificate chain for verifying the signature in the message header.
ContractSignatureCertChain	The certificate chain has to be installed in the EV for signature purposes.
ContractSignatureEncryptedPrivateKey	The private key belongs to the ContractSignatureCertChain and is encrypted based on the OEM Provisioning Certificate and the DHPublickey.
DHpublickey	The Diffie Hellman public key to generate the session key to decrypt the ContractSignatureEncryptedPrivateKey.
eMAID	An identifier to identify the ContractSignatureCertChain.

After receiving the Contract certificate, the EV sends the PaymentDetailRequest to the charging station to start the authentication process. The PaymentDetailRequest message comprises the eMAID and the ContractSignatureCertChain, which the EVs received in the previous request. If the charging station is online and connects to the CSMS, it combines the information from the ContractSignatureCertChain and its locally stored data to create the OCPP AuthorizeRequest message to send to the CSMS. The exchanged elements between OCPP and ISO 15118, mentioned in the above section, are listed in

Table 15. The CSMS verifies the data in the request and responds with the AuthorizeResponse message to the charging station. This OCPP response holds the status of whether the provided certificates are verified. Upon receiving the OCPP response, the charging station replies to the EV with the PaymentDetailResponse message containing the ResponseCode, the GenChallenge, and the EVSETimestamp. The GenChallenge element is a randomly generated number acting as a challenge to prevent the relay attack during the authentication process. The EVSETimestamp acts simply as a time source for time synchronization in the EV. Table 16 lists the elements shared between ISO 15118 and OCPP. The EV signs the body of the ISO 15118 AuthorizationRequest holding the same GenChallenge with the private key, which it obtains in CertificateInstallationResponse, and sends the request to the charging station. The charging station verifies the signature in the request and sends the ISO 15118 AuthorizationResponse containing the verification status back to the EV. The verification status can be *OnGoing* to indicate that the charging station is still processing the data, or it can be *Finished* to indicate that the authorization is successful. When the EV receives the status *Finished*, it is ready to proceed to the charging phase.

2.5.3 Target Setting and Charge Scheduling

In this phase, the EV requests a charging profile from the charging station. The charging station, in turn, inquires about a charging profile from the CSMS if it has a connection to the CSMS . In case of no connection to the CSMS, the charging station can use fallback solutions, for example, using default profiles or simply providing the requested energy. After receiving a suitable charging profile and before the energy is provided, the EV performs the functionality checks, i.e., cable check and pre-charge check. Figure 14 illustrates the message sequence of setting and scheduling the charging profile. Firstly, it sends the ChargeParameterDiscoveryRequest to the charging station. This message typically consists of the RequestingEnergyTransferMode element indicating the charging types (AC or DC), and the EVChargeParamter element containing the EV's charging requirements, for example, minimum current, minimum voltage, energy amount, et cetera (Table 17). The EVChargeParemeter element's content depends on whether the charging mode is AC or DC. Upon receiving the request, the charging station extracts the information from it and creates the OCPP NotifyEVChargingNeedsRequest to send to the CSMS . The OCPP request holds the evseID, to which the EV connects, and the ChargingNeed extracted from the EVChargeParemeter. The mentioned elements are listed in detail in Table 17

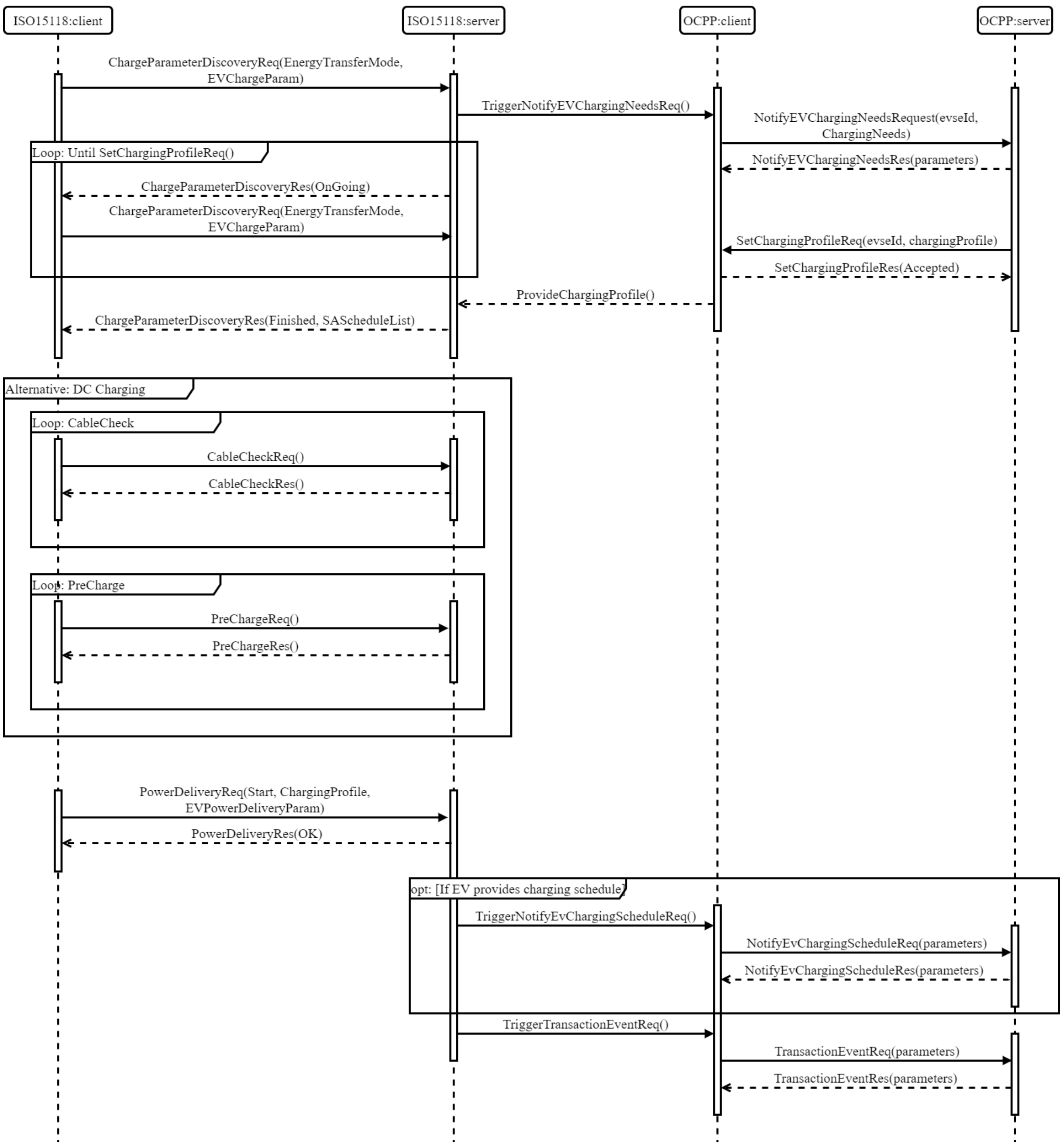

ISO15118:client
ISO15118:server
OCPP:client
OCPP:server
ChargeParameterDiscoveryReq(EnergyTransferMode, EVChargeParam)
TriggerNotifyEVChargingNeedsReq()
NotifyEVChargingNeedsRequest(evseId, ChargingNeeds)
Loop: Until SetChargingProfileReq()
NotifyEVChargingNeedsRes(parameters)
ChargeParameterDiscoveryRes(OnGoing)
ChargeParameterDiscoveryReq(EnergyTransferMode, EVChargeParam)
SetChargingProfileReq(evseId, chargingProfile)
SetChargingProfileRes(Accepted)
ProvideChargingProfile()
ChargeParameterDiscoveryRes(Finished, SAScheduleList)
Alternative: DC Charging
Loop: CableCheck
CableCheckReq()
CableCheckRes()
Loop: PreCharge
PreChargeReq()
PreChargeRes()
PowerDeliveryReq(Start, ChargingProfile, EVPowerDeliveryParam)
PowerDeliveryRes(OK)
opt: [If EV provides charging schedule]
TriggerNotifyEvChargingScheduleReq()
NotifyEvChargingScheduleReq(parameters)
NotifyEvChargingScheduleRes(parameters)
TriggerTransactionEventReq()
TransactionEventReq(parameters)
TransactionEventRes(parameters)

Figure 14: Target scheduling

The CSMS sends the NotifyEVChargingNeedsResponse to the station with the status *Accepted* to acknowledge the message. Right after sending the NotifyEVChargingNeedsRequest, the charging station replies to the EV with the ChargeParameterDiscoveryResponse. Since the charging station is still waiting for the charging profiles from the CSMS, the response message contains the EVSEProcessing element with the value *OnGoing* to indicate that the charging station is still processing the request. The charging station must perform this ping-pong message loop between the ChargeParameterDiscoveryRequest and the ChargeParameterDiscoveryResponse to prevent the timeout event from causing the

EV to terminate the communication session. When the CSMS is ready, it sends the OCPP SetChargingProfileRequest, holding the charging profiles. The charging station extracts the charging profile from the OCPP request and includes it in the ChargeParameterDiscoveryResponse to send to the EV. This time, the EVSEProcessing element of the ChargeParameterDiscoveryResponse has the value *Finished* to indicate that the negotiation is complete. The exchanged elements between the SetChargingProfileRequest and the ChargeParameterDiscoveryResponse are extracted and presented in Table 17.

In case of DC charging, the EV sends the CableCheckRequest to the charging station to request the cable check status at the charging station side and to inform the charging station that EV's connector is locked and the EV is ready to be charged. The checking process at the charging station side can last up to 40 seconds. In this case, the charging station responds to the EV with the CableCheckResponse containing its current DC_EVSEStatus and *Ongoing* status. The EV keeps sending the request until it receives the CableCheckResponse message with *Finished* status. After the cable check is completed, the EV sends the PreChargeRequest to the charging station to inform it about the required current and voltage. The charging station keeps adapting the output voltage and current according to the requested values and informs the EVs about the process by responding with the PreChargeResponse. This procedure can take up a couple of request/response until the charging station can provide the requested voltage and current. Before the charging can start, the EV sends the PowerDeliveryRequest to the charging station to request the energy delivered to the EV's battery. The delivered power must be conformed to the charging profiles negotiated in the ChargeParameterDiscoveryResponse message. The charging station replies to the EV with the PowerDeliveryResponse to confirm the requested charging energy, and it starts to provide the energy to the EV. As an optional step, the charging station can inform the CSMS about the selected charging profile by sending the OCPP NotifyEVChargingScheduleRequest. The CSMS acknowledges the requests by returning the NotifyEVChargingScheduleResponse. This acknowledgment means that the CSMS processes the request successfully. It does not state that the CSMS approves the charging profile. The charging station also sends the OCPP TransactionEventRequest to the CSMS to notify the charge station about the start of the charging. The request typically contains the event's type – Started, the time stamp, the trigger reason (i.e., the reason this message is being sent) – *ChargingStateChanged*, the sequence number, and the transaction information. The transaction information consists of detail about the transaction, e.g., transaction ID, charging state, and time spent on charging. The CSMS responds to the request with the TransactionEventResponse.

2.5.4 Charging Loop with Signed Metering Values

During the energy delivery phase, the EV periodically requests the charging status from the charging station, by which the EV can monitor the output power provided by the charging station. By regulation of some countries, the metering information record must be signed for billing purposes. Therefore, upon request, the EV sends the signed metering information to the charging station during the charging loop. Figure 15 displays the message sequence of the charging loop with signed metering values. For the AC charging case, the EV sends ChargingStatusRequest to the charging station. The charging station replies with the ChargingStatusResponse containing the EVSE's ID, the EVSE's status, the maximum provided current, and the ID of the charging profile the EVSE is using to provide the energy to the EV. If the charging station requires the metering information record to be signed, the ReceiptRequired element and the MeteringInfo element are included in the response, and the ReceiptRequired element is set to value true. For DC charging, the EV sends the CurrentDemandRequest to the charging station. The CurrentDemandRequest includes all the elements necessary for the charging station to verify the power it delivers to the EV. The charging station responds with the CurrentDemandResponse containing its charging status. Similar to AC charging, the ReceiptRequired element and the MeteringInfo element are included when signing metering information is required.

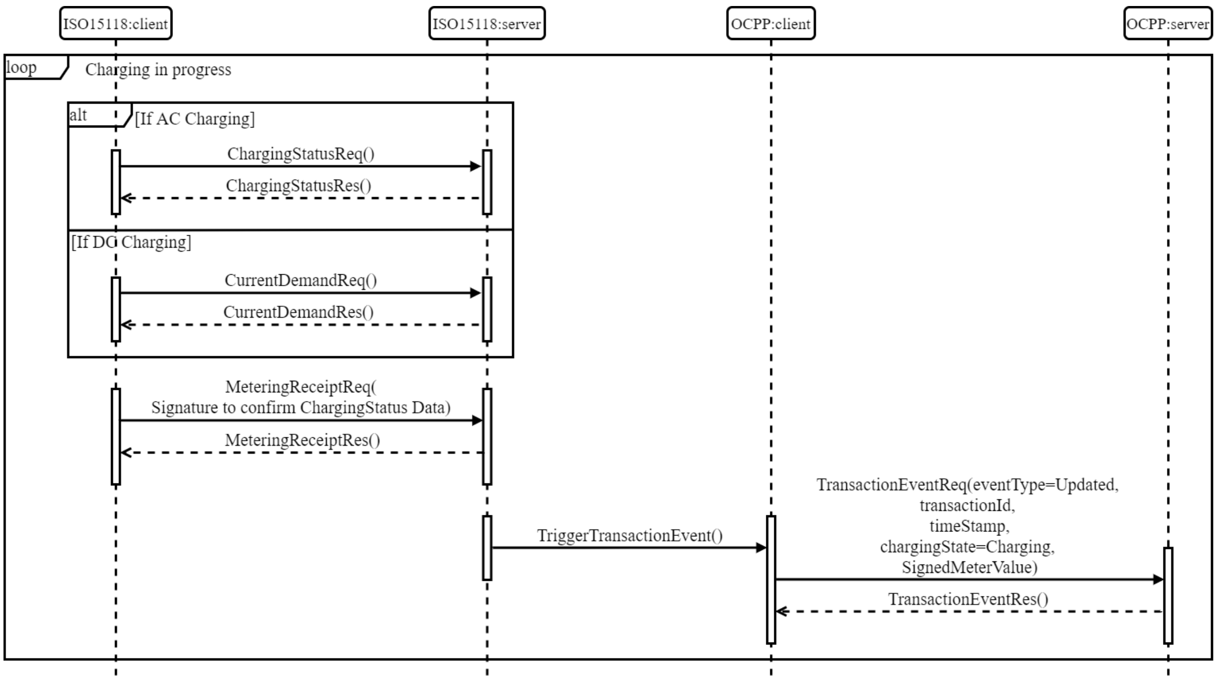


Figure 15: Charging loop with signed metering values

When signing metering information is required, the EV signs the body of the MeteringReceiptRequest with the private key from the contract certificate it obtains in the certificate installation phase. The charging station replies with the MeteringReceiptResponse to confirm the message reception. The charging station then sends the signed metering information to the CSMS by sending another OCPP TransactionEventRequest to the CSMS. This time, the TransactionEventRequest includes the event's type – Updated, the time stamp, the trigger reason (i.e., the reason this message is being sent) – Trigger, the sequence number, and the transaction information, and the signed metering information. The CSMS, again, confirms the message reception by responding to the EV with the TransactionEventResponse. The message sequence mentioned above is periodically repeated until the charging loop is complete. Table 20 illustrates the elements exchanged by ISO 15118 and OCPP.

2.5.5 CSMS Triggers the Charging Profile Renegotiation

During the charging loop, the CSMS could send a new charging profiles to the charging station according to the status of the power grid, for example, the charging station can provide more energy to the EV because the grid is not under high load (Figure 16). To initiate the charging profile, the CSMS sends the OCPP SetChargingProfileRequest to the charging station. This request consists of the new charging profile and the EVSE ID to which the charging profile is applied. The charging station acknowledges the request by sending the SetChargingProfileResponse. In the next ChargingStatusRequest, in case of AC charging, the charging station responds to the EV with the ChargingStatusResponse containing the notification *Renegotiation* to indicate that a new charging profile is going to be applied. In the case of DC charging, the same notification is applied to the CurrentDemandResponse. When receiving the response with the *Renegotiation* notification, the EV sends the PowerDeliveryRequest to trigger the renegotiation procedure. The charging station replies to the EV with the PowerDeliveryResponse to confirm the EV's request. Then, the EV sends the ChargeParameterDiscoveryRequest to get the new charging profile from the charging station. Since the charging station already receives the charging profile from the CSMS, it does not need to issue a new NotifyEVChargingNeedsRequest to the CSMS, but it can reply to the EV with the ChargeParameterDiscoveryResponse containing the new charging profile. Upon receiving the new charging profile, the EV requests power delivery with the PowerDeliveryRequest message containing the newly received charging profile. The charging station again confirms the request with the PowerDeliveryResponse. From this point, the charging loop can be started again, and energy is provided to the EV according to the new charging profile. Optionally, the charging station can inform the CSMS that the new charging profile is being used by sending the OCPP NotifyEvChargingScheduleRequest to the CSMS. The charging station receives the NotifyEvChargingScheduleResponse from the CSMS as an acknowledgment.

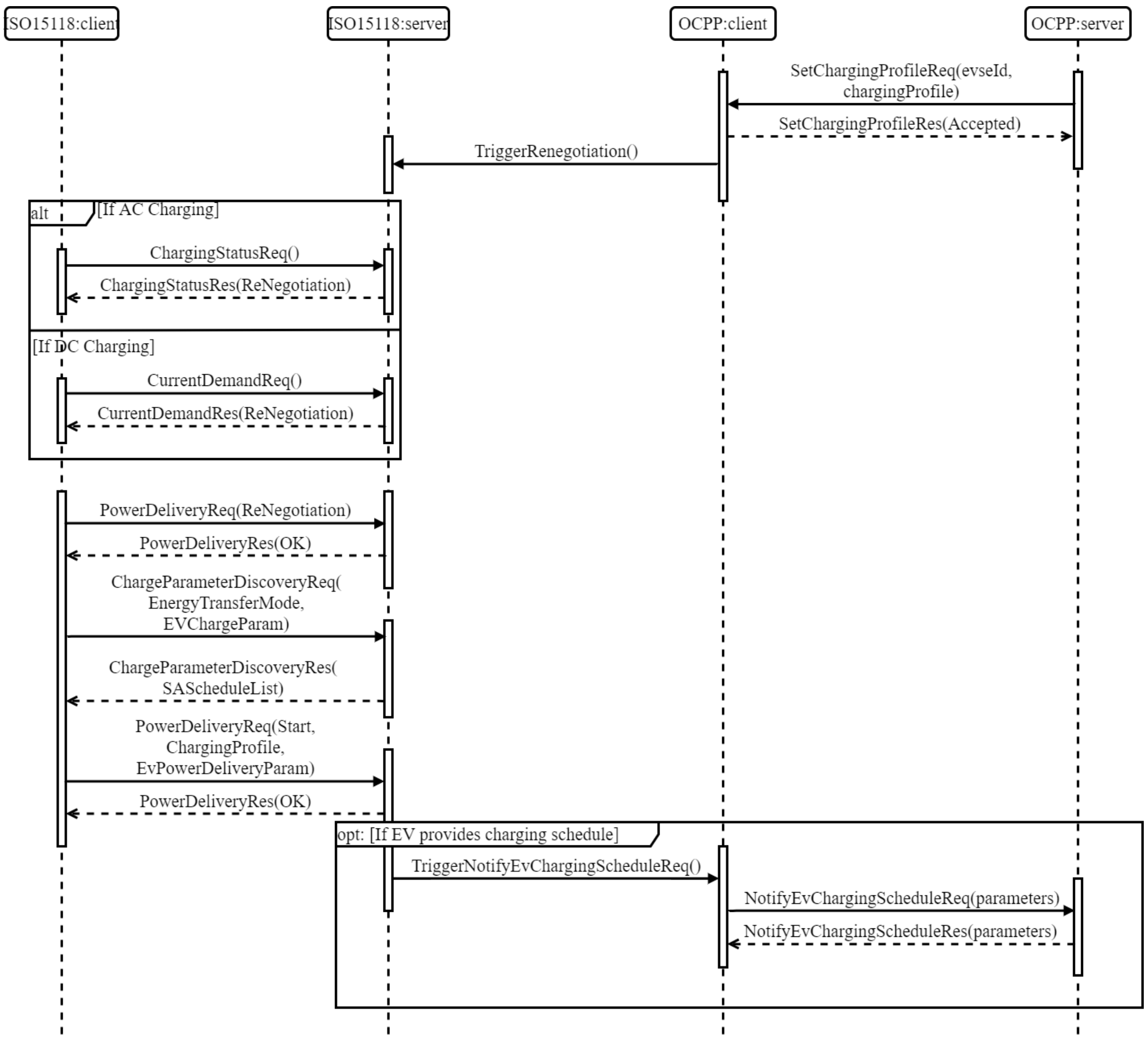

ISO15118:client
ISO15118:server
OCPP:client
OCPP:server
SetChargingProfileReq(evseId, chargingProfile)
SetChargingProfileRes(Accepted)
TriggerRenegotiation()
alt [If AC Charging]
ChargingStatusReq()
ChargingStatusRes(ReNegotiation)
[If DC Charging]
CurrentDemandReq()
CurrentDemandRes(ReNegotiation)
PowerDeliveryReq(ReNegotiation)
PowerDeliveryRes(OK)
ChargeParameterDiscoveryReq(EnergyTransferMode, EVChargeParam)
ChargeParameterDiscoveryRes(SAScheduleList)
PowerDeliveryReq(Start, ChargingProfile, EvPowerDeliveryParam)
PowerDeliveryRes(OK)
opt: [If EV provides charging schedule]
TriggerNotifyEvChargingScheduleReq()
NotifyEvChargingScheduleReq(parameters)
NotifyEvChargingScheduleRes(parameters)

Figure 16: CSMS renegotiates the charging schedule

2.5.6 End of Charging Session

When the EV's battery is fully charged, the EV sends the PowerDeliveryRequest containing the element ChargeProgress, which is set to value *Stop*. Upon receiving the request, the charging station responds with the PowerDeliveryResponse to confirm the message reception. If a wielding detection is requested according to the IEC 61851-23, the EV sends WeldingDetectionRequest to the charging station. The EV also opens the DC voltage paths' contacts and monitors the charging station's response. The charging station replies with the message WeldingDetectionResponse with the element EVSEPresentVoltage containing the value of the DC voltage on the DC paths. If the contacts are opened, the voltage has a value of *0*. Then, the EV can close the charging session by sending the message SessionStopRequest holding the element ChargingSession with the value *Terminate*. The charging station responds with the message SessionStopResponse to acknowledge the request. The charging station notifies the CSMS about the termination of the charging session by sending the OCPP message TransactionEventRequest with the element evenType having the value *Ended*. The CSMS replies to the request with the message TransactionEventResponse to confirm the charging session termination. The elements exchanged between OCPP and ISO 15118 are listed in Table 21.

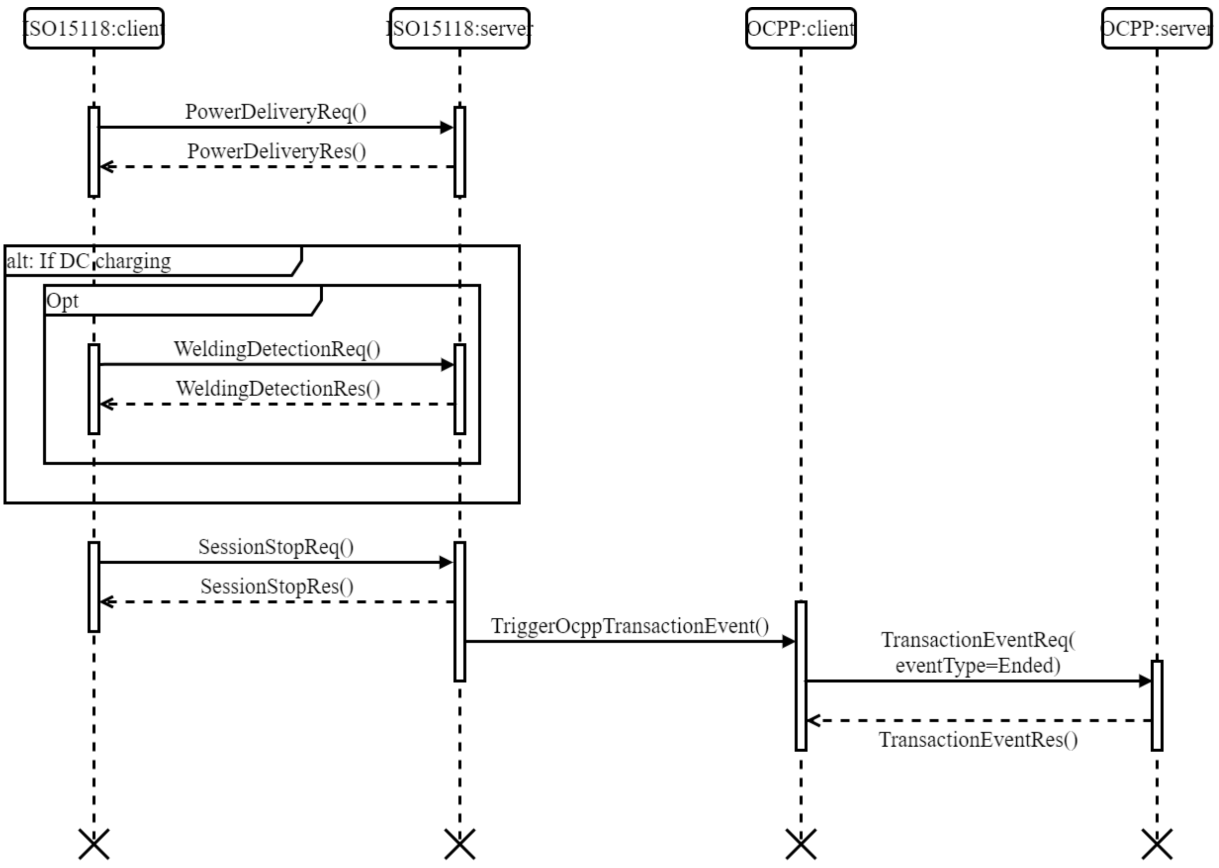


Figure 17: End of charging session

2.6 Chapter's Summary

This chapter provides an overview of several researches in the field of protocol compatibility analysis in terms of modeling method, algorithm, and result. Then, it reasons that the method proposed by the Ouederni et al., 2010 is suitable for analyzing the compatibility between ISO 15118 and OCPP because the mentioned method evaluates not only the state transitions and message exchange, but also the message parameters. After that, this chapter describes the process of analyzing the compatibility and how to derive the compatibility matrix. Then, this chapter describes the basics of ISO 15118 and OCPP protocols: their OSI model, message construction, use cases, and implementation. Both are IP-based protocols, but OCPP relies entirely on Ethernet at the data link and physical layer. ISO 15118, however, uses the PLC protocol for communication. In terms of messaging, OCPP encodes its message in JSON format, but ISO 15118 uses EXI format. ISO 15118 has a fixed message sequence and operates in a client/server fashion, where the EV must be the client and the EVSE must be the server. In contrast, OCPP is a stateless protocol. Although OCPP defines client and server roles, both can send OCPP requests and responses simultaneously. SEVENSTAX provides the ISO 15118 and OCPP protocol stacks in C programming language. Both stacks are implemented in the event-driven pattern, where the OCPP and V2G applications must respond to messages from their associated services. Finally, this chapter also covers the message sequence of a charging session involving the ISO 15118 client, the ISO 15118 server, the OCPP client, and the OCPP server. Each charging session must go through the following phases: Communication setup, Identification - Authorization - Authentication, Target Setting - Charging Scheduling, Charging Loop, Charging Profile Renegotiation (optional), and End of Charging Session. For each phase during a charging session, the messages exchanged and their parameters are explained to highlight the interactions between the above components. The interactions between the ISO 15118 server and the OCPP client will be the input for the compatibility analysis.

3 Protocols Compatibility Analysis

This chapter describes how the compatibility between ISO 15118 and OCPP is analyzed based on the method proposed by Ouederni et al., 2010. First, the behavior of both protocols is modeled using the Symbolic Transition System. Then, the compatibility degrees are calculated based on the formulas in sections. Finally, the compatibility matrix is discussed and the conclusion is drawn based on the results.

3.1 The Analysis of ISO 15118 and OCPP

Modeling the interaction between the protocols is critical because it is used as an input to the analysis and can significantly affect the outcome. Ouederni et al., 2010 and the literature in Section 2.1 did not specifically mention any techniques for modeling the system. However, the following points can be derived from the papers to create such a model:

- Protocols evolve into new states when messages are exchanged between them. In order words, a position between two message exchanges is a state.
- The model must have an initial state and one or more final states.
- Message exchange is not limited to sending data in physical channels; messages can also be sent in logical channels, such as shared memory or message queues.

With the above information, the interactions between ISO 15118 and OCPP are modeled and illustrated in Figure 18 using the Symbolic Transition System (STS) and based on Chapter 2.5.

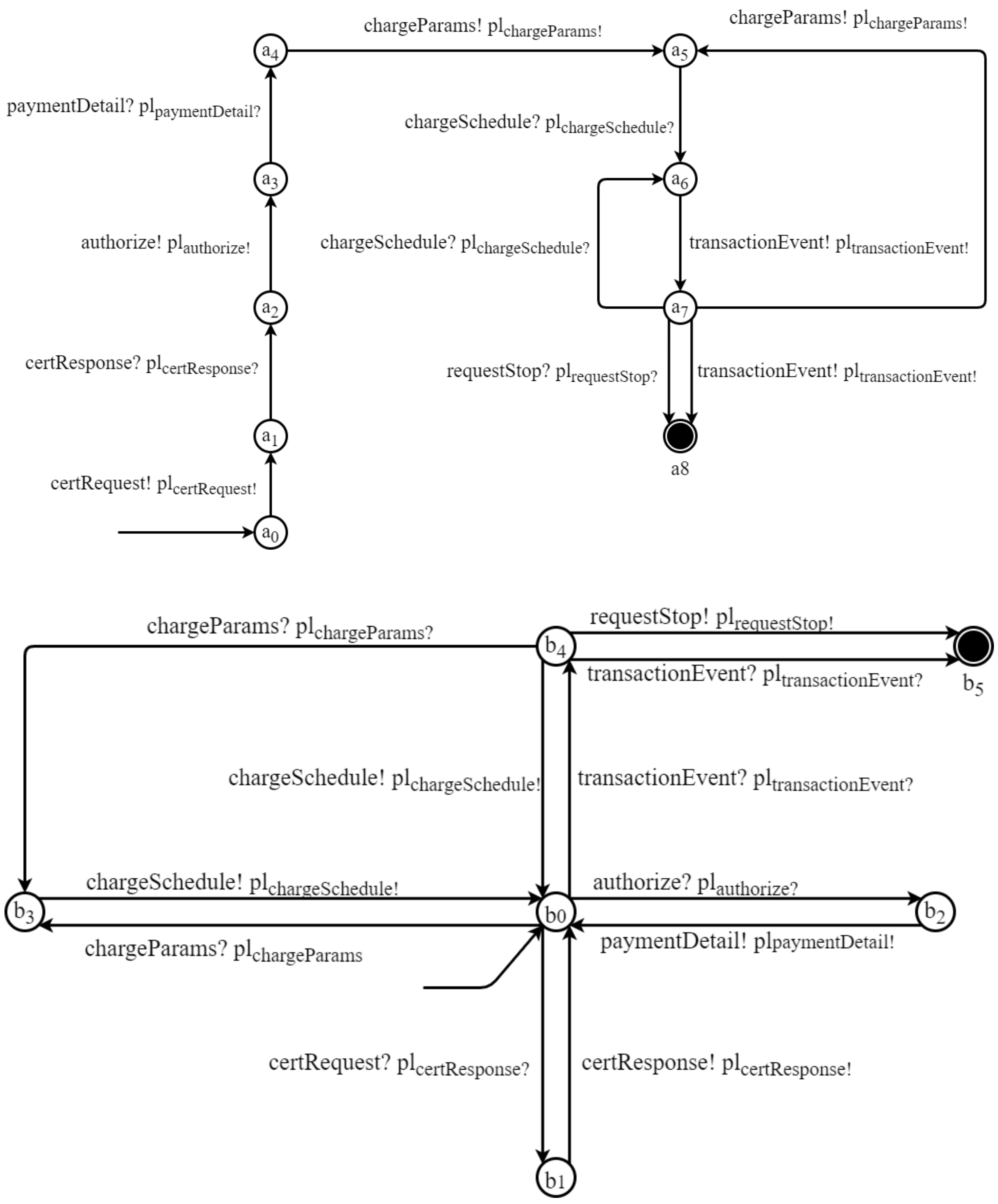

chargeParams! pl_chargeParams!
chargeParams! pl_chargeParams!
a4
a5
paymentDetail? pl_paymentDetail?
chargeSchedule? pl_chargeSchedule?
a3
a6
authorize! pl_authorize!
chargeSchedule? pl_chargeSchedule?
transactionEvent! pl_transactionEvent!
a2
a7
certResponse? pl_certResponse?
requestStop? pl_requestStop?
transactionEvent! pl_transactionEvent!
a1
a8
certRequest! pl_certRequest!
a0
chargeParams? pl_chargeParams?
requestStop! pl_requestStop!
b4
transactionEvent? pl_transactionEvent?
b5
chargeSchedule! pl_chargeSchedule!
transactionEvent? pl_transactionEvent?
chargeSchedule! pl_chargeSchedule!
authorize? pl_authorize?
b3
b0
b2
chargeParams? pl_chargeParams
paymentDetail! pl_paymentDetail!
certRequest? pl_certResponse?
certResponse! pl_certResponse!
b1

Figure 18: STS between ISO 15118 and OCPP

Note: Even though the ISO 15118 and OCPP components are deployed in the same hardware and software instance (Figure 11), and their interactions are function calls, not transceiving messages through a physical or logical bus, the analysis still uses the terminology ”sending/receiving” messages to denote the interaction between them.

The name of the states and their description are listed in Table 5 and Table 6. To keep the graph more readable, the messages' parameters are denoted as $pl_{message\_name}$. The concrete parameters are mentioned during the calculation and in Appendix C.

Table 5: Explanation of ISO 15118's STS

State	State name	Description
a_0	Initial	The ISO 15118 server handles the supportedProtocolApplicationReq, SessionSetupReq, ServiceDiscoveryReq, ServiceDetailReq, and PaymenServiceSelectionReq from the client. ISO 15118 advances to the next state when it transmits the certRequest! to the OCPP client.
a_1	CertInstallWait	The ISO 15118 server is waiting for the response from the OCPP client. It can advance to the next state when it receives certRequest?.
a_2	Authorize	The ISO 15118 server prepares and sends the authorization data in authorize! to OCPP, and then it advances to the next state.
a_3	AuthorizeWait	The ISO 15118 server waits for the response from the OCPP client.
a_4	ChargeParams	In this state, the ISO 15118 server sends the charging parameter to the OCPP client and transition to the next state.
a_5	ChargeParamWait	The ISO 15118 server is waiting for the charging schedule from the OCPP client. Upon receiving the response, the ISO 15118 server moves to the next state.
a_6	PreCharging	In this state, the ISO 15118 server sends the charging schedule to the EV and sends the transactionEvent! to the OCPP client to notify the beginning of a charging session and move to the next state.
a_7	Charging	In this state, the EV is being charged. Whenever the EV sends new charging parameters to the EVSE (hosting the ISO 15118 server), the server sends the message chargeParams! to the OCPP client to notify the renegotiation of the charging parameters and advance to state a_5. Upon receiving the new charging profile from the OCPP client via the message chargeSchedule?, the ISO 15118 advances to state a6 to send the new charging schedule to the EV. When the EV is fully charged, the ISO 15118 server sends the transactionEvent! to the OCPP client to stop the charging session and move to the final state. On the other hand, the ISO 15118 server can receive the requestStop? from the OCPP client to terminate the charging process and move to the final state.
a_8	Final(End Session)	State final.

Table 6: Explanation of OCPP's STS

State	State name	Description
b_0	Initial (Idle)	Inial state of the OCPP client. It is idle and waits for incoming messages. Upon receiving the message certRequest?, authorize?, chargeParam?, and transactionEvent?, the OCPP client advance to the following states CertWait, Authorize-Wait, ChargeScheduleWait, and Charging, respectively.
b_1	CertWait	In the states, the OCPP client waits for the certificate response message from its server. Upon receiving the response, it transmits the certResponse! to the ISO 15118 server and returns to state b_0.
b_2	AuthorizeWait	In the states, the OCPP client waits for the authorization response message from its server. Upon receiving the response, it transmits the paymentDetail! to the ISO 15118 server and returns to state b_0.
b_3	ChargeScheduleWait	In the states, the OCPP client waits for the charging schedule message from its server. Upon receiving the response, it transmits the chargeSchedule! to the ISO 15118 server and returns to state b_0.
b_4	Charging	In the charging state, the EV is being charged by the charging station. If the CSMS issues a new charging schedule, the OCPP client sends the schedule to the ISO 15118 server via the message chargeSchedule! and advances to state b_0. When the OCPP client receives new charging parameters from the ISO 15118 server via the chargeParam? message, it evolves to state b3 and waits for the new charge schedules from the OCPP server. The OCPP client can issue a charging termination by sending requestStop! to the ISO 15118 client and evolving to the final state. On the contrary, it can advance to state b5 by receiving the transactionEvent? from the ISO 15118 server.
b_5	Final	Final state. End of charging session.

From Figure 18, the reachable states are derived and illustrated as follows:

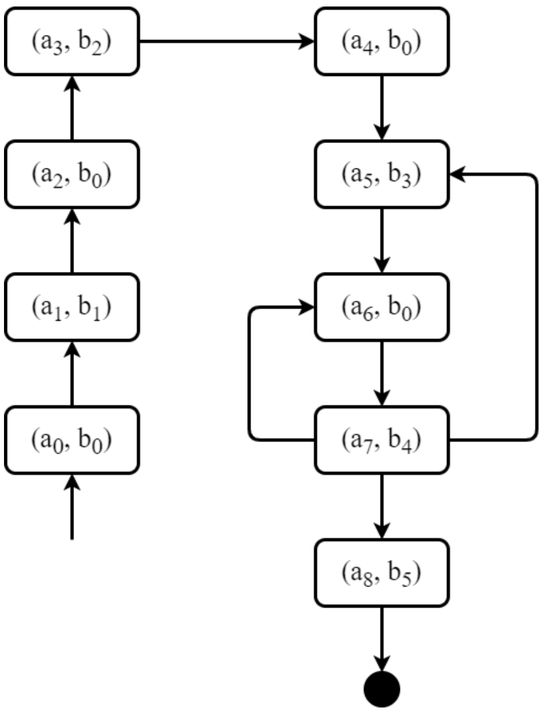


Figure 19: Reachable states

According to Ouederni et al., 2010, for the initial interaction, the state compatibility of all states has the value of 1. Therefore, the state compatibility for the first iteration is illustrated as follows.
For $k = 0$

Table 7: $COMP^0_{UR,\leftrightarrow}[a_i, b_j]$

	a_0	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8
b_0	1	1	1	1	1	1	1	1	1
b_1	1	1	1	1	1	1	1	1	1
b_2	1	1	1	1	1	1	1	1	1
b_3	1	1	1	1	1	1	1	1	1
b_4	1	1	1	1	1	1	1	1	1
b_5	1	1	1	1	1	1	1	1	1

The Calculation of $\mathbf{COMP^1_{UR,\leftrightarrow}[a_0, b_0]}$

$obs\_comp^1_{UR,\leftrightarrow}(a_0, b_0) = \frac{sum^1_{UR,\leftrightarrow}((a_0,b_0),E(a_0,T_0),R(b_0,T_0))+sum^1_{UR,\leftrightarrow}((b_0,a_0),E(b_0,T_0),R(a_0,T_0))}{\|E(a_0,T_0)\|+\|E(b_0,T_0)\|}$
where:

$$\begin{aligned}
&E(a_0, T_0) = \{(a_0, certRequest!pl_{certRequest!}, a_1)\}\\
&R(a_0, T_0) = \varnothing\\
&E(b_0, T_0) = \varnothing\\
&tau(a_0, T_0) = \varnothing\\
&tau(b_0, T_0) = \varnothing\\
&Fw(a_0, T_0) = E(a_0, T_0)\\
&Fw(b_0, T_0) = R(b_0, T_0)
\end{aligned}$$

$$\begin{aligned}
R(b_0, T_0) = \{&(b_0, certReqest?pl_{certRequest?}, b_1),\\
&(b_0, authorize?pl_{authorize?}, b_2),\\
&(b_0, chargeParams?pl_{chargeParams?}, b_3),\\
&(b_0, transactionEvent?pl_{transactionEvent?}, b_4)\}
\end{aligned}$$

The function $sum^1_{UR,\leftrightarrow}((a_0, b_0), E(a_0, T_0), R(b_0, T_0))$ calculates the sum of the best compatibility between $E(a_0, T_0)$ and $R(b_0, T_0)$. The calculation is reduced down to the transitions $(a_0, certRequest!pl_{certRequest!}, a_1)$ and $(b_0, certRequest?pl_{certRequest?}, b_1)$ because other transitions are unmatched.

$$\begin{aligned}
&\Rightarrow sum^1_{UR,\leftrightarrow}((a_0, b_0), E(a_0, T_0), R(b_0, T_0))\\
&\quad = lab\_comp(certRequest!, certRequest?) * COMP^0_{UR,\leftrightarrow}[a_1, b_1]
\end{aligned}$$

with:

$$COMP^0_{UR,\leftrightarrow}[a_1, b_1] = 1$$

$$lab\_comp(certRequest!, certRequest?) = 1 - \frac{\|unsharedTypes(pl_{certRequest!}, pl_{certRequest?})\|)}{6(\|pl_{certRequest!}\| + \|pl_{certRequest?}\|)}$$

From Table 13, the un-shared data types between the messages are:

$$unshared = \{base64encoding, string\}$$

$$\begin{aligned}
&\Rightarrow \|unsharedTypes(pl_{certRequest!}, pl_{certRequest?})\| = 2\\
&\Rightarrow lab\_comp(certRequest!, certRequest?) = 1 - \frac{2}{6*(1+1)} = 0.833 \qquad (9)\\
&\Rightarrow sum^1_{UR,\leftrightarrow}((a_0, b_0), E(a_0, T_0), R(b_0, T_0)) = 0.833
\end{aligned}$$

The function $sum^1_{UR,\leftrightarrow}((b_0, a_0), E(b_0, T_0), R(a_0, T_0))$ is based upon the compatibility between the $E(b_0, T_0)$ and $R(a_0, T_0)$. Because of $E(b_0, T_0) = \varnothing$

$$\begin{aligned}
&\Rightarrow sum^1_{UR,\leftrightarrow}((b_0, a_0), E(b_0, T_0), R(a_0, T_0)) = 0\\
&\Rightarrow obs\_comp^1_{UR,\leftrightarrow}(a_0, b_0) = \frac{0.833+0}{1} = 0.833
\end{aligned}$$

Because $tau(a_0, T_0) = \varnothing$, it can be concluded that the forward are equal to the observable compatibility $obs\_comp(a_0, T_0)$.

Proof:

$$fw\_propag^1_{UR,\leftrightarrow}(a_0, b_0) = \frac{d\_fw\_propag^1_{UR,\leftrightarrow}(a_0, b_0) + d\_fw\_propag^1_{UR,\leftrightarrow}(b_0, a_0)}{2}$$

with:

$$\begin{aligned}
d\_fw\_propag^1_{UR,\leftrightarrow}(a_0, b_0) &= \frac{\sum_{(a_0,\tau,a'_0)\in T_0} fw\_propag^1_{UR,\leftrightarrow}(a'_0, b_0) + obs\_comp^1_{UR,\leftrightarrow}(a_0, b_0)}{\|tau(a_0, T0)\| + 1}\\
&= obs\_comp^1_{UR,\leftrightarrow}(a_0, b_0)
\end{aligned}$$

$(tau(a_0, T_0) = \varnothing \Rightarrow \sum_{(a_0,\tau,a'_0)\in T_0} fw\_propag^1_{UR,\leftrightarrow}(a'_0, b_0) = 0$ and $\|tau(a_0, T_0)\| = 0)$

$$\begin{aligned}
d\_fw\_propag^1_{UR,\leftrightarrow}(b_0, a_0) &= \frac{\sum_{(b_0,\tau,b'_0)\in T_0} fw\_propag^1_{UR,\leftrightarrow}(b'_0, a_0) + obs\_comp^1_{UR,\leftrightarrow}(b_0, a_0)}{\|tau(b_0, T_0)\| + 1}\\
&= obs\_comp^1_{UR,\leftrightarrow}(b_0, a_0)
\end{aligned}$$

$(tau(b_0, T_0) = \varnothing \Rightarrow \sum_{(b_0,\tau,b'_0)\in T_0} fw\_propag^1_{CN,\leftrightarrow}(a'_0, b_0) = 0$ and $\|tau(b_0, T_0)\| = 0)$
Therefore,

$$\begin{aligned} fw\_propag^{k}_{UR,\leftrightarrow}(a_0,b_0) &= \frac{d\_fw\_propag^{k}_{UR,\leftrightarrow}(a_0,b_0) + d\_fw\_propag^{k}_{UR,\leftrightarrow}(b_0,a_0)}{2} \\ &= obs\_comp^{1}_{UR,\leftrightarrow}(a_0,b_0) \\ &= 0.833 \end{aligned}$$

Due to no internal backward propagation to (a_0, b_0), the backward propagation of (a_0, b_0) is equal to their observable compatibility.

Proof:

$$bw\_propag^{1}_{UR,\leftrightarrow}(a_0,b_0) = \frac{d\_bw\_propag^{1}_{UR,\leftrightarrow}(a_0,b_0) + d\_bw\_propag^{1}_{UR,\leftrightarrow}(b_0,a_0)}{2}$$

with:

$$\begin{aligned} d\_bw\_propag^{1}_{UR,\leftrightarrow}(a_0,b_0) &= \frac{\sum_{(a'_0,\tau,a_0)\in T_0} bw\_propag^{1}_{UR,\leftrightarrow}(a'_0,b_0) + obs\_comp^{1}_{UR,\leftrightarrow}(a_0,b_0)}{\|tau(a_0,T_0)\|+1} \\ &= obs\_comp^{1}_{CN,\leftrightarrow}(a_0,b_0) \end{aligned}$$

($\sum_{(a'_0,\tau,a_0)\in T_0} bw\_propag^{1}_{UR,\leftrightarrow}(a'_0,b_1) = 0$ because state a_0 is the initial state, which has no backward propagation)

$$\begin{aligned} d\_bw\_propag^{1}_{UR,\leftrightarrow}(b_0,a_0) &= \frac{\sum_{(b'_0,\tau,b_0)\in T_1} bw\_propag^{1}_{UR,\leftrightarrow}(b'_0,a_0) + obs\_comp^{1}_{UR,\leftrightarrow}(b_0,a_0)}{\|tau(b_0,T_0)\|+1} \\ &= obs\_comp^{1}_{UR,\leftrightarrow}(b_0,a_0) \end{aligned}$$

($\sum_{(b'_0,\tau,b_0)\in T_0} bw\_propag^{1}_{UR,\leftrightarrow}(b'_0,a_0) = 0$ because state b_0 has no internal transition from backward neighbor states.

$$\Rightarrow bw\_propag^{1}_{UR,\leftrightarrow}(a_0,b_0) = obs\_comp^{1}_{UR,\leftrightarrow}(b_0,a_0) = 0.833$$

According to Equation 7, the state compatibility of the state (a_0, b_0) is:

$$\begin{aligned} & state\_comp^{1}_{UR,\leftrightarrow}(a_0,b_0) \\ &= \frac{w_1 * fw\_propag^{1}_{UR,\leftrightarrow}(a_0,b_0) + w_2 * bw\_propag^{1}_{UR,\leftrightarrow}(a_0,b_0) + w_3 * nat(a_0,b_0)}{w_1+w_2+w_3} \end{aligned}$$

Where

- $w_1 = 5$ because a_0 has one outgoing matching transition and b_0 has four outgoing matching transitions.
- $w_2 = 4$ because b_0 has four incoming matching transitions .
- $w_3 = 1$ because there is no τ transition. $nat(a_0, b_0) = 1$ since both state are initial state.

$$\Rightarrow state\_comp^1_{UR,\leftrightarrow}(a_0, b_0) = \frac{5 * 0.833 + 4 * 0.833 + 1 * 1}{5 + 4 + 1} = 0.85$$

and

$$COMP^1_{UR,\leftrightarrow}[a_0, b_0] = \frac{COMP^0_{UR,\leftrightarrow}[a_0, b_0] + state\_comp^1_{UR,\leftrightarrow}(a_0, b_0)}{2} = \frac{1 + 0.85}{2} = 0.925$$

The Calculation of $\mathbf{COMP^1_{UR,\leftrightarrow}[a_0, b_1]}$ From Figure 18, the state (a_0, b_1) is deadlock. Therefore, theirs compatibility degree yields value 0, i.e., $COMP^1_{UR,\leftrightarrow}[a_0, b_1] = 0$

Proof:
$$obs\_comp^1_{UR,\leftrightarrow}(a_0, b_1) = \frac{sum^1_{UR,\leftrightarrow}((a_0,b_1),E(a_0,T_0),R(b_1,T_1))+sum^1_{UR,\leftrightarrow}((b_1,a_0),E(b_1,T_1),R(a_0,T_0))}{\|E(a_0,T_0)\|+\|E(b_1,T_1)\|}$$

where:

$$E(a_0, T_0) = \{(a_0, certRequest!pl_{certRequest!}, a_1)\}$$
$$R(a_0, T_0) = \varnothing$$
$$tau(a_0, T_0) = \varnothing$$
$$Fw(a_0, T_0) = E(a_0, T_0)$$

and

$$E(b_1, T_1) = \{(b_1, certResponse!pl_{certResponse!}, b_0)\}$$
$$R(b_1, T_1) = \varnothing$$
$$tau(b_1, T_1) = \varnothing$$
$$Fw(b_1, T_1) = E(b_1, T_1)$$

The function $sum^1_{UR,\leftrightarrow}((a_0, b_1), E(a_0, T_0), R(b_1, T_1))$ calculates the sum of the best compatibility between the emissions of state a_0 and the receptions of state b_1. Because there is no reception at state b_1 ($R(b_1, T_1) = \varnothing$), $sum^1_{UR,\leftrightarrow}((a_0, b_1), E(a_0, T_0), R(b_1, T_1)) = 0$. With the same argument, it can be concluded that, $sum^1_{UR,\leftrightarrow}((b_1, a_0), E(b_1, T_1), R(a_0, T_0)) = 0$.

$$\Rightarrow obs\_comp^1_{UR,\leftrightarrow}((a_0, b_1)) = 0$$

Because there is no internal forward propagation in a_0, b_1

$$fw\_propag^k_{CN,\leftrightarrow}(a_0, b_1) = obs\_comp^1_{UR,\leftrightarrow}(a_0, b_1) = 0$$

Due to no internal backward propagation to (a_0, b_1), the backward propagation of (a_0, b_1) is equal to their observable compatibility.

$$\Rightarrow bw\_propag^1_{UR,\leftrightarrow}(a_0, b_1) = obs\_comp^1_{UR,\leftrightarrow}(a_0, b_1) = 0$$

The state compatibility of state (a_0, b_1) is calculated as follows:

$$\begin{aligned}
&state\_comp^1_{UR,\leftrightarrow}(a_0, b_1) \\
&= \frac{w_1 * fw\_propag^1_{UR,\leftrightarrow}(a_0, b_1) + w_2 * bw\_propag^1_{UR,\leftrightarrow}(a_0, b_1) + w_3 * nat(a_0, b_1)}{w_1 + w_2 + w_3} \\
&= \frac{w_3 * nat(a_0, b_1)}{w_1 + w_2 + w_3} \\
&= 0
\end{aligned}$$

($nat(a_0, b_1) = 0$ because $a_0 \in I_a$ & $b_1 \notin (I_b \cup F_b)$)

And the compatibility degree is calculated as follows:

$$COMP^1_{UR,\leftrightarrow}[a_0, b_1] = \frac{COMP^0_{UR,\leftrightarrow}[a_0, b_1] + state\_comp^1_{UR,\leftrightarrow}(a_0, b_1)}{2} = \frac{1 + 0}{2} = 0.5$$

Apply the same procedure to states (a_0, b_2), (a_0, b_3), (a_0, b_4), (a_0, b_5),

$$
\begin{aligned}
state\_comp^1_{UR,\leftrightarrow}(a_0, b_1) &= \frac{w_3 * nat(a_0, b_2)}{w_1 + w_2 + w_3} = 0 \\
state\_comp^1_{UR,\leftrightarrow}(a_0, b_2) &= \frac{w_3 * nat(a_0, b_3)}{w_1 + w_2 + w_3} = 0 \\
state\_comp^1_{UR,\leftrightarrow}(a_0, b_3) &= \frac{w_3 * nat(a_0, b_4)}{w_1 + w_2 + w_3} = 0 \\
state\_comp^1_{UR,\leftrightarrow}(a_0, b_4) &= \frac{w_3 * nat(a_0, b_5)}{w_1 + w_2 + w_3} = 0
\end{aligned}
$$

since $a_0 \in I_a$ & $(b_1, b_2, b_3, b_4, b_5) \notin (I_b \cup F_b))$
and

$$
\begin{aligned}
COMP^1_{UR,\leftrightarrow}[a_0, b_2] &= COMP^1_{UR,\leftrightarrow}[a_0, b_3] \\
&= COMP^1_{UR,\leftrightarrow}[a_0, b_4] \\
&= COMP^1_{UR,\leftrightarrow}[a_0, b_5] \\
&= 0.5
\end{aligned}
$$

The rest of the compatibility degrees are calculated in Appendix D. For $k = 1$, the compatibility degree table is presented as follows:

Table 8: $COMP^1_{UR,\leftrightarrow}[a_i, b_j]$

	a_0	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8
b_0	0.925	0.5	0.936	0.5	0.931	0.5	0.882	0.914	0.5
b_1	0.5	0.933	0.6	0.6	0.6	0.584	0.584	0.563	0.5
b_2	0.5	0.6	0.6	0.933	0.6	0.584	0.584	0.563	0.5
b_3	0.5	0.584	0.584	0.584	0.584	0.978	0.571	0.693	0.5
b_4	0.5	0.563	0.563	0.563	0.702	0.766	0.679	0.97	0.5
b_5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	1

Based on the calculation performed in Appendix D, a scripting application is developed using the Python programming language to assist in the calculation of the compatibility matrix (Nguyen, 2023). The application is a command line interface that allows the user to input the paths to the graph and the number of iterations, and then outputs the compatibility matrix as results or error messages (Listing 5 and 6).

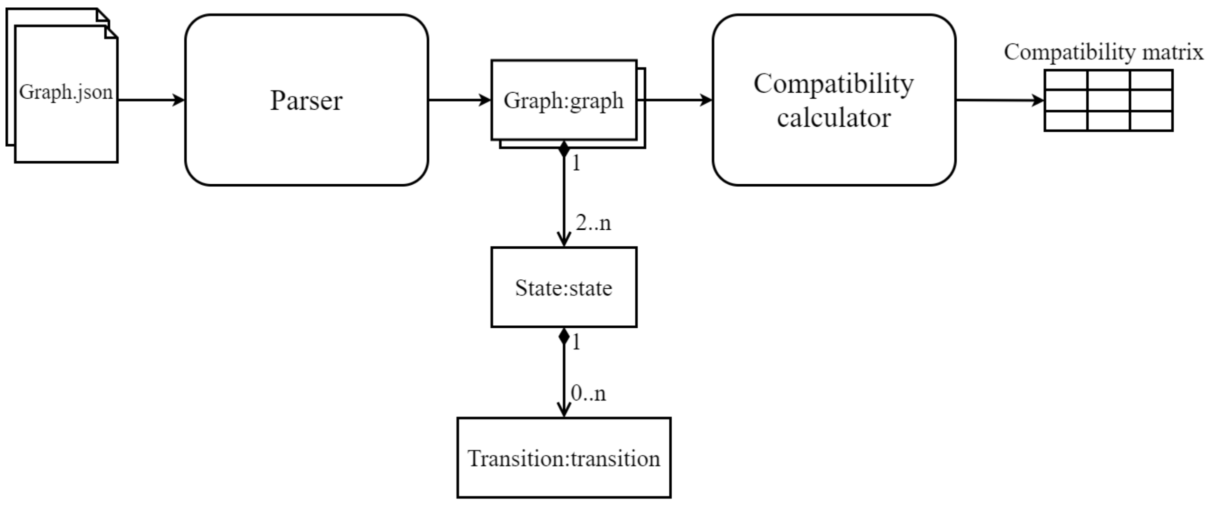


Figure 20: Compatibility Calculator application

Figure 20 illustrates the design of the tool, including its input and output data. The tool consists of two modules, the Parser and the Compatibility Calculator. The Parser is responsible for validating and parsing the input data. The input data are the JSON files describing the protocol's model, following the format described in Listing 4. After the parser successfully validates the input data, it parses data into the Graph object for processing (the validation criterion is described by Nguyen, 2023). The Graph class contains information about the associated protocols, including their states and transitions. The design of the Graph class is specified in Figure 35. The Compatibility Calculator module inputs the Graph objects to compute the compatibility matrices. Due to time constraints, the Compatibility Calculator module is implemented using the Unspecified Reception notion to calculate the compatibility degree. The results of the calculation are displayed on the terminal and saved to a text file.

For the analysis, ISO 15118's STS and OCPP's STS are converted into JSON data, which is calculated by the Compatibility calculator application for 11 iterations. The result of the calculation is presented and discussed in the following section.

3.2 Results And Discussion

3.2.1 Results

Table 9: $COMP^{11}_{UR,\leftrightarrow}[a_i, b_j]$

	a_0	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8
b_0	0.581	0	0.573	0.0	0.594	0	0.691	0.306	0
b_1	0	0.611	0.2	0.2	0.2	0.167	0.167	0.126	0
b_2	0	0.2	0.2	0.621	0.2	0.167	0.167	0.126	0
b_3	0	0.167	0.167	0.167	0.167	0.663	0.143	0.291	0
b_4	0	0.126	0.126	0.126	0.316	0.381	0.113	0.794	0
b_5	0	0	0	0	0	0	0	0	1

Table 9 illustrates the compatibility degree of ISO 15118 and OCPP calculated with the prototype tool after 11 iterations. In column a_0, (a_0, b_0) has the highest compatibility degree. It is consistent with the model where state a_0 sends certRequest! and proceeds to state a_1; and b_0 receives CertRequest? and proceeds to state b_1. However, the compatibility degree is not perfect because of the unshared data types in the messages. As of ISO 15118, certRequest! uses *base64Binary* as the data type for its CertificateInstallationRequest parameter. On the other hand, OCPP expects *string* as the data type for its equivalent parameter, ExiRequest. This data type incompatibility affects the label compatibility calculation (Equation 9), which ultimately reduces the overall compatibility of (a_0, b_0). The other states, such as (a_0, b_1), (a_0, b_2), (a_0, b_3), (a_0, b_4), and (a_0, b_5) are in a deadlock because there is no valid message exchange to allow the protocols to proceed to the next states.

In column a_1, (a_1, b_1) has the highest compatibility degree because they share the compatibility message certResponse. They also do not have a perfect compatibility degree because of the data type incompatibility (*base64Binary* and *string*). (a_1, b_0) and (a_1, b_5) have a compatibility degree of zero because they are deadlocked and have no compatibility messages. Although (a_1, b_2), (a_1, b_3), and (a_1, b_4) are also deadlocked, their compatibility degrees are not zero. The reason is that according to formula 7, the state compatibility degree is also depend on the nature of the state, and the mentioned states have the same state nature (they are neither initial state nor final state).

In column a_2, (a_2, b_0) has the highest compatibility degree, but the value is not perfect. The reason is the same as in the previous cases. The message is compatible, but the they do not have the same data types (Table 15). In this column, (a_2, b_5) has the lowest degree of compatibility because it does not have the same state type and compatibility messages.

(a_2, b_1), (a_2, b_2), (a_2, b_3), and (a_2, b_4) are also deadlocked, but they have the same state nature, so the compatibility degrees are not zero.

Column a_3 has the same behavior as column a_1, (a_3, b_0) and (a_3, b_5) have zero compatibility degrees due to incompatibility message and different state natures. The other states, such as (a_3, b_1), (a_3, b_3), and (a_3, b_4), are also deadlocked, but they have the same state nature; thus, their compatibility degrees are low but not zero. (a_3, b_2) has the highest degree of compatibility because they have the compatibility message paymentDetail. This message exchange again has unshared data types (Table 16), causing the compatibility degree to be not perfect.

In column a_4, (a_4, b_0) has the highest compatibility degree because it shares the compatible message-chargeParams. Again, the degree of compatibility is not perfect because of unshared data types (Table 17). (a_4, b_5) has zero compatibility because of deadlock and state-nature differences. (a_4, b_1), (a_4, b_2), and (a_4, b_3) have deadlock but share the same state nature; thus, the compatibility degrees are low but not zero. Interestingly, (a_4, b_4) does not have a high compatibility degree (0.316), but still has the second highest value. The reason is that it has a legal transition, i.e. $(a_4, chargeParam!, a_5)$ and $(b_4, chargeParam?, b_3)$. According to Figure 19, the transition to (a_4, b_4) does not happen because it is not in the list of reachable states. However, if this transition were to occur, it would place the charging station in an undesirable state. In state a_4, the EV is not yet charged and the ISO 15118 server sends the charging parameters to the OCPP client. In state b_4, the OCPP client is already in the charging state where the EV is already charged, which is inconsistent with the ISO 15118 server. In practice, this behavior must be prevented to avoid unsynchronization between ISO 15118 and OCPP.

Column a_5 has the same behavior as columns a_1 and a_3, with (a_5, b_0) and (a_5, b_5) having zero compatibility due to deadlock and state nature differences. (a_5, b_3) has the highest degree of compatibility, but the value is not perfect due to data type differences (Table 18). (a_5, b_4) has the same behavior as (a_4, b_4). It does not have a high compatibility degree (0.381), but it is still the second highest. The reason is that it has a legal transition, i.e. $(a_5, chargeSchedule?, a_6)$ and $(b_4, chargeSchedule!, b_0)$. (a_5, b_4) does not happen because it is not in the list of reachable states and it must be prevented for the same reason as with (a_4, b_4).

Column a_6 has the same behavior as column a_2. (a_6, b_0) has the highest degree of compatibility, but this value is not perfect due to data type differences (Table 20). (a_6, b_5) has zero compatibility due to deadlock and state nature differences. (a_6, b_1), (a_6, b_2), (a_6, b_3), and (a_6, b_4) have deadlocks but still share the same state nature. Therefore, their degrees of compatibility are not zero.

In column a_7, (a_7, b_4) has the highest degree of compatibility, but this value is not perfect for the same reason as the previous cases. (a_7, b_1), (a_7, b_2), and (a_7, b_5) are deadlocked. Nevertheless, (a_7, b_1) and (a_7, b_2) have the same state nature, so their compatibility degree is not zero. (a_7, b_0) and (a_7, b_3) have higher compatibility degrees than the deadlock states because of legal transitions that are transactionEvent and chargeSchedule, respectively. However, these states do not occur because they are not on the list of reachable states and must be prevented.

Finally, (a_8, b_5) in column a8 has the highest compatibility value because they are both states. The rest have zero compatibility due to deadlock and state nature differences.

3.2.2 Discussion

From the results section, the compatibility matrix of ISO 15118 and OCPP provides four points:

- States that are incompatible and have different state natures have zero compatibility degrees.
- States that are incompatible but have the same state nature have relatively low compatibility degrees, ranging from 0.1 to 0.2 (Table 9).
- There are unreachable states with high compatibility degrees, such as (a_4, b_4), (a_5, b_4), (a_7, b_0), or (a_7, b_3), compared to the incompatible states. They have such a compatibility degree because of compatible transitions. In practice, these unreachable states are undesirable and must be avoided.
- The highest degrees of compatibility in each column belong to the compatible states, which are also the reachable states (Figure 19). It is therefore concluded that ISO 15118 and OCPP are compatible.

Despite being compatible in states and transitions, the ISO 15118 and OCPP compatibility is not perfect because of the low label compatibility caused by the unshared data type. For example, in Table 17, when exchanging the charging parameters, ISO 15118 uses the *PhysicalValueType* data type, which is a structure of three parameters: *Value*, *Multiplier*, and *Unit*. The actual value from this data type is calculated as follows:

$$final\_value = Value * 10^{Multiplier}$$

On contrary, OCPP uses the data type *integer* for its charging parameters. Another example is ISO 15118 uses *byte* data type for the parameter FullSOC, but OCPP uses *integer* data type. Of course, not all ISO 15118 enumeration data types can be mapped one-to-one to OCPP enumeration data types. For example, in Table 17, the Energy-TransferMode parameter from ISO 15118 has one of the following enumeration values: *AC_single_phase_core*, *AC_three_phase_core*, *DC_core*, *DC_extended*, *DC_combo_core*, or *DC_unique* (ISO, 2014). Meanwhile, the requestedEnergyTransfer parameter from OCPP has one of the enumeration values: *DC*, *AC_single_phase*, *AC_two_phase*, or *AC_three_phase* (Open Charge Alliance, 2020c). The enumeration mismatch creates ambiguity during implementation because it is up to the developers to decide which ISO 15118 enumeration value matches which OCPP enumeration. For instance, the developer can convert the values *DC_extended*, *DC_combo_core*, and *DC_unique* from ISO 15118 to *DC* in OCPP. From a semantic point of view, this conversion is appropriate. However, from the perspective of ISO 15118 standards, this conversion may lose detailed information about the DC charge type, and eventually, these inconsistencies contribute to the low interoperability of the protocols.

The explanation for this behavior is that OCPP aims to be a generalized back-end protocol supporting not only ISO 15118 but also other charging-related protocols, for example, CHAdemo (Open Charge Alliance and CHAdeMo, 2020) or OpenARD (Hoekstra et al., n.d.). Therefore, the OCPP's parameters use common data types and do not include specific or domain knowledge data types. Furthermore, those data types must be large enough to hold the potential data types from other protocols. Type compatibility between parameters is important to the software implementation because it reduces the effort and complexity of the implementation. For example, if two standalone parameters have the same data type, developers can perform an assignment operation to pass a value from ISO 15118 to the OCPP message:

```
iso15118_param_a = ocpp_param_a;
```

If the parameters are packed into the structures where they have the same others, and their data types are compatible (Listing 2), the developers can use the memcpy function to pass the data:

```
memcpy(&iso15118_data, &ocpp_data, sizeof(
    iso15118_data_struture_b))
```

On the contrary, if the parameters of specific OCPP and ISO 15118 data structures are not in the same order and their data types are not compatible, the developer must manually match and do data transformation on each parameter (Listing 3). Table 10 shows a list of mismatched data type between parameters of ISO 15118 and OCPP and the method to convert them.

Table 10: Mismatched data type between the ISO 15118 and OCPP parameters

ISO 15118	OCPP	Conversion method
Base64	string	Dedicated base64 encoding and decoding functions
ResponseCodeEnum	CertificateStatusEnum	Comparing the semantic meaning of the enumeration value
EnergyTransferModeEnum	RequestedEnergyTransferEnum	Comparing the semantic meaning of the enumeration value
PhysicalValueType	integer	Dedicated conversion function
Byte	integer	Type casting
Unsigned integer	integer	Type casting
Iso15118CostkindEnum	OcppCostkindEnum	Comparing the semantic meaning of the enumeration value
unsignedLong	number	Type casting
PhysicalValueType	number	Dedicated conversion function

In summary, ISO 15118 and OCPP protocols are compatible. The interactions between the protocols start from the initial state and reach the final state, forming a list of reachable states (Figure 19). Each reachable state results in a high degree of compatibility, as shown in Table 9. However, the compatibility is not perfect because the messages exchanged do not have similar data types. In addition, such states (a_4, b_4), (a_5, b_4), (a_7, b_0), and (a_7, b_3) have high compatibility degrees but are unreachable. These states must be prevented to avoid unsynchronization between the protocols. Therefore, to implement a Bridging application between the protocols, developers must implement the helper functions that convert the between OCPP and ISO 15118 data types (the conversion methods are listed in Table 10). Since OCPP is a stateless protocol, meaning that any message can be sent and received in any state, ISO 15118 may receive OCPP messages that are not expected for that state. Therefore, developers must implement a mechanism to monitor which OCPP messages are allowed to be sent and expected to be received in a particular ISO 15118 charging state.

3.3 Chapter's Summary

This chapter describes the steps to calculate the protocol compatibility between ISO 15118 and OCPP. Based on the result of the first calculation iteration, a prototype tool is implemented to calculate the compatibility between protocols. The compatibility matrix between ISO 15118 and OCPP (Table 9) calculated by the prototype tool concludes that the protocols are not perfectly compatible due to the different data types of the exchanged messages. Therefore, when implementing a Bridging application between the two protocols, a data type conversion between the protocols must be implemented. In addition, due to the stateless nature of OCPP, a monitoring mechanism must be implemented to allow ISO 15118 to send and receive OCPP messages accordingly.

4 Development of a Protocols Bridging Application

This chapter discusses the concept, architecture, and construction of the protocol Bridging application to enable data transmission between EV, EVSE, and CSMS. Additionally, this application must tackle the findings in the previous chapter, namely:

- Data types differences between ISO 15118 and OCPP.
- Controlling which OCPP messages can be sent and received during the charging session to prevent deadlock or unexpected states.

Furthermore, the bridging application must handle the errors occurring during run-time, which are not foreseen by the theoretical analysis, such as timeout, connection interruptions, and invalid messages transmission or reception. Before the application's design is described, the operations of ISO 15118 and OCPP stack, which is the extension of Section 2.3.4 and 2.4.4, are explained to give the readers an idea of how each stack operates standalone and how to combine them. Then, based on that, the bridging application's design is described to illustrate how this design can solved the mentioned issues.

4.1 Assumptions and Requirements

The bridging application is designed with the following assumptions:

- Both ISO 15118 and OCPP stacks are deployed in the same hardware and software instance (Figure 11). Therefore, they communicate with each other via function calls and shared memory. The stacks can also be deployed in a multi-core system where one core handles the ISO 15118 stack, and the other handles the OCPP stack. Even though the compatibility analysis still holds, this setup is not in the scope of the thesis.
- The Bridging application depends on the SEVENSTAX's OCPP stack.
- The Bridging application implementation is designed to operate with SEVENSTAX's ISO 15118 stack. It is, however, open to work with ISO 15118 stacks from other vendors.

Besides the mentioned assumptions, the following high-level requirements must be considered during the design of the bridging application:

- The application must be event-driven; in other words, polling is forbidden. Rationale: The bridging application must not block other modules.

- The application must not use dynamic memory allocation, i.e., malloc(). Rationale: both protocol stacks and the Bridging application are deployed in embedded systems, where dynamic memory allocation is discouraged.
- The bridging application must depend only on SEVENSTAX's OCPP stack. Rationale: this requirement allows the bridging application and OCPP stack work with the ISO15118 stack from other vendors.
- When errors occur in the Bridging application or in the OCPP stack, they must be handled, and they must not interfere with the charging session: Rationale: This requirement ensures that if errors occur on the OCPP side, the OCPP stack can be recovered and the charging session is not interrupted.
- The Bridging application must be open for extension to the new ISO 15118 standards. Rationale: ISO 15118-20 is currently implemented, and the bridging application must support this new standard in the future.

4.2 Software Development Environment

The bridging application is developed using Visual Studio 2022 on a PC running the Windows 10 operating system (Figure 21). The Visual Studio project consists of an EVSE reference application that demonstrates how a typical charging session is implemented using the SEVENSTAX ISO 15118 software stack. The Bridging application is developed using the OCPP stacks and provides the interfaces to the EVSE application so that the EVSE application can communicate with the CSMS.

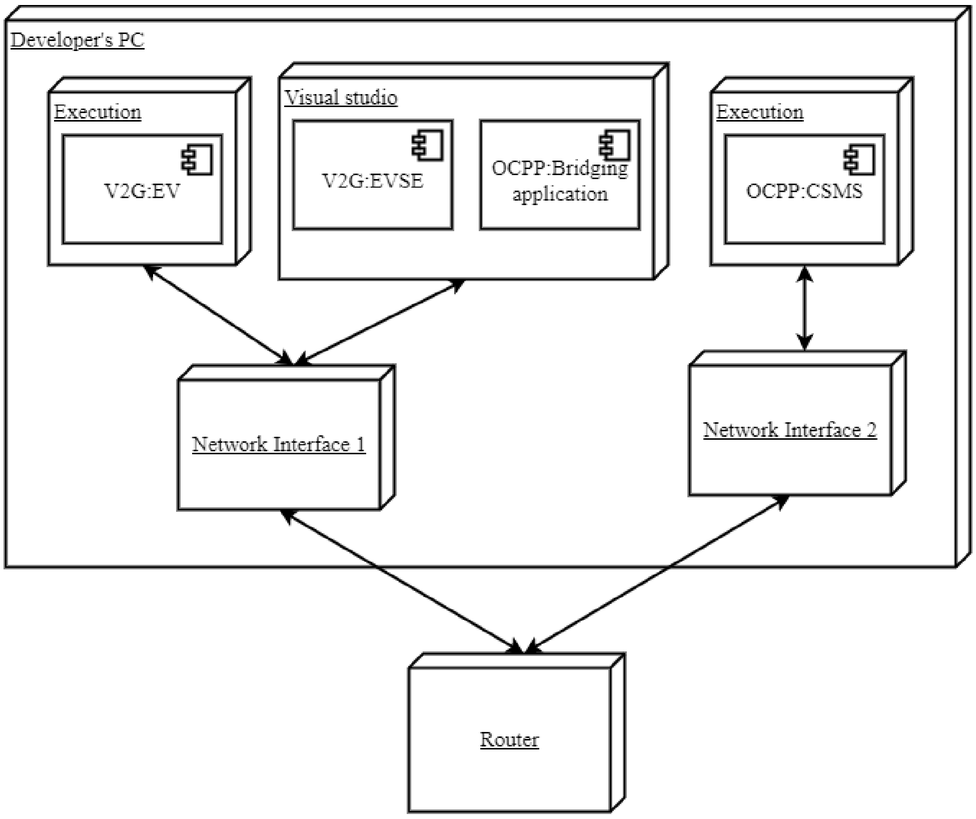


Figure 21: Software development environment

The communication between the EV and the EVSE is tested with an EV application compiled with the options using the External Identification Means (EIM) and the Contract Certificate for authentication. Similarly, a Python application using OCPP stack from The Mobility House, n.d. is implemented to simulate the operation of the CSMS and to test the interoperability of SEVENSTAX's OCPP stack with other vendors. Due to the requirements of the SEVENSTAX software stack, the EV and Bridging applications use a separated network interface from the CSMS application. In the physical layer, the EV and the EVSE communicate via the PLC protocol (Figure 5) but PLC is omitted during development to make it less dependent on hardware components and easier to test. Instead of PLC, the EV and EV applications use Ethernet to communicate with each other.

4.3 SEVENSTAX's ISO 15118 Stack Operation

As mentioned in Section 2.3.1, in ISO 15118 protocol, EV is always the one sending the request to the EVSE, and the EVSE processes the request before sending back a response to EV. Therefore, from the software implementation perspective, the ISO 15118 stack has two operations, processing requests and building the responses (Figure 22 and 23).

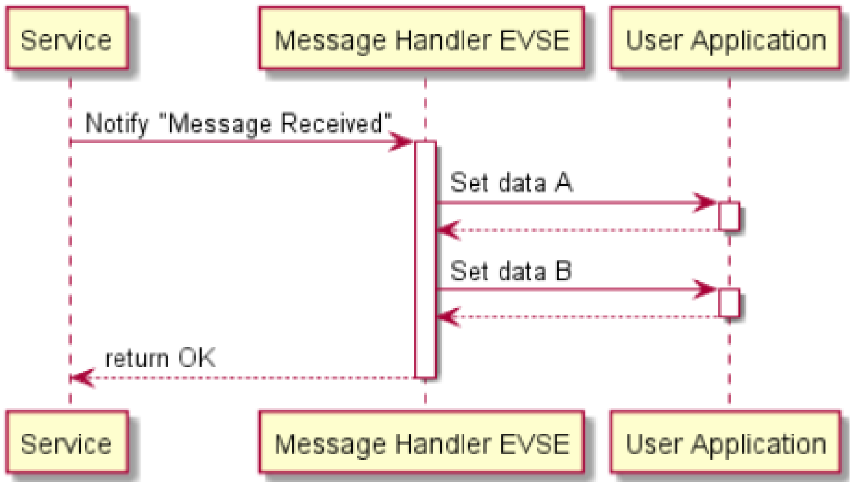


Figure 22: Handling ISO 15118 request. Extracted from SEVENSTAX GmbH, 2021

The message sequence in Figure 22 describes how the ISO 15118 application handles the ISO 15118 requests. Whenever a request comes from the EV, the Service component handles the message decoding from EXI format to C data structure (Section 2.3.4) and notifies the Message Handler EVSE about the new request. The Message Handler EVSE unpacks the C data structure and notifies the application about the parameters in the data structure.

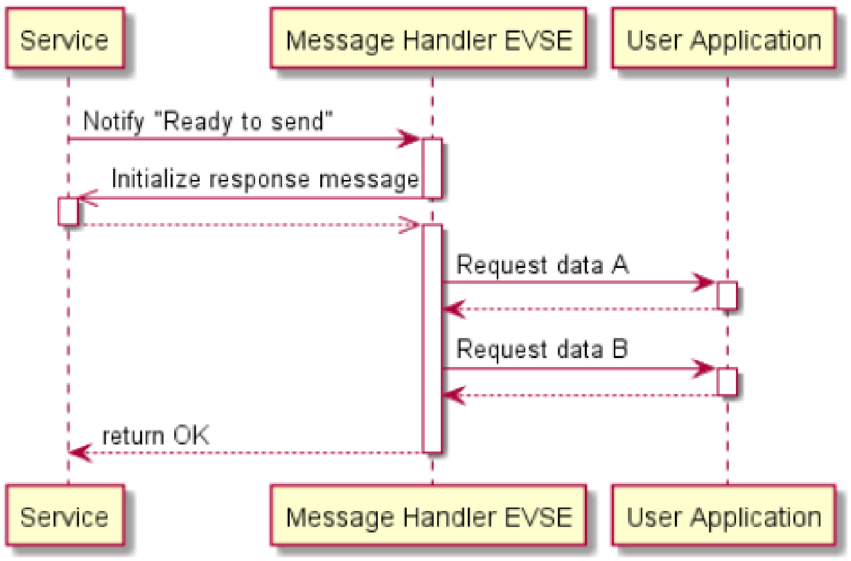


Figure 23: Handling ISO 15118 response. Extracted from SEVENSTAX GmbH, 2021

Preparing an ISO 15118 response also works similarly (Figure 23). After completely processing the request, the Service allocates resources for a response, and it notifies the Message Handler EVSE about the response message. Again, the Message Handler EVSE requests the parameters of the response from the application via callback functions.

4.4 SEVENSTAX's OCPP Stack Operation

The OCPP stack functions similarly to the ISO 15118 stack, except for one aspect. Instead of setting and getting individual parameter from the application layer, the Service module of the OCPP stack sends the complete received message to the application layer and let the application layer retrieves the parameters out of the message. Similarly, the Service allocates the data for the response and requires the application to fill in the data by itself. The reason for this implementation is that the OCPP protocol has more messages than ISO 15118; on average, each OCPP message has more parameters than the ISO 15118 message. Hence, setting and getting individual parameters are not practical.

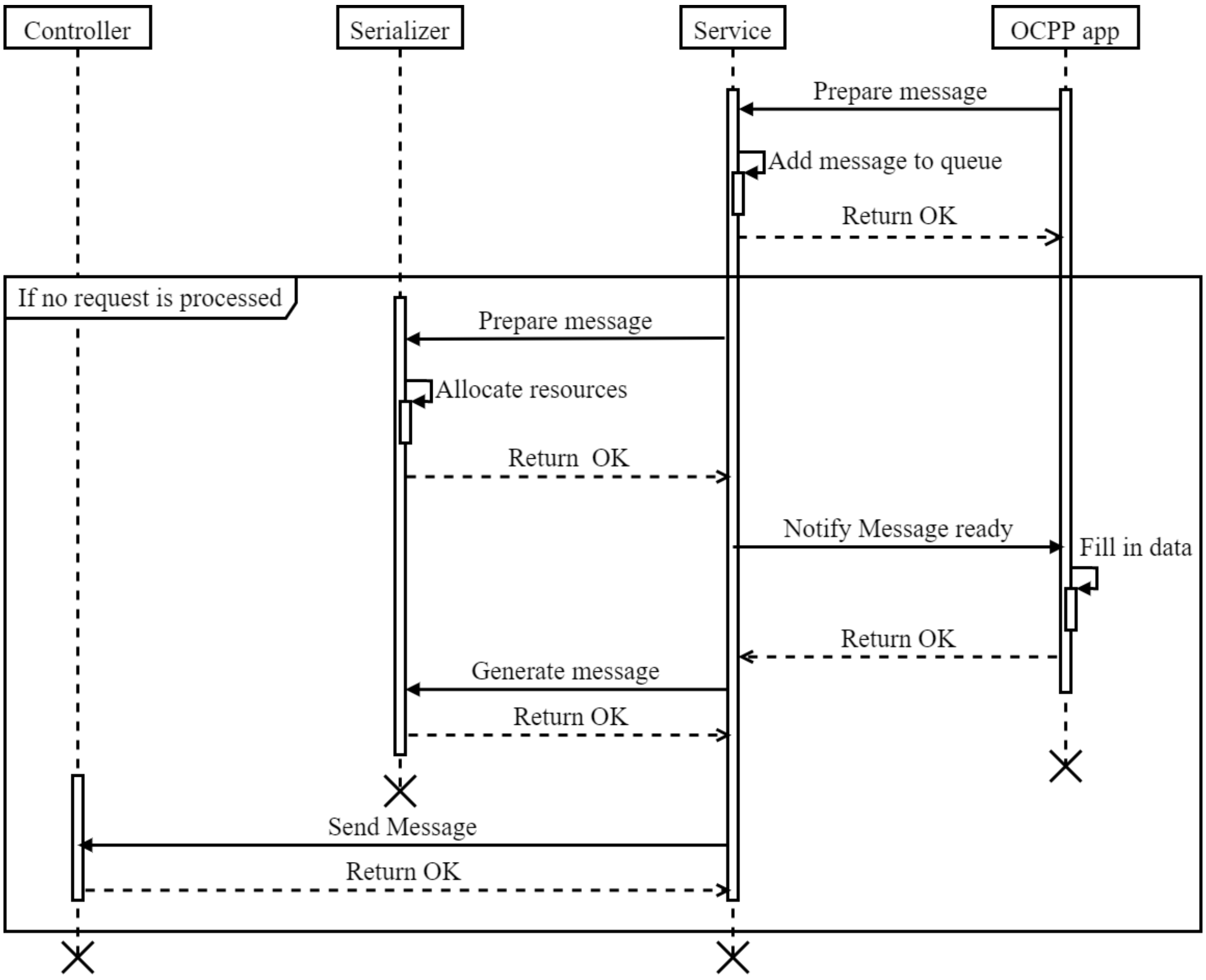


Figure 24: Transmitting OCPP message

Figure 24 illustrates the sequence of sending an OCPP message to the counterpart. Firstly, the OCPP application must issue a "Prepare message" to the Service module, and the Service module adds the command to the queue. The reason for queuing the message is that Open Charge Alliance, 2020d specifies that only one request can be sent out, and the OCPP stack is blocked until it gets a response or a timeout. As long as no request is being processed, the Service module commands the Message module to prepare resources for generating a new message. When the message is ready, the Service module notifies the OCPP application so that the application can fill in the data according to the generated OCPP message. Then, the Service module commands the Message module to serialize the message from C-structure to JSON format. When the serialization is completed, the JSON message can be sent away by the Controller module.

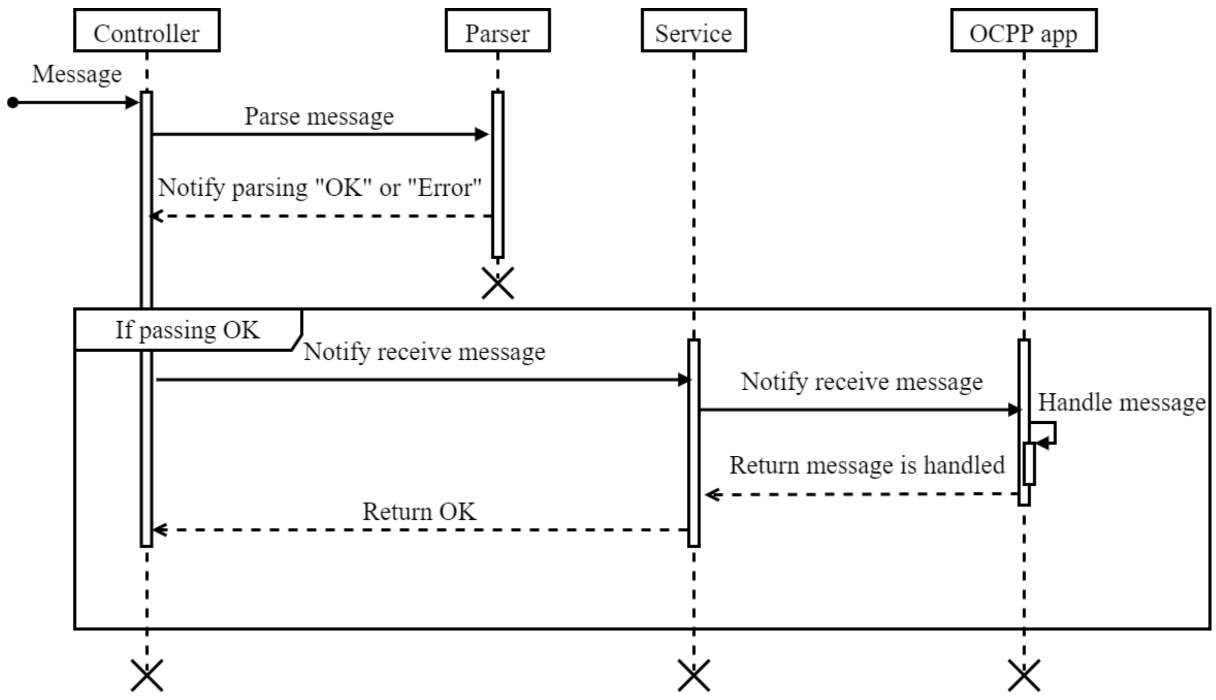


Figure 25: Receiving OCPP message

Receiving an OCPP message requires fewer steps than transmitting one (Figure 25). Whenever the controller receives an OCPP message, it triggers the parsing process. When the message is converted into the C data structure, the Controller notifies the Service, which, in turn, notifies the application about the newly received message. Then, the application is responsible for reads out the data in the message and acts accordingly.

4.5 The Bridging Application's Design

4.5.1 Structure of the Bridging Application

At first thought, there is no need to create a bridging application with the current implementation of the OCPP stack. The ISO 15118 application can transmit and receive OCPP messages directly by interacting with the OCPP Service (Figure 26).

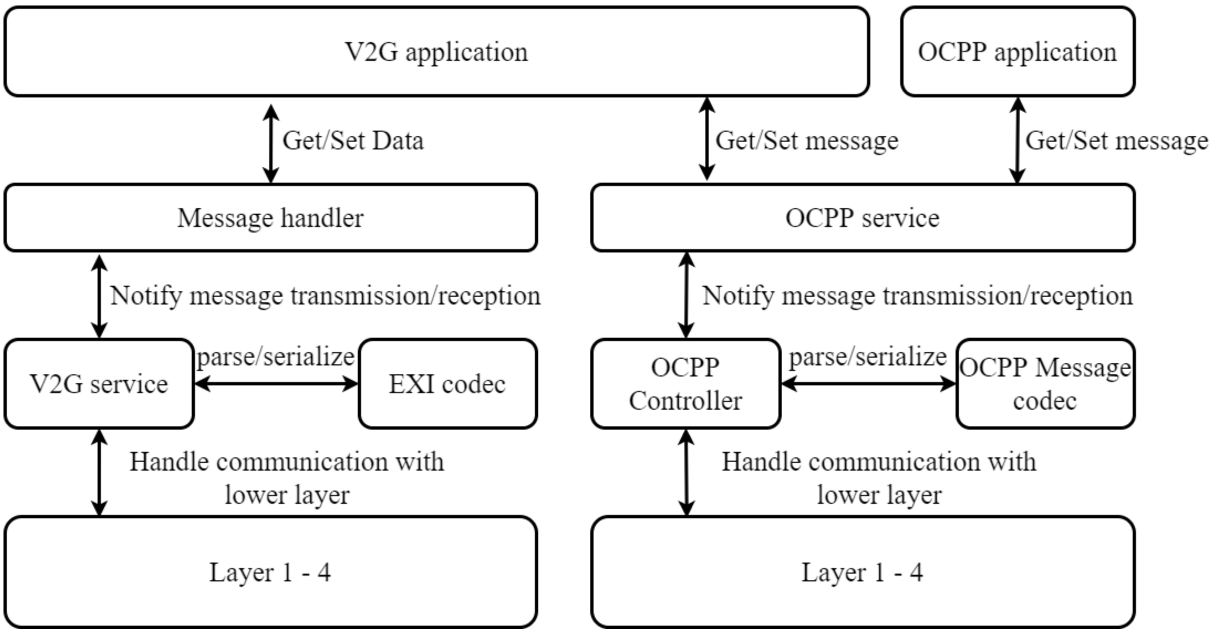


Figure 26: Application without the bridging application

However, this design exposes several flaws, such as:

- The ISO 15118 application must also handle the message sequences for transmitting and receiving OCPP messages, which exponentially increases the complexity of the application.
- The ISO 15118 contains many dependencies from the OCPP module, and there will be much work to be done if the V2G application is migrated to a new OCPP stack. On the other hand, it is also challenging to make the OCPP stack support new ISO 15118 standards.
- The ISO 15118 application gains access to all OCPP messages, which are mostly unnecessary for the interaction between ISO 15118 and OCPP. The required OCPP messages are mentioned in Chapter 2.5.

Therefore, a Bridging application is required to reduce the system's complexity. The Bridging application acts as a facade layer, which provides the interfaces to send and receive ISO 15118-related-OCPP messages. Internally, the bridging application works with the OCPP service module to ensure that messages are transmitted and received correctly. This design has the following advantages compared to the previous one:

- Instead of having one large and complicated application, it is broken down into smaller applications, each with a single responsibility (divide and conquer).

- The Bridging application acts as a facade layer, hiding the detailed implementation of the OCPP stack from the ISO 15118 developers. The developers are provided with a set of function calls (Figure 28) to trigger the OCPP message generation, and they only need to know the required information according to the specific OCPP message. This information is provided in Table 3 and Open Charge Alliance, 2020b.
- The bridging application can control which OCPP messages can be sent or received according to Chapter 2.5. Additionally, it provides the helper functions to convert ISO 15118 to OCPP data types and vice versa.
- This implementation is open for extension. If the new standard (ISO 15118-20) requires new messages, new functions can be easily added.

Consequently, the block diagram from Figure 26 is transformed into the new design in Figure 27.

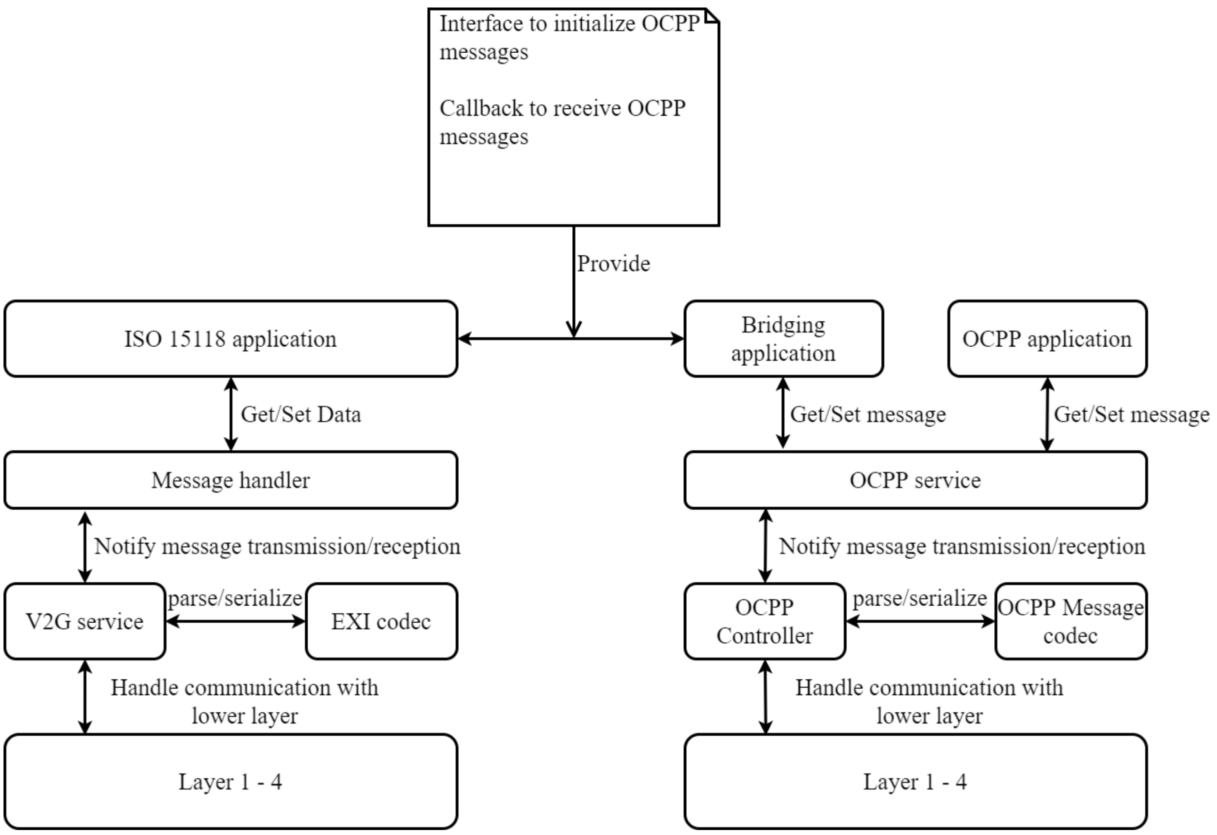
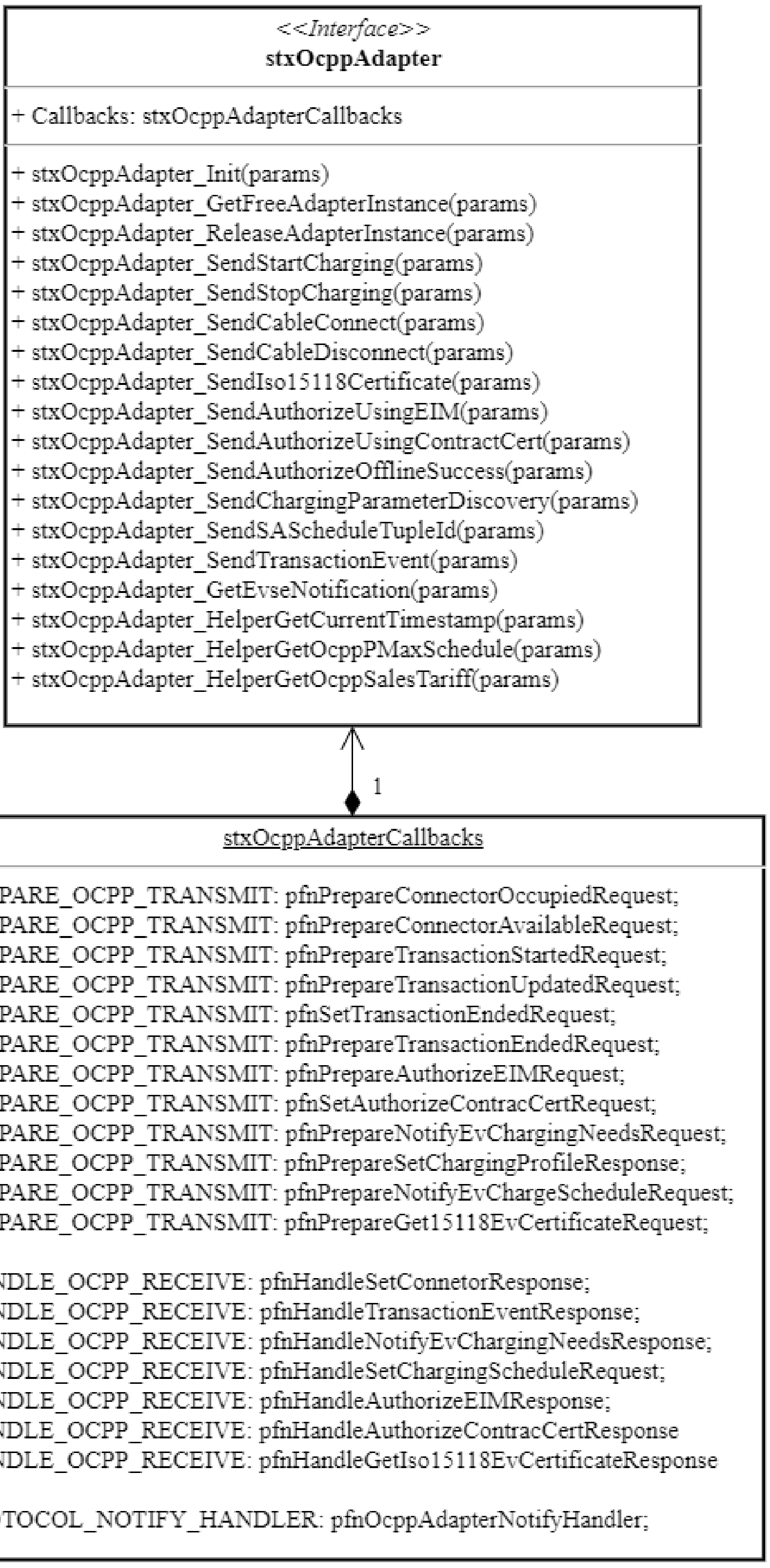


Figure 27: Application with the bridging application

4.5.2 Interfaces Design and Descriptions

Figure 28 illustrates the interfaces and callbacks provided by the Bridging application. The *stxOcppAdapter_Send* methods allow the ISO 15118 application to initiate an OCPP message. When the resource for the initiated message is available, the ISO 15118 application is notified by a callback associated with that message.



```
<<Interface>>
stxOcppAdapter

+ Callbacks: stxOcppAdapterCallbacks

+ stxOcppAdapter_Init(params)
+ stxOcppAdapter_GetFreeAdapterInstance(params)
+ stxOcppAdapter_ReleaseAdapterInstance(params)
+ stxOcppAdapter_SendStartCharging(params)
+ stxOcppAdapter_SendStopCharging(params)
+ stxOcppAdapter_SendCableConnect(params)
+ stxOcppAdapter_SendCableDisconnect(params)
+ stxOcppAdapter_SendIso15118Certificate(params)
+ stxOcppAdapter_SendAuthorizeUsingEIM(params)
+ stxOcppAdapter_SendAuthorizeUsingContractCert(params)
+ stxOcppAdapter_SendAuthorizeOfflineSuccess(params)
+ stxOcppAdapter_SendChargingParameterDiscovery(params)
+ stxOcppAdapter_SendSAScheduleTupleId(params)
+ stxOcppAdapter_SendTransactionEvent(params)
+ stxOcppAdapter_GetEvseNotification(params)
+ stxOcppAdapter_HelperGetCurrentTimestamp(params)
+ stxOcppAdapter_HelperGetOcppPMaxSchedule(params)
+ stxOcppAdapter_HelperGetOcppSalesTariff(params)

1

stxOcppAdapterCallbacks

PREPARE_OCPP_TRANSMIT: pfnPrepareConnectorOccupiedRequest;
PREPARE_OCPP_TRANSMIT: pfnPrepareConnectorAvailableRequest;
PREPARE_OCPP_TRANSMIT: pfnPrepareTransactionStartedRequest;
PREPARE_OCPP_TRANSMIT: pfnPrepareTransactionUpdatedRequest;
PREPARE_OCPP_TRANSMIT: pfnSetTransactionEndedRequest;
PREPARE_OCPP_TRANSMIT: pfnPrepareTransactionEndedRequest;
PREPARE_OCPP_TRANSMIT: pfnPrepareAuthorizeEIMRequest;
PREPARE_OCPP_TRANSMIT: pfnSetAuthorizeContracCertRequest;
PREPARE_OCPP_TRANSMIT: pfnPrepareNotifyEvChargingNeedsRequest;
PREPARE_OCPP_TRANSMIT: pfnPrepareSetChargingProfileResponse;
PREPARE_OCPP_TRANSMIT: pfnPrepareNotifyEvChargeScheduleRequest;
PREPARE_OCPP_TRANSMIT: pfnPrepareGet15118EvCertificateRequest;

HANDLE_OCPP_RECEIVE: pfnHandleSetConnetorResponse;
HANDLE_OCPP_RECEIVE: pfnHandleTransactionEventResponse;
HANDLE_OCPP_RECEIVE: pfnHandleNotifyEvChargingNeedsResponse;
HANDLE_OCPP_RECEIVE: pfnHandleSetChargingScheduleRequest;
HANDLE_OCPP_RECEIVE: pfnHandleAuthorizeEIMResponse;
HANDLE_OCPP_RECEIVE: pfnHandleAuthorizeContracCertResponse
HANDLE_OCPP_RECEIVE: pfnHandleGetIso15118EvCertificateResponse

PROTOCOL_NOTIFY_HANDLER: pfnOcppAdapterNotifyHandler;
```

Figure 28: the interfaces provided by the bridging application

For example, the ISO 15118 application can call the *stxOcppAdapter_SendCableConnect* method to notify the CSMS that the cable is connected to the EV. When the resource is ready, the ISO 15118 application is notified by the *pfnConnectorOccupiedRequest* callback. The callback is a function pointer of type *PREPARE_OCPP_TRANSMIT*:

```
typedef void(*PREPARE_OCPP_TRANSMIT)(void*pstOcppMsg,
                                     bool*bResult);
```

with:

- *pstOcppMsg*: pointer to the message, where the related ISO 15118 parameters are set to the OCPP parameters.
- *bResult*: return status

If the unmatched data types occur, they are converted in this callback according to Table 10
The ISO 15118 application can be notified when it receives a response from the CSMS.by registering to one of the *HANDLE_OCPP_RECEIVE* callbacks. For example, *pfnHandle-SetConnectorResponse* is the response callback for both *stxOcppAdapter_SetCableConnect* and *stxOcppAdapter_SetCableDisconnect*.

```
typedef void(*HANDLE_OCPP_RECEIVE)(void*pstOcppMsg,
                                   bool*bResult);
```

with:

- *pstOcppMsg*: pointer to the message containing the OCPP response. The related ISO 15118 parameters are extracted from the OCPP parameters.
- *bResult*: return status

If any data type mismatched occurs, it is converted from OCPP type to ISO 15118 type in this callback, based on Table 10.
The *PROTOCOL_NOTIFY_HANDLER* provides a callback to notify the ISO 15118 application of the following events:

- *OCPP_ADAPTER_NC_DISCONNECTED*: The bridging application is disconnected from the CSMS due to network connectivity.
- *OCPP_ADAPTER_NC_CONNECTED*: The bridging application is connected to the CSMS.

- *OCPP_ADAPTER_NC_AUTHORIZE_TIMEOUT*: Timeout event while waiting for authorization response.
- *OCPP_ADAPTER_NC_CHARGE_PROFILE_TIMEOUT*: Timeout event while waiting for the charge profile.

With this interface design, new interfaces and callbacks can be added to the Bridging Application whenever new messages must be supported.

4.5.3 Sequence for Handling the Requests and the Responses

Figure 29 illustrates the process of sending an OCPP message to the CSMS. After the ISO 15118 application receives all of the parameters from its service layer, it initiates an OCPP message by calling one of the *stxOcppAdapter_Send* methods in Figure 28, e.g., *stxOcppAdapter_SetCableConnect*. Internally, the Bridging application verifies if *stxOcppAdapter_SetCableConnect* can be called during the current state. For example, *stxOcppAdapter_SetCableConnect* must be called before providing the energy to the EV. If the state is valid, the Bridging application prepares the OCPP message, which is the *StatusNotificationRequest* (Figure 12), corresponding to *stxOcppAdapter_SetCableConnect*. After the Service allocates the resources for the initiated message, it notifies the Bridging Application. Then, the Bridging application notifies that the message is ready via the callback *pfnSetConnectorOccupiedRequest*. The task of the ISO 15118 application is to set the ISO 15118-related data to the OCPP parameters according to the requirement of the JSON schema (Open Charge Alliance, 2020c). Additionally, any data type mismatch between ISO 15118 parameters and OCPP parameters is converted by the ISO 15118 application based on the suggestion in Table 10. Before sending the OCPP message away, the Bridging application verifies whether the required parameters are presented in the OCPP messages according to Open Charge Alliance, 2020c. If errors occur during the verification, the Bridging Application notifies the ISO 15118 application via the *pfnOcppAdapterNotifyHandler* callback.

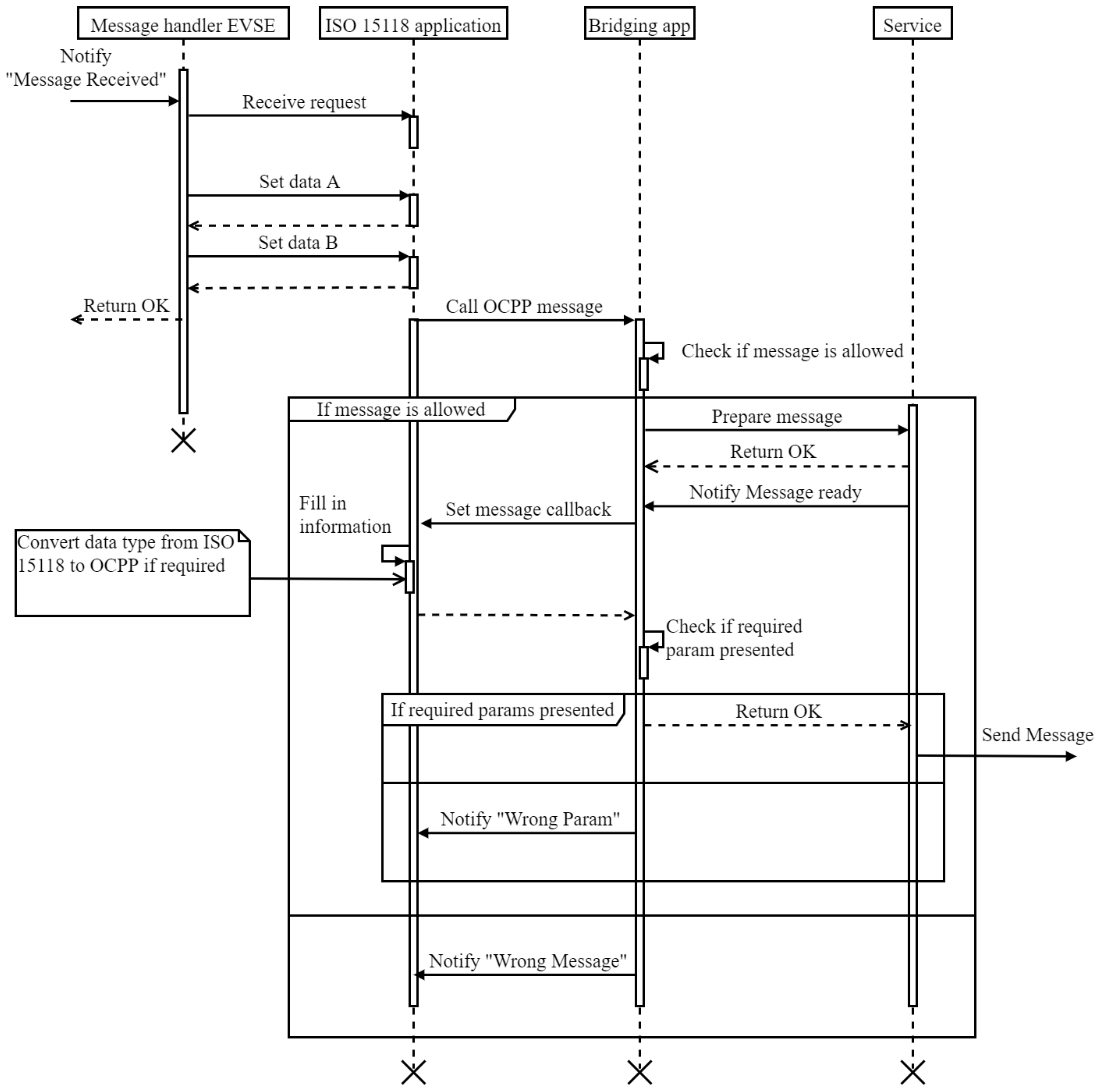


Figure 29: Bridging application handles transmitting message

After sending the OCPP message, the ISO 15118 application waits for the response from the CSMS. Within the timeout value, the Message Handler EVSE starts the procedure by checking whether the response has arrived. If the OCPP response has not yet arrived, the ISO 15118 application prepares an ISO 15118 response with Busy status. When the Bridging application receives the OCPP messages from its Service module, it verifies if the OCPP message is allowed in the current state. Then, the OCPP message is forwarded to the ISO 15118 application. The ISO 15118 application converts any mismatched data type from OCPP type to ISO 15118 type based on Table 10, stores the necessary parameters, and is ready for the subsequent request from the Message Handler EVSE.

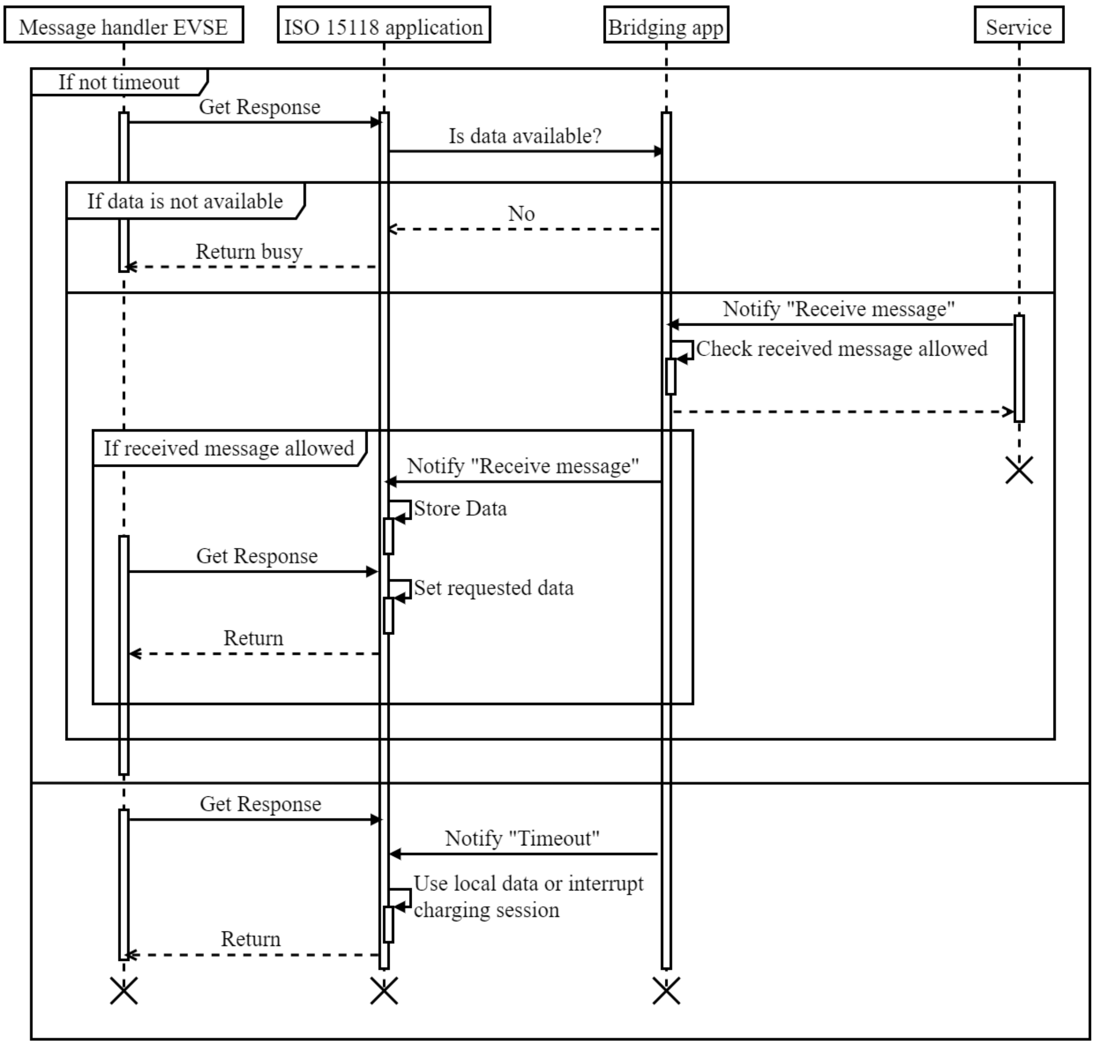


Figure 30: Bridging application handles receiving message

4.6 Error Handling

Besides enabling data exchange between protocols, the Bridging Application must handle the errors from both sides. Error handling can be divided into two categories. The first category is the errors coming from ISO 15118, namely, what happens if the ISO 15118 communication is interrupted; for instance, the driver unplugged the charging cable. The second one is from OCPP, specifically what happens if the OCPP communication is interrupted due to no network connection.

4.6.1 ISO 15118-related Error Handling

From the OCPP communication point of view, whenever the communication between the EV and EVSE is interrupted during the energy delivery phase, two statuses must be reported to the CSMS:

- Status of the connector
- Status of the transaction

The status of the transaction can be reported with the OCPP message *TransactionEventRequest*, in which the developers can specify the event type, the trigger reason, and the stopped reason. The *TransactionEventRequest* message lets the CSMS know that the energy offered to the EV has been stopped. Whenever the cable is unplugged, the *StatusNotificationEventRequest* notifies the CSMS that the connector is available again. Those handlings are specified in use cases E9 and E10 by Open Charge Alliance, 2020b. For the Bridging application, they are equal to calling the functions *stxOcppAdapter_SendStopCharging* and *stxOcppAdapter_SendCableDisconnect*. Then, the ISO 15118 application can fill in the data of those messages according to Figure 29.

4.6.2 OCPP-related Error Handling

For the OCPP communication interruption, several use cases must be handled: during certificate handling, during authorization, during charging profile negotiation, and during the charging loop.

Error handling during certificate handling phase
Open Charge Alliance, 2020b does not specify how to handle the OCPP connection interruption during the certification handling phase. Therefore, it can be assumed as follows:

- The charging station is offline before the certificate handling phase.
- The charging station is offline during the certificate handling phase.

If the charging station is offline before the certificate handling phase, the ISO 15118 application is already notified of the offline status by the Bridging application so that the ISO 15118 application can switch to other solutions, such as EIM. If the charging station is offline during the certificate handling phase, the ISO 15118 application will receive either a ”timeout” or “disconnected” notification from the Bridging application. The ISO 15118 application must then terminate the charging session.

Error handling during authorization phase
If authorization must be performed when the charging station has no connection to the CSMS, the charging station must switch to authorization using cached data or authorization using local list to authorize the EV. Use cases C13 and C15 of the Open Charge Alliance, 2020b specify the handling in detail. Nothing is to be done for the Bridging

application.

Error handling during charging profile negotiation phase
If the charging station loses its connection to the CSMS prior to or during the charging profile negotiation, the use cases K16 and K17 in Open Charge Alliance, 2020b specify that the charging station must deliver the energy to the EV according to the default or local charging profile. The Bridging Application must only notify the ISO 15118 Application that the connection is lost so that ISO 15118 Application switches to the local or default charging profile. The local or default charging profile is up to the application users to define. When the connection is later recovered, the CSMS can perform load balancing by issuing a new charging profile via the SetEvCharingProfileRequest message (Figure 16).

Error handling during charging loop
If the OCPP communication is interrupted during the charging loop, Open Charge Alliance, 2020b specifies in use cases E04, E08, and E12 that all *TransactionEventRequest* messages must be queued in the memory. These messages are then sent to the CSMS when the station is online again. From the software implementation perspective, the Bridging application and the Service module handle this situation automatically. When the station is offline, the *stxOcppAdapter_SendTransactionEvent* function commands the Service module to queue the OCPP messages in the memory instead of sending them. When the station comes back online, the Service module automatically checks for the queued messages and sends them to the CSMS.

4.7 Discussion

The Bridging application is designed to resolve the data type incompatibilities and undesired states found in the theoretical compatibility analysis when the ISO 15118 and OCPP protocols interact. In addition, during the development of the Bridging application, other problems were found, such as:

- No standardized use case to report the faulty state of the EV or EVSE
- High memory consumption of OCPP messages
- Nested conditions of OCPP messages
- Software complexity due to the serializer and parser of both protocols

The following sections discuss these problems in detail and offer suggestions for resolving them.

At the protocol level, although OCPP specifies the messages to support ISO 15118, these messages focus only on authorization, authentication, and transaction, but lack standardized use cases for reporting EV or EVSE faults. There are, of course, some messages for reporting general status. For example, the OCPP *NotifyEventRequest* message allows the charging station to report its events to the CSMS. However, this message does not have a standardized payload, and the implementation is left up to the application developers. If the message is not standardized, and in the event of a change to a new charging station operator, the new system may not understand this custom message. The Open Charge Alliance, 2020b also provides the *StatusNotificationRequest* message, which allows the charging station to report the status of its connector to the CSMS; however, the reported status is limited to the enumeration value: *Available*, *Occupied*, *Reserved*, *Unavailable* and *Faulted*. Therefore, a standard for fault reporting in general and for ISO 15118 needs to be developed to increase interoperability between systems.

During the development, the size of the OCPP C-structure messages is measured using the *sizeof()* function. Table 11 shows the ten-largest-size messages in terms of memory allocation. Out of the ten messages, the largest message (*stxReportChargingProfilesRequest*) consumes 495940 bytes, and the smallest message (*stxGetVariablesResponse*) consumes 18452 bytes. Three messages (*stxNotifyEVChargingScheduleRequest*, *stxTransactionEventRequest*, and *stxSetChargingProfileRequest*) from the list are used for the data exchange between ISO 15118 and OCPP. On a larger system, such as a microprocessor running embedded Linux, the large message size poses no problem because those systems have enough resources to allocate the messages, but it is not possible to integrate or limitedly integrate OCPP into a small embedded system running a real-time operating system or bare metal. The reason for the large-size messages is that they have a complicated structure, where objects and arrays of objects are nested together. Furthermore, the objects themselves also have a complex structure (Open Charge Alliance, 2020c).

Table 11: Ten largest-size message structures

C-structure message	Size (bytes)
stxReportChargingProfilesRequest	495940
stxNotifyChargingLimitRequest	165256
stxRequestStartTransactionRequest	127528
stxSetChargingProfileRequest	124176
stxTransactionEventRequest	98788
stxMeterValuesRequest	96456
stxNotifyReportRequest	54888
stxNotifyEVChargingScheduleRequest	41468
stxSendLocalListRequest	19152
stxGetVariablesResponse	18452

From the software point of view, the message size can be optimized by compiling them with the ”packed” option to prevent the compiler from padding data to the structure. For example, in GCC:

```
struct __attribute__((__packed__)) foo
{
    void        *p;
    char        c;
    uint32_t    x;
};
```

Additionally, the number of objects in an array in a OCPP message can be reduced to save memory. For instance, in the *SetChargingProfileRequest*’s schema (Open Charge Alliance, 2020c), the size of *chargingSchedulePeriod* attribute is specified as follows:

```
"chargingSchedulePeriod": {
    "type": "array",
    "additionalItems": false ,
    "items": {
    "$ref": "#/definitions/ChargingSchedulePeriodType"
    },
    "minItems": 1,
    "maxItems": 1024
}
```

The developers can specify the maximum number of items depending on the system’s capabilities. However, this solution will encounter problems when the charging stations are migrated to a new CSMS that supports more items than specified in the charging station. In SEVENSTAX’s implementation, the OCPP C-structure messages and objects are automatically generated by a code generator, so the code generator can be extended to support this feature. From the hardware point of view, the message size problem can be easily solved by running the OCPP (and ISO 15118) stack on a high-performance hardware platform.

OCPP is designed as a stateless protocol with the intention of supporting various front-end protocols (protocol for the communication between EV and EVSE) and giving the users the flexibility to employ the OCPP messages according to their needs. However, OCPP specifications do not specify the purpose of each message but only specify where that message is applied (OCPP use cases). Furthermore, that message shall be sent only if specific conditions are satisfied in specific use cases. For example, StatusNotificationRequest is applied in twenty-six use cases (Table 12), and B01 - Cold Boot Charging Station use case specifies that *StatusNotificationRequest* shall be sent only when the *BootNotificationResponse* containing the status *Accept*. The nested condition increases the complexity of the application since it must check for multiple conditions before transmitting or receiving a message. For example:

```
statusNotifyReq = GetStatusNotificationRequest();
if(state == cold_boot)
{
    if(boot_status == accepted)
    {
        /* Fill in the statusNotifyReq's parameters with data */
    }
}
else if (state == state_transaction)
{
    ...
}
else if ( other condition...)
{
    ...
}
SendOcppMessage(statusNotifyReq);
```

The OCPP stack copes with this issue by instead of having one big application (Figure 10), the OCPP application can be divided into smaller use-case-dedicated applications (Figure 31) by using the OCPP Service module. The OCPP Service keeps track of which application sends which OCPP message so that it can return the response to the originating application accordingly. Therefore, the same message can be handled differently depending on the use case and without various condition checking.

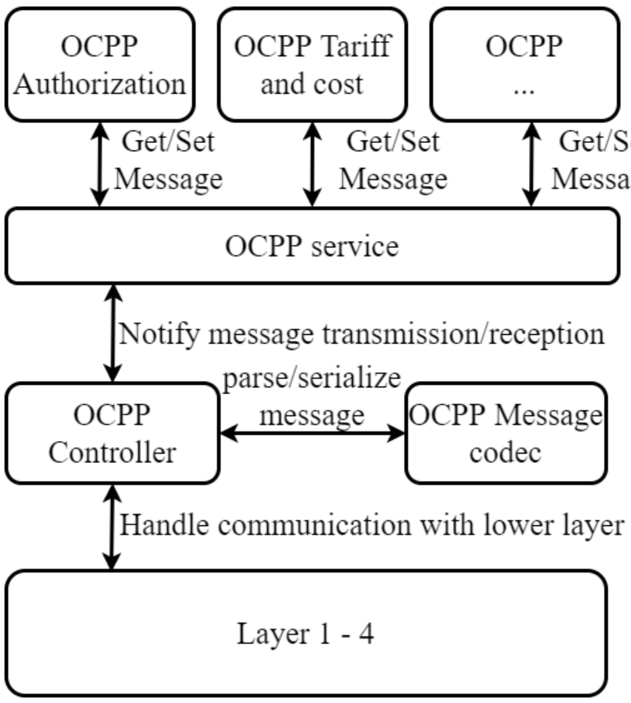


Figure 31: OCPP application block diagram

The OCPP and ISO 15118 stacks expose a complex message sequence (Figure 22, 23, 24, 25) because of the message's serializing and parsing processes. The steps for serializing and parsing data are described as follows:

- If the serializer is available, the application occupies it. Otherwise, the application must wait until the next cycle.
- The serializer allocates the fixed resource for encoding the message.
- The serializer notifies the application that the resource is available
- The application creates the message by adding information to it.
- The serializer encodes the message using dedicated libraries (JSON and EXI) and sends it away.
- When a message is received, and the parser is available, it allocates the fixed resource. Otherwise, the message is queued.
- The parser decodes the message using dedicated libraries (JSON and EXI).

- The parser notifies the application that the message is decoded and ready for processing.

In addition, OCPP uses JSON, which is a popular data format on the Web because of its readability and ease of use; however, it adds much overhead for increasing the readability. In charging communication, most messages are processed by the machine without human intervention, so there is no urgent need for a human-readable data format. Readability is helpful during development or debugging, but it must not overshadow the efficiency and resource consumption of the system, especially when the protocols are used in embedded systems.

The data formats are embodied in the ISO 15118 and OCPP specifications, and therefore there is no possibility of changing them. However, to reduce complexity, an efficient non-encoding-decoding data format could be used instead of JSON and EXI. One suggestion for such a data format is FlatBuffers (Google, n.d.-a), originally developed at Google for gaming or performance-critical applications. The key feature of FlatBuffers is that they can access serialized data without having to parse or unpack it, making them more efficient than other data formats (Google, n.d.-b). In addition, many FlatBuffer compilers are available (Google, n.d.-c) to convert the FlatBuffer schema into target code, so that message structures are always the same regardless of programming language and implementation.

4.8 Chapter's Summary

This chapter describes the design of the bridging application to show how to bridge the OCPP and ISO 15118 protocols to allow data exchange between the EV and the CSMS during a charging session. The bridging application also solves the problems found in the compatibility analysis, namely data type incompatibility and unwanted state transitions. In addition, this chapter points out several problems that were not detected by the compatibility analysis. For example, the OCPP specification does not specify safety use cases, i.e., there is no specific message sequence for the charging station to report the malfunctions of the EVs or EVSEs. In addition, OCPP messages consume large amounts of memory, making it difficult to run on small embedded devices. Several software optimizations are proposed to overcome this problem, such as reducing the number of elements in OCPP messages or preventing the compiler from padding data to the C-structure. In addition, OCPP application development is complicated because of the nested conditions for sending and receiving OCPP messages. To overcome this problem, SEVENSTAX's OCPP stack has a service layer to break the OCPP application into smaller modules, reducing the complexity of the system. Finally, both ISO 15118 and the OCPP stack are exposed to high complexity due to serializing and parsing the messages. This thesis proposes FlatBuffers as an alternative format to replace JSON and EXI, because of its read-without-unpacking serialized data characteristics.

5 Conclusion and Future Work

5.1 Conclusion

Smart Charging lets the back-end system control the charging process of potentially hundreds of thousands of EVs during peak time to reduce the stress on the local energy grid. Smart Charging can be realized only when the back-end system can communicate to the EVs being charged to influence their charging profile. Various protocols are introduced to solve this challenge. ISO 15118 and OCPP are gaining momentum in the industry as the protocols for communication between EV and EVSE; and between EVSE and the back-end system. Even though both protocols share similarities, such as IP-based protocol, and software design pattern, they must be analysis to see how well they can work together to enable seamless communication between EVs and the back-end system.

The first part of the thesis provides an overview of both protocols to highlight their similarities and differences regarding the network stack, the data format, and the use cases. Then both protocols are composed into a message sequence illustrating their interactions. The message sequence is transformed into the model comprising states and messages, and based on the model, this thesis analyzes the compatibility of the protocols using the method proposed by the Ouederni et al., 2010. Based on this method, the author developed a prototype tool to automate the calculation of the compatibility degree.

The outcome of the analysis shows that both protocols are compatible with each other in terms of state transitions. However, their compatibility degrees are low because of incompatible data types among exchanged parameters. The reason is that ISO 15118 uses some domain-specific data types for its parameters; meanwhile, OCPP uses generic data types with the intention of supporting as many front-end protocols as possible. The incompatible data type shows that when integrating both protocols into a system, the protocols integrators must do data type conversion whenever the data type incompatibility occurs to ensure the protocols' interoperability. Furthermore, the integrators must apply a mechanism to monitor the message exchange between the protocols to prevent unwanted state transitions.

In the second part, this thesis describes the design concept of the Bridging application between the ISO 15118 and OCPP protocol. The Bridging application provides a facade interface to the ISO 15118 application with the goals such as hiding the complexity of the OCPP stack, verifying the validity of states and messages, providing helper functions to convert data between the protocols, and handling the errors. The bridging application also demonstrates how the problems in the compatibility analysis can be solved by its design and interfaces. Furthermore, the development of the Bridging application also points out further problems overlooked by the compatibility analysis.

The Bridging application development points out that there is no standard or specification for reporting the malfunction of the EVs or the charging stations. Fault reporting allows the Smart Charging participants to act promptly when incidents occur. The size of the OCPP messages is measured, and the result indicates that it is not possible to deploy the OCPP stack in the small embedded systems without tweaking the OCPP messages due to limited resources. Therefore, running the OCPP stack on a more resource-capable platform is recommended, i.e., running the OCPP stack on a processor with an Operating System instead of a microcontroller. The thesis also points out that using JSON and EXI as data formats makes the OCPP and ISO 15118 stacks more complex because these data formats require serializers and parsers. Furthermore, JSON is a human-readable format that adds unnecessary overhead since the messages are processed by machines, not humans. The suggestion for this challenge is that, in the future, protocols designers can switch to Flatbuffer, which is boasted for its efficiency and low memory footprint.

In conclusion, this thesis points out that, regardless of the differences in data types, ISO 15118 and OCPP are compatible and can be used to realize Smart Charging. When developing a Bridging Application to bridge the protocols, the protocol adaptors must convert the data type between protocols, handles the exchanged message to prevent unwanted state transitions, and handles the complexity of parsing and serializing data.

5.2 Furture Work

Although the thesis achieved its goals of analyzing the compatibility of the protocol and designing a protocols Bridging application, several areas can be improved in the future to obtain better results.

The parameters' depth in a data structure must be analyzed in the parameter compatibility calculation since they affect the compatibility of the protocol and increase the complexity of the protocol bridging application. Then, the $param\_comp(pl_1, pl_2)$ becomes:

$$par\_comp(pl_1, pl_2) = \frac{number(pl_1, pl_2) + order(pl_1 and pl_2) + type(pl_1, pl_2) + depth(pl_1, pl_2)}{4}$$

Where $depth(pl_1, pl_2)$ is the depth analysis between the parameter lists pl_1 and pl_1

The compatibility calculation prototype tool (Nguyen, 2023) is developed and tested intensively based on the calculation presented in this thesis. Therefore, it must be extended to accept the calculation based on other compatibility notations. The user experience can be improved by designing the graphical interface, where users can design and edit their model instead of describing the model with JSON files. Result reporting can be improved to show not only the step-by-step calculations and result matrices but also illustrates which messages and transitions contribute to the incompatibility.

The bridging application is designed based on common software design practices, SEVENSTAX's design guidelines, and SEVENSTAX's software architecture. However, several works, such as Tan et al., 2009, Yellin and Strom, 1997, and Nejati et al., 2007, propose methods and techniques for service composition and protocol adaptor implementation. Those methods and techniques can be applied to build the composition model or an adaptor component, which can be used to realize the software implementation. Additionally, those methods and techniques can be integrated into the prototype tool so that the tool can model an adaptor component and potentially generate the code template for the software implementation.

References

Aalst, W. M. P. v. d., Mooji, A. J., Stahl, C., & Wolf, K. (2009). Service interaction: Patterns, formalization, and analysis. In *Formal methods for web services* (pp. 42–88).

AMPECO. (n.d.). *Smart charging: All you need to know.* Retrieved March 5, 2023, from https://www.ampeco.com/guides/smart-charging/

Bermejo, C., Geissmann, T., Möller, T., Nägele, F., & Winter, R. (2021). *The impact of electromobility on the german electric grid.* Retrieved March 5, 2023, from https://www.mckinsey.com/industries/electric-power-and-natural-gas/our-insights/the-impact-of-electromobility-on-the-german-electric-grid

Brand, D., & Zafiropulo, P. (1983). On communicating finite-state machines. *Journal of the Association for Computing Machinery*, 323–342.

DIN. (2019). Konduktive ladesysteme für elektrofahrzeuge - teil 1: Allgemeine anforderungen.

Elabd, E., Coquery, E., & Hacid, M.-S. (2009). Compatibility and replaceability analysis of timed web services protocols. *2009 Second International Conference on Computer and Electrical Engineering*, *2*, 15–19. https://doi.org/10.1109/ICCEE.2009.106

Engel, H., Hensley, R., Knupfer, S., & Sahdev, S. (2018). *The potential impact of electric vehicles on global energy systems.* Retrieved March 5, 2023, from https://www.mckinsey.com/industries/automotive-and-assembly/our-insights/the-potential-impact-of-electric-vehicles-on-global-energy-systems

Gao, C., & Wei, J. (2011). Checking compatibility of context-aware service protocols. *Proceedings of 2011 IEEE 6th International Symposium on Service Oriented System (SOSE)*, 335–340. https://doi.org/10.1109/SOSE.2011.6139125

Google. (n.d.-a). Flatbuffers. Retrieved March 12, 2023, from https://google.github.io/flatbuffers/index.html

Google. (n.d.-b). Flatbuffers benchmarks for c++. Retrieved March 12, 2023, from https://google.github.io/flatbuffers/flatbuffers_benchmarks.html

Google. (n.d.-c). Platform/language/feature support. Retrieved March 28, 2023, from https://google.github.io/flatbuffers/flatbuffers_support.html

Hoekstra, A., Bienert, R., Wargers, A., Singh, H., & Voskuilen, P. (n.d.). *Using openadr with ocpp.* https://www.openchargealliance.org/uploads/files/OCA-Using-OpenADR-with-ocpp.pdf

ISO. (2014). Iso15118 - 2: Network and application protocol requirements.

ISO. (2019). Iso15118 - 1: General information and use-case definition.

Kern, D. (2021). *Privacy-preserving architecture for ev charging and billing* (Master's thesis). Technische Universität. Darmstadt. https://doi.org/10.26083/tuprints-00018558

Klapwijk, P., & Driessen, L. (2017). Ev related protocol study.

Manning, C., & Schütze, H. (1999). *Foundations of statistical natural language processing*. MIT Press.

Martens, A. (2003). On compatibility of web services. *Petri Net Newsletter*, *65*(12-20), 100.

Neaimeh, M., & Andersen, P. B. (2020). Mind the gap- open communication protocols for vehicle grid integration. *Energy Informatics*. https://doi.org/10.1186/s42162-020-0103-1

Nejati, S., Sabetzadeh, M., Chechik, M., Easterbrook, S., & Zave, P. (2007). Matching and merging of statecharts specifications. *29th International Conference on Software Engineering (ICSE'07)*, 54–64. https://doi.org/10.1109/ICSE.2007.50

Nguyen, Q. H. (2023). Compatibility calculation application. https://github.com/QuangHaiNguyen/protocol_compatibility_calculation

Open Charge Alliance. (2020a). Ocpp 2.0.1: Part 0 - introduction. https://www.openchargealliance.org/protocols/ocpp-201/

Open Charge Alliance. (2020b). Ocpp 2.0.1: Part 2 - specification. https://www.openchargealliance.org/protocols/ocpp-201/

Open Charge Alliance. (2020c). Ocpp 2.0.1: Part 3 - schema. https://www.openchargealliance.org/protocols/ocpp-201/

Open Charge Alliance. (2020d). Part 4 - json over websockets implementation guide. https://www.openchargealliance.org/protocols/ocpp-201/

Open Charge Alliance & CHAdeMo. (2020). Using ocpp with chademo. https://www.openchargealliance.org/uploads/files/Using-OCPP-with-CHAdeMO.pdf

Ouederni, M., Salaün, G., & Pimentel, E. (2010). *Measuring the compatibility of service interaction protocols - technical report iti 4-10*.

Pedersen, T., Patwardhan, S., & Michelizzi, J. (2004). Wordnet::similarity.

Schmutzler, J., Andersen, C., & Wietfeld, C. (2013). Evaluation of ocpp and iec 61850 for smart charging electric vehicles. *World Electric Vehicle Journal*, *6*, 863–874. https://doi.org/10.3390/wevj6040863

SEVENSTAX GmbH. (2021). Sevenstax v2g user manual, 10–11.

Tan, W., Fan, Y., & Zhou, M. (2009). A petri net-based method for compatibility analysis and composition of web services in business process execution language. *Automation Science and Engineering, IEEE Transactions on*, *6*, 94–106. https://doi.org/10.1109/TASE.2008.916747

The Federal Government. (2020). *Climate-friendly transport*. Retrieved March 5, 2023, from https://www.bundesregierung.de/breg-en/issues/climate-action/climate-friendly-transport-1795842

The Mobility House. (n.d.). Ocpp. https://github.com/mobilityhouse/ocpp

WC3, T. (2002). *Schema for xml signatures*. http://www.w3.org/2000/09/xmldsig#

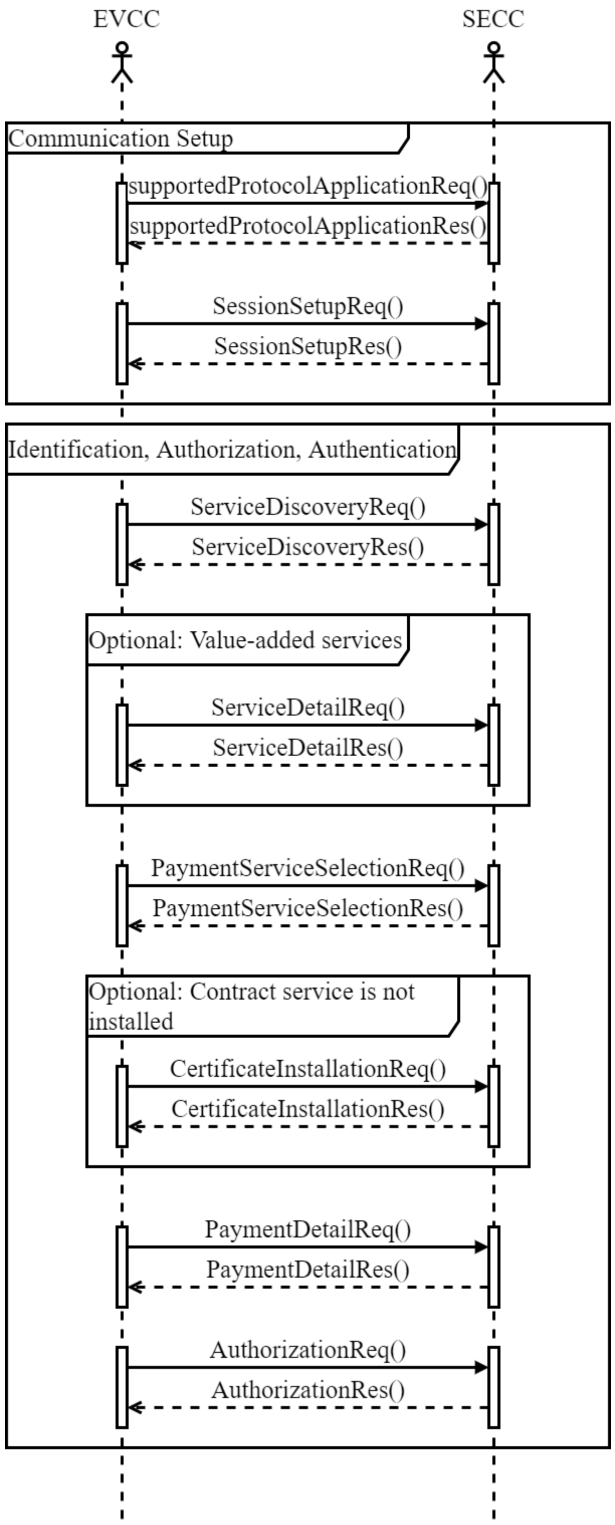
Wellisch, D., Lenz, J., Faschingbauer, A., Pöschl, R., & Kunze, S. (2015). Vehicle-to-grid ac charging station: An approach for smart charging development.

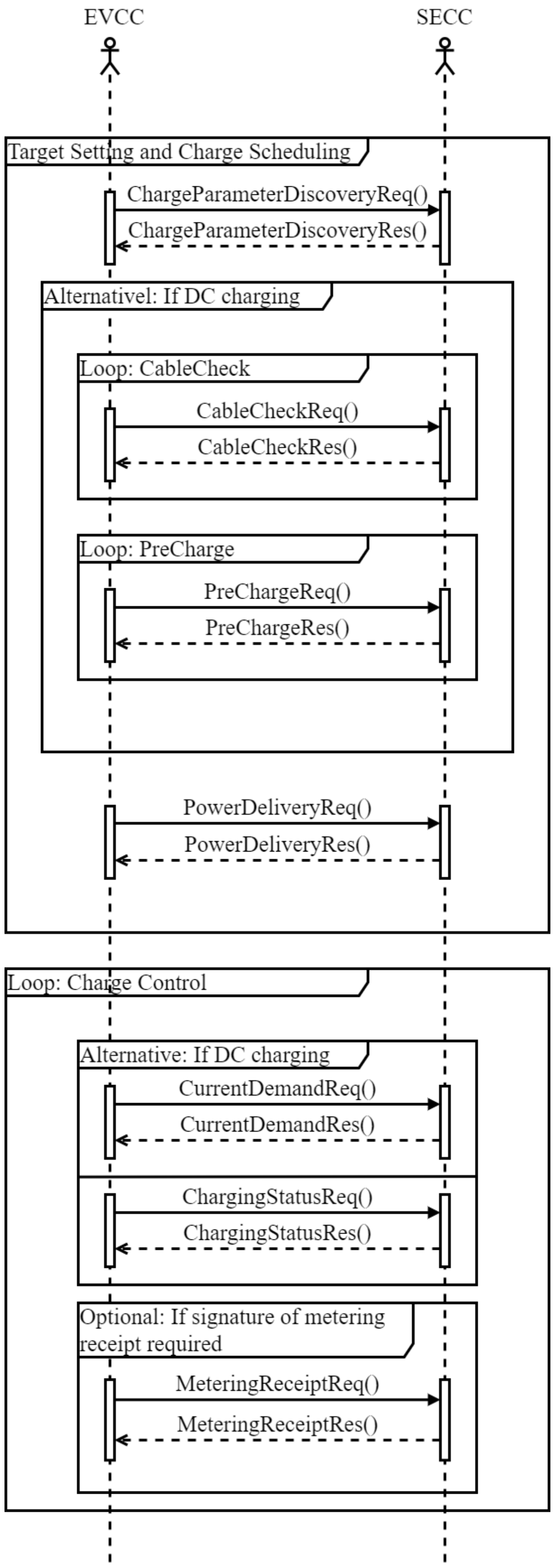
Wu, Z., Deng, S., Li, Y., & Wu, J. (2009). Computing compatibility in dynamic service composition. *Knowledge and Information Systems*, 107–129. https://doi.org/0.1007/s10115-008-0143-5

Yang, Y., Zhang, M., Ye, X., Chen, H., & Lian, X. (2009). Protocol compatibility verification for web services interaction. *2009 Fifth International Conference on Semantics, Knowledge and Grid*, 106–112. https://doi.org/10.1109/SKG.2009.86

Yellin, D. M., & Strom, R. E. (1997). Protocol specifications and component adaptors. *ACM Trans. Program. Lang. Syst.*, *19*(2), 292–333. https://doi.org/10.1145/244795.244801

Appendix A - ISO 15118


EVCC
SECC
Communication Setup
supportedProtocolApplicationReq()
supportedProtocolApplicationRes()
SessionSetupReq()
SessionSetupRes()
Identification, Authorization, Authentication
ServiceDiscoveryReq()
ServiceDiscoveryRes()
Optional: Value-added services
ServiceDetailReq()
ServiceDetailRes()
PaymentServiceSelectionReq()
PaymentServiceSelectionRes()
Optional: Contract service is not installed
CertificateInstallationReq()
CertificateInstallationRes()
PaymentDetailReq()
PaymentDetailRes()
AuthorizationReq()
AuthorizationRes()


EVCC
SECC
Target Setting and Charge Scheduling
ChargeParameterDiscoveryReq()
ChargeParameterDiscoveryRes()
Alternativel: If DC charging
Loop: CableCheck
CableCheckReq()
CableCheckRes()
Loop: PreCharge
PreChargeReq()
PreChargeRes()
PowerDeliveryReq()
PowerDeliveryRes()
Loop: Charge Control
Alternative: If DC charging
CurrentDemandReq()
CurrentDemandRes()
ChargingStatusReq()
ChargingStatusRes()
Optional: If signature of metering receipt required
MeteringReceiptReq()
MeteringReceiptRes()

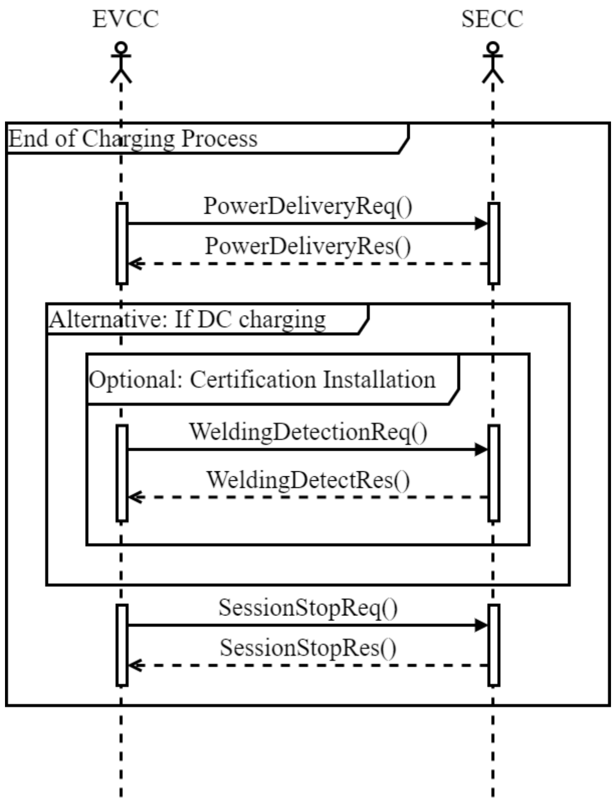
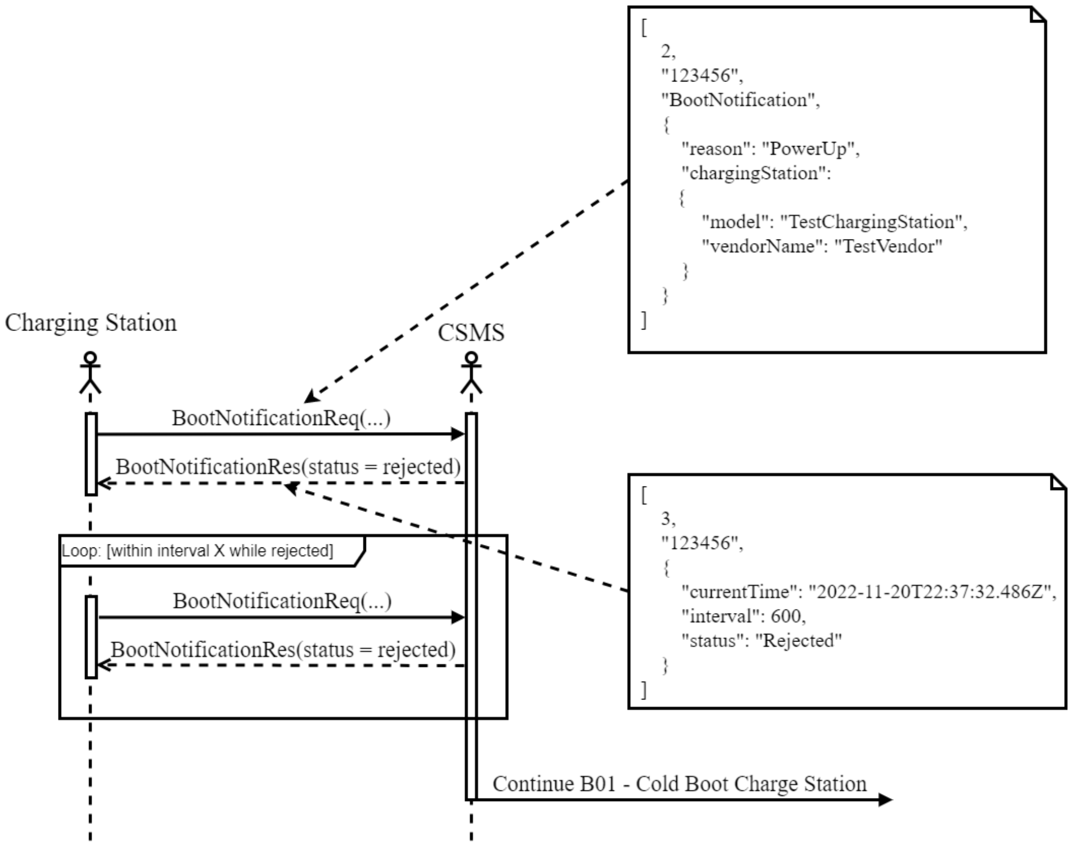


Figure 32: Sequence diagram of a ISO 15118 charging session

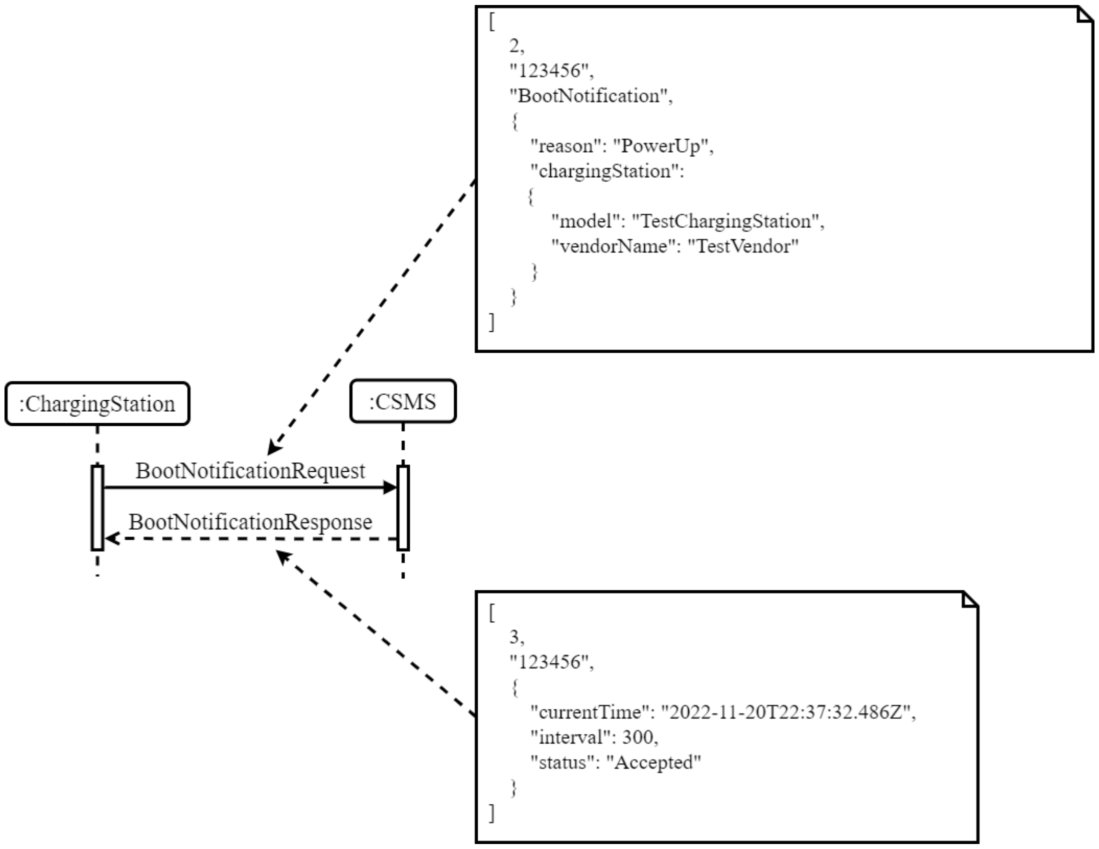
Appendix B - OCPP 2.0.1


Charging Station
CSMS
BootNotificationReq(...)
BootNotificationRes(status = rejected)
Loop: [within interval X while rejected]
BootNotificationReq(...)
BootNotificationRes(status = rejected)
Continue B01 - Cold Boot Charge Station

```
[
   2,
   "123456",
   "BootNotification",
   {
      "reason": "PowerUp",
      "chargingStation":
      {
         "model": "TestChargingStation",
         "vendorName": "TestVendor"
      }
   }
]
```

```
[
   3,
   "123456",
   {
      "currentTime": "2022-11-20T22:37:32.486Z",
      "interval": 600,
      "status": "Rejected"
   }
]
```

Figure 33: Boot Charge Station - Rejected



```
[
  2,
  "123456",
  "BootNotification",
  {
    "reason": "PowerUp",
    "chargingStation":
    {
      "model": "TestChargingStation",
      "vendorName": "TestVendor"
    }
  }
]
```

```
[
  3,
  "123456",
  {
    "currentTime": "2022-11-20T22:37:32.486Z",
    "interval": 300,
    "status": "Accepted"
  }
]
```

Figure 34: Boot Notification message sequence

Listing 1: Example of OCPP error message

```
[
    4,
    "162376037",
    "NotSupported",
    "SetDisplayMessageRequest not implemented",
    {}
]
```

Table 12: Use cases involving the StatusNotificationRequest

Index	Use case name
1	B01 - Cold Boot Charging Station
2	B04 - Offline Behavior Idle Charging Station
3	B12 - Reset - With Ongoing Transaction
4	C02 - Authorization using a start button
5	C03 - Authorization using credit/debit card
6	C05 - Authorization for CSMS initiated transactions
7	C06 - Authorization using local id type
8	C12 - Start Transaction - Cached Id
9	E02 - Start Transaction - Cable Plugin First
10	E03 - Start Transaction - IdToken First
11	E06 - Stop Transaction options
12	E07 - Transaction locally stopped by IdToken
13	E09 - When cable disconnected on EV-side: Stop Transaction
14	E10 - When cable disconnected on EV-side: Suspend Transaction
15	F01 - Remote Start Transaction - Cable Plugin First
16	F02 - Remote Start Transaction - Remote Start First
17	F03 - Remote Stop Transaction
18	G01 - Status Notification
19	G03 - Change Availability EVSE/Connector
20	G04 - Change Availability Charging Station
21	H01 - Reservation
22	H02 - Cancel Reservation
23	H03 - Use a reserved EVSE
24	H04 - Reservation Ended, not used
25	K02 - Central Smart Charging
26	K05 - Remote Start Transaction with Charging Profile

Appendix C - The interactions between ISO 15118 and OCPP 2.0.1

Table 13: Exchanged elements between Get15118EvCertificateRequest and CertificateInstallationRequest

	ISO15118 CertificateInstallationRequest	OCPP Get15118EvCertificateRequest
Element	CertificateInstallationRequest: base64Binary	ExiRequest:string

Table 14: Exchanged elements between Get15118EvCertificateResponse and CertificateInstallationResponse

	ISO15118 CertificateInstallationResponse	OCPP Get15118EvCertificateResponse
Element	CertificateInstallationResponse: base64Binary	ExiResponse:string

Table 15: Exchanged elements between PaymentDetailRequest and AuthorizeRequest

	ISO15118 PaymentDetailRequest	OCPP AuthorizeRequest
Element	eMAID:string Certificate:base64Binary	IdToken.IdToken:string Certificate:string

Table 16: Exchanged elements between PaymentDetailResponse and AuthorizeResponse

	ISO15118 PaymentDetailResponse	OCPP AuthorizeResponse
Element	ResponseCode:iso_enum	CertificateStatus:ocpp_enum

Table 17: Exchanged elements between ChargeParameterRequest and NotifyEVChargingNeedsRequest

	ISO15118 ChargeParameterRequest	OCPP NotifyEVChargingNeedsRequest
Common element	EnergyTransferMode:iso_enum	requestedEnergyTransfer:ocpp_enum
AC elements	EVChargeParameter. EAmount:PhysicalValueType	ChargingNeeds.AcChargingParameters. energyAmount:integer
	ChargingNeeds. EVMinCurrent:PhysicalValueType	ChargingNeeds.AcChargingParameters. EvMinCurrent:integer
	ChargingNeeds. EVMaxCurrent:PhysicalValueType	ChargingNeeds.AcChargingParameters. EvMaxCurrent:integer
	ChargingNeeds. EVMaxVoltage:PhysicalValueType	ChargingNeeds.AcChargingParameters. EvMaxVoltage:integer
DC elements	EVChargeParameter. EVMaximumCurrentLimit:PhysicalValueType	ChargingNeeds.DcChargingParameters. EvMaxCurrent:integer
	EVChargeParameter. EVMaximumVoltageLimit:PhysicalValueTypee	ChargingNeeds.DcChargingParameters. EvMaxVoltage:integer
	EVChargeParameter. EVEnergyRequest:PhysicalValueType	ChargingNeeds.DcChargingParameters. EnergyAmount:integer
	EVChargeParameter. EVMaximumPowerLimit:PhysicalValueType	ChargingNeeds.DcChargingParameters. EvMaxPower:integer
	EVChargeParameter. DCEVStatus.EVRESSSOC:byte	ChargingNeeds.DcChargingParameters. StateOfCharge:integer
	EVChargeParameter. FullSOC:byte	ChargingNeeds.DcChargingParameters. FullSoC:integer
	EVChargeParameter. BulkSOC:byte	ChargingNeeds.DcChargingParameters. BulkSoC:integer

Table 18: Exchanged elements between ChargeParameterReponse and SetChargingProfileRequest

	ISO15118 ChargeParameterReponse	OCPP SetChargingProfileRequest
Elements	SASchedules.SAScheduleTupleID:unsignedByte	ChargingProfile.ChargingSchedule.Id:int
	SASchedules.PMaxSchedule. PMaxScheduleEntry.RelativeTimeInterval.start:unsignedInt	ChargingProfile.chargingSchedule. ChargingSchedulePeriod.StartPeriod:int
	SASchedules.PMaxSchedule. PMaxScheduleEntry.PMax:PhysicalValueType	ChargingProfile.chargingSchedule. ChargingSchedulePeriod.limit:number
	SASchedules.SalesTariff. SalesTariffID:unsignedByte	ChargingProfile.ChargingSchedule. SalesTariff.Id:int
	SASchedules.SalesTariff. SalesTariffDescription:string	ChargingProfile.ChargingSchedule. SalesTariff.SalesTariffDescription:string
	SASchedules.SalesTariff. NumEPriceLevels:unsignedByte	ChargingProfile.ChargingSchedule. SalesTariff.NumEPriceLevels.Id:int
	SASchedules. SalesTariff.SalesTariffEntry. RelativeTimeInterval.Start:unsignedInt	ChargingProfile.ChargingSchedule. SalesTariff.SalesTariffEntry. RelativeTimeInterval.Start:int
	SASchedules. SalesTariff.SalesTariffEntry. RelativeTimeInterval.Duration:unsignedInt	ChargingProfile.ChargingSchedule. SalesTariff.SalesTariffEntry. RelativeTimeInterval.Duration:integer
	SASchedules. SalesTariff.SalesTariffEntry. EPriceLevel:unsignedByte	ChargingProfile.ChargingSchedule. SalesTariff.SalesTariffEntry. EPriceLevel:integer
	SASchedules. SalesTariff.SalesTariffEntry. ConsumptionCost.StartValue:PhysicalValueType	ChargingProfile.ChargingSchedule. SalesTariff.SalesTariffEntry. ConsumptionCost.StartValue:number
	SASchedules. SalesTariff.SalesTariffEntry. ConsumptionCost.Cost.Amount:unsignedInt	ChargingProfile.ChargingSchedule. SalesTariff.SalesTariffEntry. ConsumptionCost.Cost.Amount:integer

	ISO15118 ChargeParameterReponse	OCPP SetChargingProfileRequest
Elements	SASchedules. SalesTariff.SalesTariffEntry. ConsumptionCost.Cost.AmountMultiplier:byte	ChargingProfile.ChargingSchedule. SalesTariff.SalesTariffEntry. ConsumptionCost.Cost.AmountMultiplier:integer
	SASchedules. SalesTariff.SalesTariffEntry. Cost.CostKind:iso_enum	ChargingProfile.ChargingSchedule. SalesTariff.SalesTariffEntry. ConsumptionCost.Cost.CostKind:ocpp_enum

Table 19: Exchanged elements between PowerDeliveryRequest and NotifyEVChargingsSchedule

	ISO15118 PowerDeliveryRequest	OCPP NotifyEVChargingsSchedule
Element	SAScheduleTupleID:unsignedByte	EChargingSchedule.Id:int

Table 20: Exchanged elements between MeteringReceiptRequest and TransactionEventRequest

	ISO15118 MeteringReceiptRequest	OCPP TransactionEventRequest
Element	MeterInfo. MeterReading:unsignedLong	meterValue.sampledValue. value:number
	MeterInfo. MeterReading:base64Binary	meterValue.sampledValue. signedMeterValue. signedMeterData:string

Table 21: Exchanged elements between SessionStopRequest and TransactionEventRequest

	ISO15118 SessionStopRequest	OCPP TransactionEventRequest
Element	TransactionEvent:enum	ChargingSession:enum

Appendix D - Compatibility Calculation

For $k = 1$

The Calculation of $\mathbf{COMP^1_{UR,\leftrightarrow}[a_1, b_0]}$, $\mathbf{COMP^1_{UR,\leftrightarrow}[a_1, b_2]}$, $\mathbf{COMP^1_{UR,\leftrightarrow}[a_1, b_3]}$, $\mathbf{COMP^1_{UR,\leftrightarrow}[a_1, b_4]}$, and $\mathbf{COMP^1_{UR,\leftrightarrow}[a_1, b_5]}$

The state (a_1, b_0), (a_1, b_2), (a_1, b_3) ,(a_1, b_4), and (a_1, b_5) have dead lock because their message is not compatible.

$$\begin{aligned}
\Rightarrow obs\_comp^1_{UR,\leftrightarrow}(a_1, b_0) &= obs\_comp^1_{UR,\leftrightarrow}(a_1, b_2) \\
&= obs\_comp^1_{UR,\leftrightarrow}(a_1, b_3) \\
&= obs\_comp^1_{UR,\leftrightarrow}(a_1, b_4) \\
&= obs\_comp^1_{UR,\leftrightarrow}(a_1, b_5) \\
&= 0
\end{aligned}$$

$$\begin{aligned}
\Rightarrow fw\_propag^1_{UR,\leftrightarrow}(a_1, b_0) &= fw\_propag^1_{UR,\leftrightarrow}(a_1, b_2) \\
&= fw\_propag^1_{UR,\leftrightarrow}(a_1, b_3) \\
&= fw\_propag^1_{UR,\leftrightarrow}(a_1, b_4) \\
&= fw\_propag^1_{UR,\leftrightarrow}(a_1, b_5) \\
&= bw\_propag^1_{UR,\leftrightarrow}(a_1, b_0) \\
&= bw\_propag^1_{UR,\leftrightarrow}(a_1, b_2) \\
&= bw\_propag^1_{UR,\leftrightarrow}(a_1, b_3) \\
&= bw\_propag^1_{UR,\leftrightarrow}(a_1, b_4) \\
&= bw\_propag^1_{UR,\leftrightarrow}(a_1, b_5) \\
&= 0
\end{aligned}$$

The state compatibility is calculated as follows:

$$\begin{aligned}
state\_comp^{1}_{UR,\leftrightarrow}(a_1,b_0) &= \frac{w_3 * nat(a_1,b_0)}{w_1+w_2+w_3} = \frac{1*0}{5+5+1} = 0\\
state\_comp^{1}_{UR,\leftrightarrow}(a_1,b_2) &= \frac{w_3 * nat(a_1,b_2)}{w_1+w_2+w_3} = \frac{1*1}{2+2+1} = 0.2\\
state\_comp^{1}_{UR,\leftrightarrow}(a_1,b_3) &= \frac{w_3 * nat(a_1,b_3)}{w_1+w_2+w_3} = \frac{1*1}{2+3+1} = 0.167\\
state\_comp^{1}_{UR,\leftrightarrow}(a_1,b_4) &= \frac{w_3 * nat(a_1,b_4)}{w_1+w_2+w_3} = \frac{1*1}{5+2+1} = 0.125\\
state\_comp^{1}_{UR,\leftrightarrow}(a_1,b_5) &= \frac{w_3 * nat(a_1,b_4)}{w_1+w_2+w_3} = \frac{1*0}{1+3+1} = 0
\end{aligned}$$

and

$$\begin{aligned}
COMP^{1}_{UR,\leftrightarrow}[a_1,b_0] &= \frac{COMP^{0}_{UR,\leftrightarrow}[a_1,b_0] + state\_comp^{1}_{UR,\leftrightarrow}(a_1,b_0)}{2} = \frac{1+0}{2} = 0.5\\
COMP^{1}_{UR,\leftrightarrow}[a_1,b_2] &= \frac{COMP^{0}_{UR,\leftrightarrow}[a_1,b_2] + state\_comp^{1}_{UR,\leftrightarrow}(a_1,b_2)}{2} = \frac{1+0.2}{2} = 0.6\\
COMP^{1}_{UR,\leftrightarrow}[a_1,b_3] &= \frac{COMP^{0}_{UR,\leftrightarrow}[a_1,b_3] + state\_comp^{1}_{UR,\leftrightarrow}(a_1,b_3)}{2} = \frac{1+0.167}{2} = 0.584\\
COMP^{1}_{UR,\leftrightarrow}[a_1,b_4] &= \frac{COMP^{0}_{UR,\leftrightarrow}[a_1,b_4] + state\_comp^{1}_{UR,\leftrightarrow}(a_1,b_4)}{2} = \frac{1+0.125}{2} = 0.563\\
COMP^{1}_{UR,\leftrightarrow}[a_1,b_5] &= \frac{COMP^{0}_{UR,\leftrightarrow}[a_1,b_5] + state\_comp^{1}_{UR,\leftrightarrow}(a_1,b_5)}{2} = \frac{1+0.2}{2} = 0.5
\end{aligned}$$

The Calculation of $\mathbf{COMP^{1}_{UR,\leftrightarrow}[a_1,b_1]}$

$$obs\_comp^{1}_{UR,\leftrightarrow}(a_1,b_1) = \frac{sum^{1}_{UR,\leftrightarrow}((a_1,b_1),E(a_1,T_1),R(b_1,T_1)) + sum^{1}_{UR,\leftrightarrow}((b_1,a_1),E(b_1,T_1),R(a_1,T_1))}{\|E(a_1,T_1)\|+\|E(b_1,T_1)\|}$$

where:

$$\begin{aligned}
&E(a_1,T_1) = \varnothing\\
&R(a_1,T_1) = \{(a_1, certResponse?pl_{certResponse?}, a_2)\}\\
&tau(a_1,T_1) = \varnothing\\
&Fw(a_1,T_1) = \{(a_1, certResponse?pl_{certResponse?}, a_2)\}
\end{aligned}$$

and

$$\begin{aligned}
&E(b_1, T_1) = \{(b_1, certResponse!pl_{certResponse!}, b_0)\}\\
&R(b_1, T_1) = \varnothing\\
&tau(b_1, T_1) = \varnothing\\
&Fw(b_1, T_1) = \{(b_1, certResponse!pl_{certResponse!}, b_0)\}
\end{aligned}$$

The function $sum^1_{UR,\leftrightarrow}((a_1, b_1), E(a_1, T_1), R(b_1, T_1))$ calculates the sum of the best compatibility between $E(a_1, T_1)$ and $R(b_1, T_1)$. Because, $E(a_1, T_1) = \varnothing$

$$sum^1_{UR,\leftrightarrow}((a_1, b_1), E(a_1, T_1), R(b_1, T_1)) = 0$$

The function $sum^1_{UR,\leftrightarrow}((b_1, a_1), E(b_1, T_1), R(a_1, T_1))$ calculates the sum of the best compatibility between $E(b_1, T_1)$ and $R(a_1, T_1)$.

$$\begin{aligned}
\Rightarrow &sum^1_{UR,\leftrightarrow}((b_1, a_1), E(b_1, T_1), R(a_1, T_1))\\
&= lab\_comp(certResponse!, certResponse?) * COMP^0_{UR,\leftrightarrow}[b_0, a_2]
\end{aligned}$$

with

$$COMP^0_{UR,\leftrightarrow}[b_0, a_2] = 1$$

$$\begin{aligned}
&lab\_comp(certResponse!, certResponse?)\\
&= 1 - \frac{\|unsharedTypes(pl_{certResponse!}, pl_{certResponse?})\|)}{6(\|pl_{certResponse!}\| + \|pl_{certResponse?}\|)}
\end{aligned}$$

From Table 14, The data type between two messages are base64encoding and string respectively.

$$\Rightarrow \|unsharedTypes(pl_{certResponse!}, pl_{certResponse?})\| = 2$$

$$\begin{aligned}
\Rightarrow &lab\_comp(certResponse!, certResponse?)\\
&= 1 - \frac{2}{6(1+1)}\\
&= 0.833
\end{aligned} \tag{10}$$

$$\Rightarrow sum^1_{UR,\leftrightarrow}((b_1, a_1), E(b_1, T_1), R(a_1, T_1)) = 0.833$$
$$\Rightarrow obs\_comp^1_{UR,\leftrightarrow}(a_1, b_1) = \frac{0 + 0.833}{0 + 1} = 0.833$$

(Both state (a_1, b_1) do not have any incoming or outgoing τ transition.)

$$\begin{aligned}\Rightarrow & fw\_propag^1_{UR,\leftrightarrow}(a_1, b_1) \\ &= bw\_propag^1_{UR,\leftrightarrow}(a_1, b_1) \\ &= obs\_comp^1_{UR,\leftrightarrow}(a_1, b_1) = 0.833\end{aligned}$$

The state compatibility of state (a_1, b_1) is calculated as follows:

$$\begin{aligned}& state\_comp^1_{UR,\leftrightarrow}(a_1, b_1) \\ &= \frac{w_1 * fw\_propag^1_{UR,\leftrightarrow}(a_1, b_1) + w_2 * bw\_propag^1_{UR,\leftrightarrow}(a_1, b_1) + w_3 * nat(a_1, b_1)}{w_1 + w_2 + w_3}\end{aligned}$$

with:

- $w_1 = 2$ because a_1 has one outgoing transmission and b_1 has one outgoing transmission.
- $w_2 = 2$ because a_1 has one incoming transmission and b_1 has one incoming transmission.
- $w_3 = 1$ and $nat(a_1, b_1) = 1$ because $a_1 \notin (I_a \cup F_a)$ & $b_1 \notin (I_b \cup F_b)$

$$\Rightarrow state\_comp^1_{UR,\leftrightarrow}(a_1, b_1) = \frac{2 * 0.833 + 2 * 0.833 + 1 * 1}{2 + 2 + 1} = 0.866$$

And the compatibility degree is calculated as follows:

$$COMP^1_{UR,\leftrightarrow}[a_1, b_1] = \frac{COMP^0_{UR,\leftrightarrow}[a_1, b_1] + state\_comp^1_{UR,\leftrightarrow}(a_1, b_1)}{2} = \frac{1 + 0.866}{2} = 0.933$$

The Calculation of $\mathbf{COMP^1_{UR,\leftrightarrow}[a_2, b_0]}$

$$obs\_comp^1_{UR,\leftrightarrow}(a_2, b_0) = \frac{sum^1_{UR,\leftrightarrow}((a_2,b_0),E(a_2,T_2),R(b_0,T_0))+sum^1_{UR,\leftrightarrow}((b_0,a_2),E(b_0,T_0),R(a_2,T_2))}{\|E(a_2,T_2)\|+\|E(b_0,T_0)\|}$$

Where:

$$\begin{aligned}
&E(a_2, T_2) = \{(a_2, authorize!pl_{authorize!}, a_3)\}\\
&R(a_2, T_2) = \varnothing\\
&tau(a_2, T_2) = \varnothing\\
&Fw(a_2, T_2) = \{(a_2, authorize!pl_{authorize!}, a_3)\}
\end{aligned}$$

and

$$\begin{aligned}
R(b_0, T_0) = \{&(b_0, certReqest?pl_{certRequest?}, b_1),\\
&(b_0, authorize?pl_{authorize?}, b_2),\\
&(b_0, chargeParams?pl_{chargeParams?}, b_3),\\
&(b_0, transactionEvent?pl_{transactionEvent?}, b_4)\}\\
E(b_0, T_0) =&\varnothing\\
tau(b_0, T_0) =&\varnothing\\
Fw(b_0, T_0) =&R(b_0, T_0)
\end{aligned}$$

The function $sum^1_{UR,\leftrightarrow}((a_2, b_0), E(a_2, T_2), R(b_0, T_0))$ calculates the sum of the best compatibility between $E(a_2, T_2)$ and $R(b_0, T_0)$.

$$\begin{aligned}
&sum^1_{UR,\leftrightarrow}((a_2, b_2), E(a_2, T_2), R(b_0, T_0))\\
&= lab\_comp(authorize!, authorize?) * COMP^0_{UR,\leftrightarrow}[a_3, b_2]
\end{aligned}$$

Where:

$$lab\_comp(authorize!, authorize?) = 1 - \frac{\|unsharedTypes(pl_{authorize!}, pl_{authorize?})\|)}{6(\|pl_{authorize!}\| + \|pl_{authorize?}\|)}$$

Table 15 illustrates that both message have one uncommon data - base64Binary

$$\Rightarrow lab\_comp(authorize!, authorize?) = 1 - \frac{1}{6 * (2 + 2)} = 0.958$$

$$\Rightarrow sum^1_{UR,\leftrightarrow}((a_2, b_2), E(a_2, T_2), R(b_0, T_0)) = 0.958$$

The function $sum^1_{UR,\leftrightarrow}((b_0, a_0), E(b_0, T_0), R(a_2, T_2))$ is based upon the compatibility between the $E(b_0, T_0)$ and $R(a_2, T_2)$. However, $E(b_0, T_0) = \varnothing$,

$$\Rightarrow sum^1_{UR,\leftrightarrow}((b_0, a_2), E(b_0, T_0), R(a_2, T_2)) = 0$$
$$\Rightarrow obs\_comp^1_{UR,\leftrightarrow}(a_2, b_0) = \frac{0.958 + 0}{1} = 0.958$$

$$fw\_propag^1_{UR,\leftrightarrow}(a_2, b_0) = \frac{d\_fw\_propag^1_{UR,\leftrightarrow}(a_2, b_0) + d\_fw\_propag^1_{UR,\leftrightarrow}(b_0, a_2)}{2}$$

with:

$$d\_fw\_propag^1_{UR,\leftrightarrow}(a_2, b_0) = \frac{\sum_{(a_2,\tau,a'_2)\in T_0} fw\_propag^1_{CN,\leftrightarrow}(a'_2, b_0) + obs\_comp^1_{CN,\leftrightarrow}(a_2, b_2)}{\|tau(a_2, T2)\| + 1}$$
$$= obs\_comp^1_{CN,\leftrightarrow}(a_2, b_0)$$

Both state (a_2, b_0) do not have any incoming or outgoing τ transition.

$$\Rightarrow fw\_propag^1_{UR,\leftrightarrow}(a_2, b_0)$$
$$= bw\_propag^1_{UR,\leftrightarrow}(a_2, b_0)$$
$$= obs\_comp^1_{UR,\leftrightarrow}(a_2, b_0) = 0.958$$

The state compatibility of state (a_2, b_0) is calculated as follows:

$$state\_comp^1_{UR,\leftrightarrow}(a_2, b_0)$$
$$= \frac{w_1 * fw\_propag^1_{UR,\leftrightarrow}(a_2, b_0) + w_2 * bw\_propag^1_{UR,\leftrightarrow}(a_2, b_0) + w_3 * nat(a_2, b_0)}{w_1 + w_2 + w_3}$$

with $w_1 = 5$, $w_2 = 5$, and $w_3 = 1$ and $nat(a_2, b_0) = 0$ because $a_2 \notin (I_a \cup F_a)$ & $b_0 \in I_b$

$$\Rightarrow state\_comp^1_{UR,\leftrightarrow}(a_2, b_0) = \frac{5 * 0.958 + 5 * 0.958 + 0.958 * 0}{5 + 5 + 1} = 0.871$$

And the compatibility degree is calculated as follows:

$$COMP^1_{UR,\leftrightarrow}[a_2, b_0] = \frac{COMP^0_{UR,\leftrightarrow}[a_2, b_0] + state\_comp^1_{UR,\leftrightarrow}(a_2, b_0)}{2} = \frac{1 + 0.871}{2} = 0.936$$

The Calculation of $\mathbf{COMP^1_{UR,\leftrightarrow}[a_2, b_1]}$, $\mathbf{COMP^1_{UR,\leftrightarrow}[a_2, b_2]}$, $\mathbf{COMP^1_{UR,\leftrightarrow}[a_2, b_3]}$, $\mathbf{COMP^1_{UR,\leftrightarrow}[a_2, b_4]}$, and $\mathbf{COMP^1_{UR,\leftrightarrow}[a_2, b_5]}$

The state (a_2, b_1), (a_2, b_2), (a_2, b_3) ,(a_2, b_4), and (a_2, b_5) have dead lock because their message is not compatible.

$$\begin{aligned}
\Rightarrow obs\_comp^1_{UR,\leftrightarrow}(a_2, b_1) &= obs\_comp^1_{UR,\leftrightarrow}(a_2, b_2)\\
&= obs\_comp^1_{UR,\leftrightarrow}(a_2, b_3)\\
&= obs\_comp^1_{UR,\leftrightarrow}(a_2, b_4)\\
&= obs\_comp^1_{UR,\leftrightarrow}(a_2, b_5)\\
&= 0
\end{aligned}$$

$$\begin{aligned}
\Rightarrow fw\_propag^1_{UR,\leftrightarrow}(a_2, b_1) &= fw\_propag^1_{UR,\leftrightarrow}(a_2, b_2)\\
&= fw\_propag^1_{UR,\leftrightarrow}(a_2, b_3)\\
&= fw\_propag^1_{UR,\leftrightarrow}(a_2, b_4)\\
&= fw\_propag^1_{UR,\leftrightarrow}(a_2, b_5)\\
&= bw\_propag^1_{UR,\leftrightarrow}(a_2, b_1)\\
&= bw\_propag^1_{UR,\leftrightarrow}(a_2, b_2)\\
&= bw\_propag^1_{UR,\leftrightarrow}(a_2, b_3)\\
&= bw\_propag^1_{UR,\leftrightarrow}(a_2, b_4)\\
&= bw\_propag^1_{UR,\leftrightarrow}(a_2, b_5)\\
&= 0
\end{aligned}$$

The state compatibility is calculated as follows

$$\begin{aligned}
state\_comp^1_{UR,\leftrightarrow}(a_2, b_1) &= \frac{w_3 * nat(a_2, b_1)}{w_1 + w_2 + w_3} = \frac{1 * 1}{2 + 2 + 1} = 0.2\\
state\_comp^1_{UR,\leftrightarrow}(a_2, b_2) &= \frac{w_3 * nat(a_2, b_2)}{w_1 + w_2 + w_3} = \frac{1 * 1}{2 + 2 + 1} = 0.2\\
state\_comp^1_{UR,\leftrightarrow}(a_2, b_3) &= \frac{w_3 * nat(a_2, b_3)}{w_1 + w_2 + w_3} = \frac{1 * 1}{2 + 3 + 1} = 0.167\\
state\_comp^1_{UR,\leftrightarrow}(a_2, b_4) &= \frac{w_3 * nat(a_2, b_4)}{w_1 + w_2 + w_3} = \frac{1 * 1}{5 + 2 + 1} = 0.125\\
state\_comp^1_{UR,\leftrightarrow}(a_2, b_5) &= \frac{w_3 * nat(a_2, b_5)}{w_1 + w_2 + w_3} = \frac{1 * 0}{1 + 3 + 1} = 0
\end{aligned}$$

and

$$COMP^1_{UR,\leftrightarrow}[a_2,b_1] = \frac{COMP^0_{UR,\leftrightarrow}[a_2,b_1] + state\_comp^1_{UR,\leftrightarrow}(a_2,b_1)}{2} = \frac{1+0.2}{2} = 0.6$$
$$COMP^1_{UR,\leftrightarrow}[a_2,b_2] = \frac{COMP^0_{UR,\leftrightarrow}[a_2,b_2] + state\_comp^1_{UR,\leftrightarrow}(a_2,b_2)}{2} = \frac{1+0.2}{2} = 0.6$$
$$COMP^1_{UR,\leftrightarrow}[a_3,b_3] = \frac{COMP^0_{UR,\leftrightarrow}[a_3,b_3] + state\_comp^1_{UR,\leftrightarrow}(a_3,b_3)}{2} = \frac{1+0.167}{2} = 0.584$$
$$COMP^1_{UR,\leftrightarrow}[a_4,b_4] = \frac{COMP^0_{UR,\leftrightarrow}[a_4,b_4] + state\_comp^1_{UR,\leftrightarrow}(a_4,b_4)}{2} = \frac{1+0.125}{2} = 0.563$$
$$COMP^1_{UR,\leftrightarrow}[a_5,b_5] = \frac{COMP^0_{UR,\leftrightarrow}[a_5,b_5] + state\_comp^1_{UR,\leftrightarrow}(a_5,b_5)}{2} = \frac{1+0}{2} = 0.5$$

The Calculation of $\mathbf{COMP^1_{UR,\leftrightarrow}[a_3,b_2]}$

$$obs\_comp^1_{UR,\leftrightarrow}(a_3,b_2) = \frac{sum^1_{UR,\leftrightarrow}((a_3,b_2),E(a_3,T_3),R(b_2,T_2))+sum^1_{UR,\leftrightarrow}((b_2,a_3),E(b_2,T_2),R(a_3,T_3))}{\|E(a_3,T_3)\|+\|E(b_2,T_2)\|}$$

where:

$$\begin{aligned}
&E(a_3,T_3) = \varnothing\\
&R(a_3,T_3) = \{(a_3, paymentDetail?pl_{paymentDetail?}, a_4)\}\\
&tau(a_3,T_3) = \varnothing\\
&Fw(a_3,T_3) = R(a_3,T_3)
\end{aligned}$$

and

$$\begin{aligned}
&E(b_2,T_2) = \{(b_2, paymentDetail!pl_{paymentDetail!}, b_0)\}\\
&R(b_2,T_2) = \varnothing\\
&tau(b_2,T_2) = \varnothing\\
&Fw(b_2,T_2) = E(b_2,T_2)
\end{aligned}$$

The function $sum^1_{UR,\leftrightarrow}((a_3,b_2),E(a_3,T_3),R(b_2,T_2))$ calculates the sum of the best compatibility between $E(a_3,T_3)$ and $R(b_2,T_2)$. Because, $E(a_3,T_3)=\varnothing$

$$sum^1_{UR,\leftrightarrow}((a_3,b_2),E(a_3,T_3),R(b_2,T_2)) = 0$$

The function $sum^1_{UR,\leftrightarrow}((b_2,a_3),E(b_2,T_2),R(a_3,T_3))$ calculates the sum of the best compatibility between $E(b_2,T_2)$ and $R(a_3,T_3)$.

$$\begin{aligned}&\Rightarrow sum^1_{UR,\leftrightarrow}((b_2,a_3),E(b_2,T_2),R(a_3,T_3))\\&= lab\_comp(paymentDetail!,paymentDetail?) * COMP^0_{UR,\leftrightarrow}[b_0,a_4]\end{aligned}$$

with

$$COMP^0_{UR,\leftrightarrow}[b_0,a_4] = 1$$

$$\begin{aligned}&lab\_comp(paymentDetail!,paymentDetail?)\\&= 1 - \frac{\|unsharedTypes(pl_{paymentDetail!},pl_{paymentDetail?})\|)}{6(\|pl_{paymentDetail!}\| + \|pl_{paymentDetail?}\|)}\end{aligned}$$

The list of parameters between message *paymentDetail*! and *paymentDetail*! is listed in Table 16. Because the parameters have different enumeration value, $\|unsharedTypes(pl_{certResponse!},pl_{certResponse?})\| = 2$.

$$\Rightarrow lab\_comp(certResponse!,certResponse?) = 1 - \frac{2}{6(1+1)} = 0.833$$

$$\begin{aligned}&\Rightarrow sum^1_{UR,\leftrightarrow}((b_2,a_3),E(b_2,T_2),R(a_3,T_3)) = 0.833\\&\Rightarrow obs\_comp^1_{UR,\leftrightarrow}(a_3,b_2) = \frac{0+0.833}{0+1} = 0.833\end{aligned}$$

Both state (a_3,b_2) do not have any incoming or outgoing τ transitions.

$$\begin{aligned}&\Rightarrow fw\_propag^1_{UR,\leftrightarrow}(a_3,b_2)\\&\quad= bw\_propag^1_{UR,\leftrightarrow}(a_3,b_2)\\&\quad= obs\_comp^1_{UR,\leftrightarrow}(a_3,b_2) = 0.833\end{aligned}$$

The state compatibility of state (a_3,b_2) is calculated as follows:

$$\begin{aligned}&state\_comp^1_{UR,\leftrightarrow}(a_3,b_2)\\&= \frac{w_1 * fw\_propag^1_{UR,\leftrightarrow}(a_3,b_2) + w_2 * bw\_propag^1_{UR,\leftrightarrow}(a_3,b_2) + w_3 * nat(a_3,b_2)}{w_1+w_2+w_3}\end{aligned}$$

with $w_1 = 2$, $w_2 = 2$, $w_3 = 1$ and $nat(a_3,b_2) = 1$ because $a_3 \notin (I_a \cup F_a)$ & $b_2 \notin (I_b \cup F_b)$

$$\begin{aligned}
&\Rightarrow state\_comp^1_{UR,\leftrightarrow}(a_3,b_2) = \frac{2*0.833+2*0.833+1*1}{2+2+1} = 0.866\\
&\Rightarrow COMP^1_{UR,\leftrightarrow}[a_3,b_2]\\
&\quad = \frac{COMP^0_{UR,\leftrightarrow}[a_3,b_2] + state\_comp^1_{UR,\leftrightarrow}(a_3,b_2)}{2}\\
&\quad = \frac{1+0.866}{2} = 0.933
\end{aligned}$$

The Calculation of $\mathbf{COMP^1_{UR,\leftrightarrow}[a_3,b_0]}$, $\mathbf{COMP^1_{UR,\leftrightarrow}[a_3,b_1]}$, $\mathbf{COMP^1_{UR,\leftrightarrow}[a_3,b_3]}$, $\mathbf{COMP^1_{UR,\leftrightarrow}[a_3,b_4]}$, and $\mathbf{COMP^1_{UR,\leftrightarrow}[a_2,b_5]}$

The observable compatibility *obs_comp* between states (a_3,b_0), (a_3,b_1), (a_3,b_3), (a_3,b_4), and (a_3,b_5) have value zero because their labels are not compatible. For example the emissions in a_3 and b_0 have the same direction (receiving).

$$\begin{aligned}
\Rightarrow obs\_comp^1_{UR,\leftrightarrow}(a_3,b_0) &= obs\_comp^1_{UR,\leftrightarrow}(a_3,b_1)\\
&= obs\_comp^1_{UR,\leftrightarrow}(a_3,b_3)\\
&= obs\_comp^1_{UR,\leftrightarrow}(a_3,b_4)\\
&= obs\_comp^1_{UR,\leftrightarrow}(a_3,b_5)\\
&= 0
\end{aligned}$$

Furthermore, in those states, there are no forward and backward τ transitions in those states.

$$\begin{aligned}
\Rightarrow fw\_propag^1_{CN,\leftrightarrow}(a_3,b_0) &= fw\_propag^1_{CN,\leftrightarrow}(a_3,b_1)\\
&= fw\_propag^1_{CN,\leftrightarrow}(a_3,b_3)\\
&= fw\_propag^1_{CN,\leftrightarrow}(a_3,b_4)\\
&= fw\_propag^1_{CN,\leftrightarrow}(a_3,b_5)\\
&= bw\_propag^1_{CN,\leftrightarrow}(a_3,b_0)\\
&= bw\_propag^1_{CN,\leftrightarrow}(a_3,b_1)\\
&= bw\_propag^1_{CN,\leftrightarrow}(a_3,b_3)\\
&= bw\_propag^1_{CN,\leftrightarrow}(a_3,b_4)\\
&= bw\_propag^1_{CN,\leftrightarrow}(a_3,b_5)\\
&= 0
\end{aligned}$$

The state compatibility is calculated as follows:

$$\begin{aligned}
state\_comp^{1}_{UR,\leftrightarrow}(a_3,b_0) &= \frac{w_3 * nat(a_3,b_0)}{w_1+w_2+w_3} = \frac{1*0}{5+5+1} = 0\\
state\_comp^{1}_{UR,\leftrightarrow}(a_3,b_1) &= \frac{w_3 * nat(a_3,b_1)}{w_1+w_2+w_3} = \frac{1*1}{2+2+1} = 0.2\\
state\_comp^{1}_{UR,\leftrightarrow}(a_3,b_3) &= \frac{w_3 * nat(a_3,b_3)}{w_1+w_2+w_3} = \frac{1*1}{2+3+1} = 0.167\\
state\_comp^{1}_{UR,\leftrightarrow}(a_3,b_4) &= \frac{w_3 * nat(a_3,b_4)}{w_1+w_2+w_3} = \frac{1*1}{5+2+1} = 0.125\\
state\_comp^{1}_{UR,\leftrightarrow}(a_3,b_5) &= \frac{w_3 * nat(a_3,b_5)}{w_1+w_2+w_3} = \frac{1*0}{1+3+1} = 0
\end{aligned}$$

and

$$\begin{aligned}
COMP^{1}_{UR,\leftrightarrow}[a_3,b_0] &= \frac{COMP^{0}_{UR,\leftrightarrow}[a_3,b_0] + state\_comp^{1}_{UR,\leftrightarrow}(a_3,b_0)}{2} = \frac{1+0}{2} = 0.5\\
COMP^{1}_{UR,\leftrightarrow}[a_3,b_1] &= \frac{COMP^{0}_{UR,\leftrightarrow}[a_3,b_1] + state\_comp^{1}_{UR,\leftrightarrow}(a_3,b_1)}{2} = \frac{1+0.2}{2} = 0.6\\
COMP^{1}_{UR,\leftrightarrow}[a_3,b_3] &= \frac{COMP^{0}_{UR,\leftrightarrow}[a_3,b_3] + state\_comp^{1}_{UR,\leftrightarrow}(a_3,b_3)}{2} = \frac{1+0.167}{2} = 0.584\\
COMP^{1}_{UR,\leftrightarrow}[a_3,b_4] &= \frac{COMP^{0}_{UR,\leftrightarrow}[a_3,b_4] + state\_comp^{1}_{UR,\leftrightarrow}(a_3,b_4)}{2} = \frac{1+0.125}{2} = 0.563\\
COMP^{1}_{UR,\leftrightarrow}[a_3,b_5] &= \frac{COMP^{0}_{UR,\leftrightarrow}[a_3,b_5] + state\_comp^{1}_{UR,\leftrightarrow}([a_3,b_5)}{2} = \frac{1+0}{2} = 0.5
\end{aligned}$$

The Calculation of $\mathbf{COMP^{1}_{UR,\leftrightarrow}[a_4,b_0]}$

$$obs\_comp^{1}_{UR,\leftrightarrow}(a_4,b_0) = \frac{sum^{1}_{UR,\leftrightarrow}((a_4,b_0),E(a_4,T_4),R(b_0,T_0)) + sum^{1}_{UR,\leftrightarrow}((b_0,a_4),E(b_0,T_0),R(a_4,T_4))}{\|E(a_4,T_4)\| + \|E(b_0,T_0)\|}$$

Where:

$$\begin{aligned}
E(a_4,T_4) &= \{(a_4, chargeParams!pl_{chargeParams!}, a_5)\}\\
R(a_4,T_4) &= \varnothing\\
tau(a_4,T_4) &= \varnothing\\
Fw(a_4,T_4) &= E(a_4,T_4)
\end{aligned}$$

and

$$\begin{aligned}
R(b_0,T_0) = \{&(b_0, certReqest?pl_{certRequest?}, b_1),\\
&(b_0, authorize?pl_{authorize?}, b_2),\\
&(b_0, chargeParams?pl_{chargeParams?}, b_3),\\
&(b_0, transactionEvent?pl_{transactionEvent?}, b_4)\}\\
E(b_0,T_0) =&\varnothing\\
tau(b_0,T_0) =&\varnothing\\
Fw(b_0,T_0) =&R(b_0,T_0)
\end{aligned}$$

The function $sum^1_{UR,\leftrightarrow}((a_4,b_0),E(a_4,T_4),R(b_0,T_0))$ calculates the sum of the best compatibility between $E(a_4,T_4)$ and $R(b_0,T_0)$.

$$\begin{aligned}
&sum^1_{UR,\leftrightarrow}((a_4,b_0),E(a_4,T_4),R(b_0,T_0))\\
&= lab\_comp(chargeParams!, chargeParams?) * COMP^0_{UR,\leftrightarrow}[a_5,b_3]
\end{aligned}$$

Where:

$$\begin{aligned}
&lab\_comp(chargeParams!, chargeParams?)\\
&= 1 - \frac{\|unsharedTypes(pl_{chargeParams!}, pl_{chargeParams?})\|)}{6(\|pl_{chargeParams!}\| + \|pl_{chargeParams?}\|)}
\end{aligned}$$

and

$$COMP^0_{UR,\leftrightarrow}[a_5,b_3] = 1$$

Table 17 illustrates the list of exchange parameters and their data types. The parameters from DC charging are used for this calculation.

$$unshared\_types = \{iso\_enum, ocpp\_enum, PhysicalValueType, integer, byte\}$$

$$\begin{aligned}
&\Rightarrow lab\_comp(chargeParams!, chargeParams?) \\
&\quad = 1 - \frac{\|unshared\_types\|)}{6(\|pl_{chargeParams!}\| + \|pl_{chargeParams?}\|)} \\
&\quad = 1 - \frac{5}{6(8+8)} \\
&\quad = 0.948
\end{aligned} \tag{11}$$

$$\Rightarrow sum^1_{UR,\leftrightarrow}((a_4, b_0), E(a_4, T_4), R(b_0, T_0)) = 0.948$$

The function $sum^1_{UR,\leftrightarrow}((b_0, a_4), E(b_0, T_0), R(a_4, T_4))$ is based upon the compatibility between the $E(b_0, T_0)$ and $R(a_4, T_4)$. However, $E(b_0, T_0) = \varnothing$,

$$\begin{aligned}
&\Rightarrow sum^1_{UR,\leftrightarrow}((b_0, a_4), E(b_0, T_0), R(a_4, T_4)) = 0 \\
&\Rightarrow obs\_comp^1_{UR,\leftrightarrow}(a_4, b_0) = \frac{0.948 + 0}{1} = 0.948
\end{aligned}$$

Both state (a_4, b_0) do not have any incoming or outgoing τ transitions.

$$\begin{aligned}
&\Rightarrow fw\_propag^1_{UR,\leftrightarrow}(a_4, b_0) \\
&\quad = bw\_propag^1_{UR,\leftrightarrow}(a_4, b_0) \\
&\quad = obs\_comp^1_{UR,\leftrightarrow}(a_4, b_0) = 0.948
\end{aligned}$$

The state compatibility of state (a_4, b_0) is calculated as follows:

$$\begin{aligned}
&state\_comp^1_{UR,\leftrightarrow}(a_4, b_0) \\
&= \frac{w_1 * fw\_propag^1_{UR,\leftrightarrow}(a_4, b_0) + w_2 * bw\_propag^1_{UR,\leftrightarrow}(a_4, b_0) + w_3 * nat(a_4, b_0)}{w_1 + w_2 + w_3}
\end{aligned}$$

with $w_1 = 5$, $w_2 = 5$, and $w_3 = 1$ and $nat(a_4, b_0) = 0$ because $a_4 \notin (I_a \cup F_a)$ & $b_0 \in I_b$

$$\Rightarrow state\_comp^1_{UR,\leftrightarrow}(a_2, b_0) = \frac{5 * 0.948 + 5 * 0.948 + 1 * 0}{5 + 5 + 1} = 0.862$$

And the compatibility degree is calculated as follows:

$$COMP^1_{UR,\leftrightarrow}[a_4, b_0] = \frac{COMP^0_{UR,\leftrightarrow}[a_4, b_0] + state\_comp^1_{UR,\leftrightarrow}(a_4, b_0)}{2} = \frac{1 + 0.862}{2} = 0.931$$

The Calculation of $\mathbf{COMP^1_{UR,\leftrightarrow}[a_4, b_4]}$

$$obs\_comp^1_{UR,\leftrightarrow}(a_4, b_4) = \frac{sum^1_{UR,\leftrightarrow}((a_4,b_4),E(a_4,T_4),R(b_4,T_4))+sum^1_{UR,\leftrightarrow}((b_4,a_4),E(b_4,T_4),R(a_4,T_4))}{\|E(a_4,T_4)\|+\|E(b_4,T_4)\|}$$

Where:

$$\begin{aligned}
&E(a_4, T_4) = \{(a_4, chargeParams!pl_{chargeParams!}, a_5)\}\\
&R(a_4, T_4) = \varnothing\\
&tau(a_4, T_4) = \varnothing\\
&Fw(a_4, T_4) = E(a_4, T_4)
\end{aligned}$$

and

$$\begin{aligned}
R(b_4, T_4) =&\{(b_4, chargeParams?pl_{chargeParams?}, b_3),\\
&(b_4, transactionEvent?pl_{transactionEvent?}, b_5)\}\\
E(b_4, T_4) =&\{(b_4, chargeSchedule!pl_{chargeSchedule!}, b_0),\\
&(b_4, requestStop!pl_{requestStop!}, b_5)\}\\
tau(b_4, T_4) =&\varnothing\\
Fw(b_4, T_4) =&E(b_4, T_4) \cup R(b_4, T_4)
\end{aligned}$$

The function $sum^1_{UR,\leftrightarrow}((a_4, b_4), E(a_4, T_4), R(b_4, T_4))$ calculates the sum of the best compatibility between $E(a_4, T_4)$ and $R(b_4, T_4)$. The calculation is reduced to: $(chargeParams!, chargeParams?)$ because $(chargeParams!, transactionEvent?)$ is not label compatible.

$$\begin{aligned}
&sum^1_{UR,\leftrightarrow}((a_4, b_4), E(a_4, T_4), R(b_4, T_4))\\
&= lab\_comp(chargeParams!, chargeParams?) * COMP^0_{UR,\leftrightarrow}[a_5, b_3]
\end{aligned}$$

Where:

$$\begin{aligned}
&lab\_comp(chargeParams!, chargeParams?)\\
&= 1 - \frac{\|unsharedTypes(pl_{chargeParams!}, pl_{chargeParams?})\|)}{6(\|pl_{chargeParams!}\| + \|pl_{chargeParams?}\|)}
\end{aligned}$$

and

$$COMP^0_{UR,\leftrightarrow}[a_5, b_3] = 1$$

From Equation 11, $lab\_comp(chargeParams!, chargeParams?) = 0.948$

$$\Rightarrow sum^1_{UR,\leftrightarrow}((a_4, b_4), E(a_4, T_4), R(b_4, T_4)) = 0.948$$

The function $sum^1_{UR,\leftrightarrow}((b_4, a_4), E(b_4, T_4), R(a_4, T_4))$ is based upon the compatibility between the $E(b_4, T_4)$ and $R(a_4, T_4)$. However, $E(a_4, T_4) = \varnothing$,

$$\begin{aligned}
&\Rightarrow sum^1_{UR,\leftrightarrow}((b_4, a_4), E(b_4, T_4), R(a_4, T_4)) = 0 \\
&\Rightarrow obs\_comp^1_{UR,\leftrightarrow}(a_4, b_4) = \frac{0.948 + 0}{1 + 2} = 0.316
\end{aligned}$$

Both state (a_4, b_4) do not have any incoming or outgoing τ transitions.

$$\begin{aligned}
\Rightarrow & fw\_propag^1_{UR,\leftrightarrow}(a_4, b_4) \\
&= bw\_propag^1_{UR,\leftrightarrow}(a_4, b_4) \\
&= obs\_comp^1_{UR,\leftrightarrow}(a_4, b_4) = 0.316
\end{aligned}$$

The state compatibility of state (a_4, b_4) is calculated as follows:

$$\begin{aligned}
&state\_comp^1_{UR,\leftrightarrow}(a_4, b_4) \\
&= \frac{w_1 * fw\_propag^1_{UR,\leftrightarrow}(a_4, b_4) + w_2 * bw\_propag^1_{UR,\leftrightarrow}(a_4, b_4) + w_3 * nat(a_4, b_4)}{w_1 + w_2 + w_3}
\end{aligned}$$

with $w_1 = 5$, $w_2 = 2$, and $w_3 = 1$ and $nat(a_4, b_4) = 1$ because $a_4 \notin (I_a \cup F_a)$ & $b_0 \notin (I_b \cup F_b)$

$$\Rightarrow state\_comp^1_{UR,\leftrightarrow}(a_4, b_4) = \frac{5 * 0.316 + 2 * 0.316 + 1 * 1}{5 + 2 + 1} = 0.402$$

And the compatibility degree is calculated as follows:

$$COMP^1_{UR,\leftrightarrow}[a_4, b_4] = \frac{COMP^0_{UR,\leftrightarrow}[a_4, b_4] + state\_comp^1_{UR,\leftrightarrow}(a_4, b_4)}{2} = \frac{1 + 0.402}{2} = 0.701$$

The Calculation of $\mathbf{COMP^1_{UR,\leftrightarrow}[a_4, b_1]}$, $\mathbf{COMP^1_{UR,\leftrightarrow}[a_4, b_2]}$, $\mathbf{COMP^1_{UR,\leftrightarrow}[a_4, b_3]}$, and $\mathbf{COMP^1_{UR,\leftrightarrow}[a_4, b_5]}$

By observing the Figure 18, the observable compatibility of the states (a_4, b_1), (a_4, b_2), (a_4, b_3), and (a_4, b_5) are equal to zero because of dead lock.

$$\begin{aligned}
\Rightarrow fw\_propag^1_{CN,\leftrightarrow}(a_4, b_1) &= fw\_propag^1_{CN,\leftrightarrow}(a_4, b_2)\\
&= fw\_propag^1_{CN,\leftrightarrow}(a_4, b_3)\\
&= fw\_propag^1_{CN,\leftrightarrow}(a_4, b_5)\\
&= bw\_propag^1_{CN,\leftrightarrow}(a_4, b_1)\\
&= bw\_propag^1_{CN,\leftrightarrow}(a_4, b_2)\\
&= bw\_propag^1_{CN,\leftrightarrow}(a_4, b_3)\\
&= bw\_propag^1_{CN,\leftrightarrow}(a_4, b_5)\\
&= 0
\end{aligned}$$

The state compatibility is calculated as follows:

$$\begin{aligned}
state\_comp^1_{UR,\leftrightarrow}(a_4, b_1) &= \frac{w_3 * nat(a_4, b_1)}{w_1 + w_2 + w_3} = \frac{1 * 1}{2 + 2 + 1} = 0.2\\
state\_comp^1_{UR,\leftrightarrow}(a_4, b_2) &= \frac{w_3 * nat(a_4, b_2)}{w_1 + w_2 + w_3} = \frac{1 * 1}{2 + 2 + 1} = 0.2\\
state\_comp^1_{UR,\leftrightarrow}(a_4, b_3) &= \frac{w_3 * nat(a_4, b_3)}{w_1 + w_2 + w_3} = \frac{1 * 1}{2 + 3 + 1} = 0.167\\
state\_comp^1_{UR,\leftrightarrow}(a_4, b_5) &= \frac{w_3 * nat(a_4, b_5)}{w_1 + w_2 + w_3} = \frac{1 * 0}{5 + 2 + 1} = 0
\end{aligned}$$

and

$$\begin{aligned}
COMP^1_{UR,\leftrightarrow}[a_4, b_1] &= \frac{COMP^0_{UR,\leftrightarrow}[a_4, b_1] + state\_comp^1_{UR,\leftrightarrow}(a_4, b_1)}{2} = \frac{1 + 0.2}{2} = 0.6\\
COMP^1_{UR,\leftrightarrow}[a_4, b_2] &= \frac{COMP^0_{UR,\leftrightarrow}[a_4, b_2] + state\_comp^1_{UR,\leftrightarrow}(a_4, b_2)}{2} = \frac{1 + 0.2}{2} = 0.6\\
COMP^1_{UR,\leftrightarrow}[a_4, b_3] &= \frac{COMP^0_{UR,\leftrightarrow}[a_4, b_3] + state\_comp^1_{UR,\leftrightarrow}(a_4, b_3)}{2} = \frac{1 + 0.167}{2} = 0.584\\
COMP^1_{UR,\leftrightarrow}[a_4, b_5] &= \frac{COMP^0_{UR,\leftrightarrow}[a_4, b_5] + state\_comp^1_{UR,\leftrightarrow}(a_4, b_5)}{2} = \frac{1 + 0}{2} = 0.5
\end{aligned}$$

The Calculation of $\mathbf{COMP^1_{UR,\leftrightarrow}[a_5, b_3]}$

$$obs\_comp^{1}_{UR,\leftrightarrow}(a_5, b_3) = \frac{sum^{1}_{UR,\leftrightarrow}((a_5,b_3),E(a_5,T_5),R(b_3,T_3))+sum^{1}_{UR,\leftrightarrow}((b_3,a_5),E(b_3,T_3),R(a_5,T_5))}{\|E(a_5,T_5)\|+\|E(b_3,T_3)\|}$$

Where:

$$\begin{aligned}
&E(a_5, T_5) = \varnothing\\
&R(a_5, T_5) = \{(a_4, chargeSchedule?pl_{chargeSchedule?}, a_6)\}\\
&tau(a_5, T_5) = \varnothing\\
&Fw(a_5, T_5) = E(a_5, T_5)
\end{aligned}$$

and

$$\begin{aligned}
&R(b_3, T_3) = \varnothing\\
&E(b_3, T_3) = \{(b_3, chargeSchedule!pl_{chargeSchedule!}, b_0)\}\\
&tau(b_3, T_3) = \varnothing\\
&Fw(b_3, T_3) = E(b_3, T_3)
\end{aligned}$$

The function $sum^{1}_{UR,\leftrightarrow}((a_5, b_3), E(a_5, T_5), R(b_3, T_3))$ calculates the sum of the best compatibility between $E(a_5, T_5)$ and $R(b_3, T_3)$. Since $E(a_5, T_5) = \varnothing$,

$$sum^{1}_{UR,\leftrightarrow}((a_5, b_3), E(a_5, T_5), R(b_3, T_3)) = 0$$

The function $sum^{1}_{UR,\leftrightarrow}((b_3, a_5), E(b_3, T_3), R(a_5, T_5))$ is based upon the compatibility between the $E(b_3, T_3)$ and $R(a_5, T_5)$.

$$\begin{aligned}
&sum^{1}_{UR,\leftrightarrow}((b_3, a_5), E(b_3, T_3), R(a_5, T_5))\\
&= lab\_comp(chargeSchedule!, chargeSchedule?) * COMP^{0}_{UR,\leftrightarrow}[b_0, a_6]
\end{aligned}$$

Where:

$$\begin{aligned}
&lab\_comp(chargeParams!, chargeParams?)\\
&= 1 - \frac{\|chargeSchedule(pl_{chargeSchedule!}, pl_{chargeSchedule?})\|)}{6(\|pl_{chargeSchedule!}\| + \|pl_{chargeSchedule?}\|)}
\end{aligned}$$

and

$$COMP^0_{UR,\leftrightarrow}[b_0, a_6] = 1$$

From Table 18, the list of of un-shared data types are

$$\begin{aligned} unshared\_types = \{&PhysicalValueType \\ &number \\ &unsignedByte, \\ &unsignedInteger, \\ &integer, \\ &byte, \\ &iso\_enum, \\ &ocpp\_enum)\} \end{aligned}$$

$$\begin{aligned} \Rightarrow &lab\_comp(chargeSchedule!, chargeSchedule?) \\ &= 1 - \frac{\|unshared\_types\|)}{6(\|pl_{chargeSchedule!}\| + \|pl_{chargeSchedule?}\|)} \\ &= 1 - \frac{8}{6(13+13)} \\ &= 0.949 \end{aligned} \tag{12}$$

The state (a_5, b_3) does not have any outgoing or incoming τ transitions

$$\begin{aligned} \Rightarrow &fw\_propag^k_{UR,\leftrightarrow}(a_5, b_3) \\ &= bw\_propag^1_{UR,\leftrightarrow}(a_5, b_3) \\ &= lab\_comp(chargeSchedule!, chargeSchedule?) = 0.949 \end{aligned}$$

The state compatibility of state (a_5, b_3) is calculated as follows:

$$\begin{aligned} &state\_comp^1_{UR,\leftrightarrow}(a_5, b_3) \\ &= \frac{w_1 * fw\_propag^1_{UR,\leftrightarrow}(a_5, b_3) + w_2 * bw\_propag^1_{UR,\leftrightarrow}(a_5, b_3) + w_3 * nat(a_5, b_3)}{w_1 + w_2 + w_3} \end{aligned}$$

with $w_1 = 2$, $w_2 = 4$, and $w_3 = 1$ and $nat(a_5, b_3) = 1$ because $a_5 \notin (I_a \cup F_a)$ & $b_3 \notin (I_b \cup F_b)$

$$\Rightarrow state\_comp^1_{UR,\leftrightarrow}(a_5, b_3) = \frac{2*0.949 + 4*0.949 + 1*1}{2+4+1} = 0.956$$

And the compatibility degree is calculated as follows:

$$COMP^1_{UR,\leftrightarrow}[a_5, b_3] = \frac{COMP^0_{UR,\leftrightarrow}[a_5, b_3] + state\_comp^1_{UR,\leftrightarrow}(a_5, b_3)}{2} = \frac{1+0.956}{2} = 0.978$$

The Calculation of $\mathbf{COMP^1_{UR,\leftrightarrow}[a_5, b_4]}$

$$obs\_comp^1_{UR,\leftrightarrow}(a_5, b_4) = \frac{sum^1_{UR,\leftrightarrow}((a_5,b_4),E(a_5,T_5),R(b_4,T_4)) + sum^1_{UR,\leftrightarrow}((b_4,a_5),E(b_4,T_4),R(a_5,T_5))}{\|E(a_5,T_5)\| + \|E(b_4,T_4)\|}$$

Where:

$$\begin{aligned}
&E(a_5, T_5) = \varnothing \\
&R(a_5, T_5) = \{(a_4, chargeSchedule?pl_{chargeSchedule?}, a_6)\} \\
&tau(a_5, T_5) = \varnothing \\
&Fw(a_5, T_5) = E(a_5, T_5)
\end{aligned}$$

and

$$\begin{aligned}
R(b_4, T_4) =& \{(b_4, chargeParams?pl_{chargeParams?}, b_3), \\
& (b_4, transactionEvent?pl_{transactionEvent?}, b_5)\} \\
E(b_4, T_4) =& \{(b_4, chargeSchedule!pl_{chargeSchedule!}, b_0), \\
& (b_4, requestStop!pl_{requestStop!}, b_5)\} \\
tau(b_4, T_4) =& \varnothing \\
Fw(b_4, T_4) =& E(b_4, T_4) \cup R(b_4, T_4)
\end{aligned}$$

The function $sum^1_{UR,\leftrightarrow}((a_5, b_4), E(a_5, T_5), R(b_4, T_4))$ calculates the sum of the best compatibility between $E(a_5, T_5)$ and $R(b_4, T_4)$. Since $E(a_5, T_5) = \varnothing$,

$$sum^1_{UR,\leftrightarrow}((a_5, b_4), E(a_5, T_5), R(b_4, T_4)) = 0$$

The function $sum^1_{UR,\leftrightarrow}((b_4, a_5), E(b_4, T_4), R(a_5, T_5))$ calculates the sum of the best compatibility between $E(b_4, T_4)$ and $R(a_5, T_5)$. The calculation is reduced to only $(chargeParams!, chargeParams?)$ because $(requestStop!, chargeSchedule?)$ is not label compatible.

$$\begin{aligned}&sum^1_{UR,\leftrightarrow}((b_4,a_5),E(b_4,T_4),R(a_5,T_5))\\&= lab\_comp(chargeParams!, chargeParams?) * COMP^0_{UR,\leftrightarrow}[b_0,a_6]\end{aligned}$$

Where:

$$\begin{aligned}&lab\_comp(chargeParams!, chargeParams?)\\&= 1 - \frac{\|unsharedTypes(pl_{chargeParams!}, pl_{chargeParams?})\|)}{6(\|pl_{chargeParams!}\| + \|pl_{chargeParams?}\|)}\end{aligned}$$

and

$$COMP^0_{UR,\leftrightarrow}[b_0,a_6] = 1$$

From Equation 11, $lab\_comp(chargeParams!, chargeParams?) = 0.948$

$$\begin{aligned}&\Rightarrow sum^1_{UR,\leftrightarrow}((b_4,a_5),E(b_4,T_4),R(a_5,T_5)) = 0.948\\&\Rightarrow obs\_comp^1_{UR,\leftrightarrow}(a_5,b_4) = \frac{0+0.948}{0+2} = 0.474\end{aligned}$$

The state (a_5, b_4) does not have any outgoing or incoming τ transitions

$$\Rightarrow fw\_propag^k_{CN,\leftrightarrow}(a_5,b_4) = bw\_propag^1_{UR,\leftrightarrow}(a_5,b_4) = 0.474$$

The state compatibility of state (a_5, b_4) is calculated as follows:

$$\begin{aligned}&state\_comp^1_{UR,\leftrightarrow}(a_5,b_4)\\&= \frac{w_1 * fw\_propag^1_{UR,\leftrightarrow}(a_5,b_4) + w_2 * bw\_propag^1_{UR,\leftrightarrow}(a_5,b_4) + w_3 * nat(a_5,b_4)}{w_1+w_2+w_3}\end{aligned}$$

with $w_1 = 5$, $w_2 = 3$, and $w_3 = 1$ and $nat(a_5, b_4) = 1$ because $a_5 \notin (I_a \cup F_a)$ & $b_4 \notin (I_b \cup F_b)$

$$\Rightarrow state\_comp^1_{UR,\leftrightarrow}(a_5,b_4) = \frac{5*0.474+3*0.474+1*1}{5+3+1} = 0.532$$

And the compatibility degree is calculated as follows:

$$COMP^1_{UR,\leftrightarrow}[a_5,b_4] = \frac{COMP^0_{UR,\leftrightarrow}[a_5,b_4] + state\_comp^1_{UR,\leftrightarrow}(a_5,b_4)}{2} = \frac{1+0.532}{2} = 0.766$$

The Calculation of $\mathbf{COMP^1_{UR,\leftrightarrow}[a_5,b_0]}$, $\mathbf{COMP^1_{UR,\leftrightarrow}[a_5,b_1]}$, $\mathbf{COMP^1_{UR,\leftrightarrow}[a_5,b_2]}$, and $\mathbf{COMP^1_{UR,\leftrightarrow}[a_5,b_5]}$

By observing the Figure 18, the observable compatibility of the states (a_5,b_0), (a_5,b_1), (a_5,b_2), and (a_5,b_5) are equal to zero because of dead lock. Therefore, their forward and backward compatibility is also zero.
The state compatibility is calculated as follows:

$$\begin{aligned}
state\_comp^1_{UR,\leftrightarrow}(a_5,b_0) &= \frac{w_3 * nat(a_5,b_0)}{w_1+w_2+w_3} = \frac{1*0}{2+2+1} = 0\\
state\_comp^1_{UR,\leftrightarrow}(a_5,b_1) &= \frac{w_3 * nat(a_5,b_1)}{w_1+w_2+w_3} = \frac{1*1}{2+3+1} = 0.167\\
state\_comp^1_{UR,\leftrightarrow}(a_5,b_2) &= \frac{w_3 * nat(a_5,b_2)}{w_1+w_2+w_3} = \frac{1*1}{2+3+1} = 0.167\\
state\_comp^1_{UR,\leftrightarrow}(a_5,b_5) &= \frac{w_3 * nat(a_5,b_5)}{w_1+w_2+w_3} = \frac{1*0}{1+4+1} = 0
\end{aligned}$$

and

$$\begin{aligned}
COMP^1_{UR,\leftrightarrow}[a_5,b_0] &= \frac{COMP^0_{UR,\leftrightarrow}[a_5,b_0] + state\_comp^1_{UR,\leftrightarrow}(a_5,b_0)}{2} = \frac{1+0}{2} = 0.5\\
COMP^1_{UR,\leftrightarrow}[a_5,b_1] &= \frac{COMP^0_{UR,\leftrightarrow}[a_5,b_1] + state\_comp^1_{UR,\leftrightarrow}(a_5,b_1)}{2} = \frac{1+0.167}{2} = 0.584\\
COMP^1_{UR,\leftrightarrow}[a_5,b_2] &= \frac{COMP^0_{UR,\leftrightarrow}[a_5,b_2] + state\_comp^1_{UR,\leftrightarrow}(a_5,b_2)}{2} = \frac{1+0.167}{2} = 0.584\\
COMP^1_{UR,\leftrightarrow}[a_5,b_5] &= \frac{COMP^0_{UR,\leftrightarrow}[a_5,b_5] + state\_comp^1_{UR,\leftrightarrow}(a_5,b_5)}{2} = \frac{1+0}{2} = 0.5
\end{aligned}$$

The Calculation of $\mathbf{COMP^1_{UR,\leftrightarrow}[a_6,b_0]}$

$$obs\_comp^1_{UR,\leftrightarrow}(a_6,b_0) = \frac{sum^1_{UR,\leftrightarrow}((a_6,b_0),E(a_6,T_6),R(b_0,T_0)) + sum^1_{UR,\leftrightarrow}((b_0,a_6),E(b_0,T_0),R(a_6,T_6))}{\|E(a_6,T_6)\| + \|E(b_0,T_0)\|}$$

Where:

$$
\begin{aligned}
E(a_6, T_6) &= \{(a_6, transactionEvent!pl_{transactionEvent!}, a_7)\}\\
R(a_6, T_6) &= \varnothing\\
tau(a_6, T_6) &= \varnothing\\
Fw(a_6, T_6) &= E(a_6, T_6)
\end{aligned}
$$

and

$$
\begin{aligned}
R(b_0, T_0) = \{&(b_0, certReqest?pl_{certRequest?}, b_1),\\
&(b_0, authorize?pl_{authorize?}, b_2),\\
&(b_0, chargeParams?pl_{chargeParams?}, b_3),\\
&(b_0, transactionEvent?pl_{transactionEvent?}, b_4)\}\\
E(b_0, T_0) =&\varnothing\\
tau(b_0, T_0) =&\varnothing\\
Fw(b_0, T_0) =&R(b_0, T_0)
\end{aligned}
$$

The function $sum^1_{UR,\leftrightarrow}((a_6, b_0), E(a_6, T_6), R(b_0, T_0))$ calculates the sum of the best compatibility between $E(a_6, T_6)$ and $R(b_0, T_0)$.

$$
\begin{aligned}
&sum^1_{UR,\leftrightarrow}((a_6, b_0), E(a_6, T_6), R(b_0, T_0))\\
&= lab\_comp(transactionEvent!, transactionEvent?) * COMP^0_{UR,\leftrightarrow}[a_7, b_4]
\end{aligned}
$$

Where:

$$
\begin{aligned}
&lab\_comp(transactionEvent!, transactionEvent?)\\
&= 1 - \frac{\|unsharedTypes(pl_{transactionEvent!}, pl_{transactionEvent?})\|)}{6(\|pl_{transactionEvent!}\| + \|pl_{transactionEvent?}\|)}
\end{aligned}
$$

and

$$COMP^0_{UR,\leftrightarrow}[a_7, b_4] = 1$$

Table 20 shows that the un-shared data types between two messages are $unshared = \{unsignedLong, number, base64Binary, string\}$

$$\begin{aligned}
&\Rightarrow lab\_comp(transactionEvent!, transactionEvent?)\\
&= 1 - \frac{4}{6(2+2)}\\
&= 0.833
\end{aligned} \tag{13}$$

$$\Rightarrow sum^1_{UR,\leftrightarrow}((a_6, b_0), E(a_6, T_6), R(b_0, T_0)) = 0.833$$

The function $sum^1_{UR,\leftrightarrow}((b_0, a_6), E(b_0, T_0), R(a_6, T_6))$ is based upon the compatibility between the $E(b_0, T_0)$ and $R(a_6, T_6)$. However, $E(b_0, T_0) = \varnothing$,

$$\begin{aligned}
&\Rightarrow sum^1_{UR,\leftrightarrow}((b_0, a_6), E(b_0, T_0), R(a_6, T_6)) = 0\\
&\Rightarrow obs\_comp^1_{UR,\leftrightarrow}(a_4, b_0) = \frac{0.833 + 0}{1} = 0.833
\end{aligned}$$

The state (a_6, b_0) does not have any outgoing or incoming τ transitions

$$\Rightarrow fw\_propag^k_{UR,\leftrightarrow}(a_6, b_0) = bw\_propag^1_{UR,\leftrightarrow}(a_6, b_0) = 0.833$$

The state compatibility of state (a_6, b_0) is calculated as follows:

$$\begin{aligned}
&state\_comp^1_{UR,\leftrightarrow}(a_6, b_0)\\
&= \frac{w_1 * fw\_propag^1_{UR,\leftrightarrow}(a_6, b_0) + w_2 * bw\_propag^1_{UR,\leftrightarrow}(a_6, b_0) + w_3 * nat(a_6, b_0)}{w_1 + w_2 + w_3}
\end{aligned}$$

with $w_1 = 5$, $w_2 = 6$, and $w_3 = 1$ and $nat(a_6, b_0) = 0$ because $a_5 \notin (I_a \cup F_a)$ & $b_0 \in I_b$

$$\Rightarrow state\_comp^1_{UR,\leftrightarrow}(a_6, b_0) = \frac{5 * 0.833 + 6 * 0.833 + 1 * 0}{5 + 6 + 1} = 0.764$$

And the compatibility degree is calculated as follows:

$$COMP^1_{UR,\leftrightarrow}[a_6, b_0] = \frac{COMP^0_{UR,\leftrightarrow}[a_6, b_0] + state\_comp^1_{UR,\leftrightarrow}(a_6, b_0)}{2} = \frac{1 + 0.764}{2} = 0.882$$

The Calculation of $\mathbf{COMP^1_{UR,\leftrightarrow}[a_6, b_4]}$

$$obs\_comp^1_{UR,\leftrightarrow}(a_6, b_4) = \frac{sum^1_{UR,\leftrightarrow}((a_6,b_4),E(a_6,T_6),R(b_4,T_4)) + sum^1_{UR,\leftrightarrow}((b_4,a_6),E(b_4,T_4),R(a_6,T_6))}{\|E(a_6,T_6)\| + \|E(b_4,T_4)\|}$$

Where:

$$\begin{aligned}
E(a_6, T_6) &= \{(a_6, transactionEvent!pl_{transactionEvent!}, a_7)\} \\
R(a_6, T_6) &= \varnothing \\
tau(a_6, T_6) &= \varnothing \\
Fw(a_6, T_6) &= E(a_6, T_6)
\end{aligned}$$

and

$$\begin{aligned}
R(b_4, T_4) =& \{(b_4, chargeParams?pl_{chargeParams?}, b_3), \\
& (b_4, transactionEvent?pl_{transactionEvent?}, b_5)\} \\
E(b_4, T_4) =& \{(b_4, chargeSchedule!pl_{chargeSchedule!}, b_0), \\
& (b_4, requestStop!pl_{requestStop!}, b_5)\} \\
tau(b_4, T_4) =& \varnothing \\
Fw(b_4, T_4) =& E(b_4, T_4) \cup R(b_4, T_4)
\end{aligned}$$

The function $sum^1_{UR,\leftrightarrow}((a_6, b_4), E(a_6, T_6), R(b_4, T_4))$ calculates the sum of the best compatibility between $E(a_6, T_6)$ and $R(b_4, T_4)$.

$$\begin{aligned}
& sum^1_{UR,\leftrightarrow}((a_6, b_4), E(a_6, T_6), R(b_4, T_4)) \\
& = lab\_comp(transactionEvent!, transactionEvent?) * COMP^0_{UR,\leftrightarrow}[a_7, b_3]
\end{aligned}$$

Where:

$$\begin{aligned}
& lab\_comp(transactionEvent!, transactionEvent?) \\
& = 1 - \frac{\|unsharedTypes(pl_{transactionEvent!}, pl_{transactionEvent?})\|)}{6(\|pl_{transactionEvent!}\| + \|pl_{transactionEvent?}\|)}
\end{aligned}$$

and

$$COMP^0_{UR,\leftrightarrow}[a_7, b_3] = 1$$

Table 20 shows that the un-shared data types between two messages are $unshared = \{unsignedLong, number, base64Binary, string\}$

$$\begin{aligned}\Rightarrow & lab\_comp(chargeParams!, chargeParams?)\\ &= 1 - \frac{4}{6(2+2)}\\ &= 0.833\end{aligned}$$

$$\Rightarrow sum^1_{UR,\leftrightarrow}((a_6, b_4), E(a_6, T_6), R(b_4, T_4)) = 0.833$$

The function $sum^1_{UR,\leftrightarrow}((b_3, a_6), E(b_3, T_3), R(a_6, T_6))$ is based upon the compatibility between the $E(b_4, T_4)$ and $R(a_6, T_6)$. However, $R(a_6, T_6) = \varnothing$,

$$\begin{aligned}&\Rightarrow sum^1_{UR,\leftrightarrow}((b4, a_6), E(b_4, T_4), R(a_6, T_6)) = 0\\ &\Rightarrow obs\_comp^1_{UR,\leftrightarrow}(a_6, b_4) = \frac{0.833 + 0}{1 + 2} = 0.278\end{aligned}$$

The state (a_6, b_4) does not have any outgoing or incoming τ transitions

$$\Rightarrow fw\_propag^k_{UR,\leftrightarrow}(a_6, b_4) = bw\_propag^1_{UR,\leftrightarrow}(a_6, b_4) = 0.278$$

The state compatibility of state (a_6, b_4) is calculated as follows:

$$\begin{aligned}&state\_comp^1_{UR,\leftrightarrow}(a_6, b_4)\\ &= \frac{w_1 * fw\_propag^1_{UR,\leftrightarrow}(a_6, b_4) + w_2 * bw\_propag^1_{UR,\leftrightarrow}(a_6, b_4) + w_3 * nat(a_6, b_4)}{w_1 + w_2 + w_3}\end{aligned}$$

with $w_1 = 5$, $w_2 = 3$, and $w_3 = 1$ and $nat(a_6, b_4) = 1$ because $a_6 \notin (I_a \cup F_a)$ & $b_4 \notin (I_b \cup F_b)$

$$\Rightarrow state\_comp^1_{UR,\leftrightarrow}(a_6, b_4) = \frac{5 * 0.278 + 3 * 0.278 + 1 * 1}{5 + 3 + 1} = 0.358$$

And the compatibility degree is calculated as follows:

$$COMP^1_{UR,\leftrightarrow}[a_6, b_4] = \frac{COMP^0_{UR,\leftrightarrow}[a_6, b_4] + state\_comp^1_{UR,\leftrightarrow}(a_5, b_3)}{2} = \frac{1 + 0.358}{2} = 0.679$$

The Calculation of $\mathbf{COMP^1_{UR,\leftrightarrow}[a_6, b_1]}$, $\mathbf{COMP^1_{UR,\leftrightarrow}[a_6, b_2]}$, $\mathbf{COMP^1_{UR,\leftrightarrow}[a_6, b_3]}$, and $\mathbf{COMP^1_{UR,\leftrightarrow}[a_5, b_5]}$

By observing the Figure 18, the observable compatibility of the states (a_6, b_1), (a_6, b_2),

(a_6, b_3), and (a_6, b_5) are equal to zero because of dead lock. Therefore, their forward and backward compatibility is also zero.
The state compatibility is calculated as follows:

$$\begin{aligned}
state\_comp^1_{UR,\leftrightarrow}(a_6, b_1) &= \frac{w_3 * nat(a_6, b_2)}{w_1 + w_2 + w_3} = \frac{1 * 1}{2 + 3 + 1} = 0.167 \\
state\_comp^1_{UR,\leftrightarrow}(a_6, b_2) &= \frac{w_3 * nat(a_6, b_2)}{w_1 + w_2 + w_3} = \frac{1 * 1}{2 + 3 + 1} = 0.167 \\
state\_comp^1_{UR,\leftrightarrow}(a_6, b_3) &= \frac{w_3 * nat(a_6, b_3)}{w_1 + w_2 + w_3} = \frac{1 * 1}{2 + 4 + 1} = 0.142 \\
state\_comp^1_{UR,\leftrightarrow}(a_6, b_5) &= \frac{w_3 * nat(a_6, b_5)}{w_1 + w_2 + w_3} = \frac{1 * 0}{1 + 4 + 1} = 0
\end{aligned}$$

and

$$\begin{aligned}
COMP^1_{UR,\leftrightarrow}[a_6, b_1] &= \frac{COMP^0_{UR,\leftrightarrow}[a_6, b_1] + state\_comp^1_{UR,\leftrightarrow}(a_6, b_1)}{2} = \frac{1 + 0.167}{2} = 0.584 \\
COMP^1_{UR,\leftrightarrow}[a_6, b_2] &= \frac{COMP^0_{UR,\leftrightarrow}[a_6, b_2] + state\_comp^1_{UR,\leftrightarrow}(a_6, b_2)}{2} = \frac{1 + 0.167}{2} = 0.584 \\
COMP^1_{UR,\leftrightarrow}[a_6, b_3] &= \frac{COMP^0_{UR,\leftrightarrow}[a_6, b_3] + state\_comp^1_{UR,\leftrightarrow}(a_6, b_3)}{2} = \frac{1 + 0.142}{2} = 0.571 \\
COMP^1_{UR,\leftrightarrow}[a_6, b_5] &= \frac{COMP^0_{UR,\leftrightarrow}[a_6, b_5] + state\_comp^1_{UR,\leftrightarrow}(a_6, b_5)}{2} = \frac{1 + 0}{2} = 0.5
\end{aligned}$$

The Calculation of $\mathbf{COMP^1_{UR,\leftrightarrow}[a_7, b_0]}$

$$obs\_comp^1_{UR,\leftrightarrow}(a_7, b_0) = \frac{sum^1_{UR,\leftrightarrow}((a_7,b_0),E(a_7,T_7),R(b_0,T_0)) + sum^1_{UR,\leftrightarrow}((b_0,a_7),E(b_0,T_0),R(a_7,T_7))}{\|E(a_7,T_7)\| + \|E(b_0,T_0)\|}$$

Where:

$$\begin{aligned}
E(a_7, T_7) =& \{(a_7, transactionEvent!pl_{transactionEvent!}, a_8), \\
& (a_7, chargeParams!pl_{chargeParams!}, a_5)\} \\
R(a_7, T_7) =& \{(a_7, requestStop?_pl_{requestStop?}, a_8), \\
& (a_7, chargeSchedule?pl_{chargeSchedule?}, a_5)\} \\
tau(a_7, T_7) =& \varnothing \\
Fw(a_7, T_7) =& E(a_7, T_7) \cup R(a_7, T_7)
\end{aligned}$$

and

$$\begin{aligned}
R(b_0,T_0) &= \{(b_0, certReqest?pl_{certRequest?}, b_1),\\
&\quad (b_0, authorize?pl_{authorize?}, b_2),\\
&\quad (b_0, chargeParams?pl_{chargeParams?}, b_3),\\
&\quad (b_0, transactionEvent?pl_{transactionEvent?}, b_4)\}\\
E(b_0,T_0) &= \varnothing\\
tau(b_0,T_0) &= \varnothing\\
Fw(b_0,T_0) &= R(b_0,T_0)
\end{aligned}$$

The function $sum^1_{UR,\leftrightarrow}((a_7,b_0),E(a_7,T_7),R(b_0,T_0))$ calculates the sum of the best compatibility between $E(a_7,T_7)$ and $R(b_0,T_0)$.

$$\begin{aligned}
&sum^1_{UR,\leftrightarrow}((a_7,b_0),E(a_7,T_7),R(b_0,T_0))\\
&= lab\_comp(transactionEvent!, transactionEvent?) * COMP^0_{UR,\leftrightarrow}[a_8,b_4]\\
&+ lab\_comp(chargeParams!, chargeParams?) * COMP^0_{UR,\leftrightarrow}[a_5,b_3]
\end{aligned}$$

Where:

$$\begin{aligned}
&lab\_comp(transactionEvent!, transactionEvent?)\\
&= 1 - \frac{\|unsharedTypes(pl_{transactionEvent!}, pl_{transactionEvent?})\|)}{6(\|pl_{transactionEvent!}\| + \|pl_{transactionEvent?}\|)}\\
&+ 1 - \frac{\|unsharedTypes(pl_{chargeParams!}, pl_{chargeParams?})\|)}{6(\|pl_{chargeParams!}\| + \|pl_{chargeParams?}\|)}
\end{aligned}$$

and

$$COMP^0_{UR,\leftrightarrow}[a_8,b_4] = 1, COMP^0_{UR,\leftrightarrow}[a_5,b_3] = 1$$

From Equation 13, $lab\_comp(transactionEvent!, transactionEvent?) = 0.833$ and from Equation 11, $lab\_comp(chargeParams!, chargeParams?) = 0.948$

$$\Rightarrow sum^1_{UR,\leftrightarrow}((a_7,b_0),E(a_7,T_7),R(b_0,T_0)) = 0.833 + 0.948 = 1.781$$

The function $sum^1_{UR,\leftrightarrow}((b_0,a_7),E(b_0,T_0),R(a_7,T_7))$ is based upon the compatibility between the $E(b_0,T_0)$ and $R(a_7,T_7)$. However, $E(b_0,T_0) = \varnothing$,

$$
\begin{aligned}
&\Rightarrow sum^1_{UR,\leftrightarrow}((b_0,a_7),E(b_0,T_0),R(a_7,T_7)) = 0\\
&\Rightarrow obs\_comp^1_{UR,\leftrightarrow}(a_7,b_0) = \frac{1.781+0}{2} = 0.891
\end{aligned}
$$

The state (a_7, b_0) does not have any incoming or outgoing τ transitions

$$\Rightarrow fw\_propag^k_{CN,\leftrightarrow}(a_7,b_0) = bw\_propag^1_{UR,\leftrightarrow}(a_7,b_0) = 0.891$$

The state compatibility of state (a_7, b_0) is calculated as follows:

$$
\begin{aligned}
&state\_comp^1_{UR,\leftrightarrow}(a_7,b_0)\\
&= \frac{w_1 * fw\_propag^1_{UR,\leftrightarrow}(a_7,b_0) + w_2 * bw\_propag^1_{UR,\leftrightarrow}(a_7,b_0) + w_3 * nat(a_7,b_0)}{w_1+w_2+w_3}
\end{aligned}
$$

with $w_1 = 8$, $w_2 = 5$, and $w_3 = 1$ and $nat(a_7, b_0) = 0$ because $a_7 \notin (I_a \cup F_a)$ & $b_0 \in I_b$

$$\Rightarrow state\_comp^1_{UR,\leftrightarrow}(a_7,b_0) = \frac{8*0.891+5*0.891+1*0}{8+5+1} = 0.827$$

And the compatibility degree is calculated as follows:

$$COMP^1_{UR,\leftrightarrow}[a_7,b_0] = \frac{COMP^0_{UR,\leftrightarrow}[a_7,b_0] + state\_comp^1_{UR,\leftrightarrow}(a_7,b_0)}{2} = \frac{1+0.827}{2} = 0.914$$

The Calculation of $\mathbf{COMP^1_{UR,\leftrightarrow}[a_7, b_3]}$

$$obs\_comp^1_{UR,\leftrightarrow}(a_7,b_3) = \frac{sum^1_{UR,\leftrightarrow}((a_7,b_3),E(a_7,T_7),R(b_3,T_3)) + sum^1_{UR,\leftrightarrow}((b_3,a_7),E(b_3,T_3),R(a_7,T_7))}{\|E(a_7,T_7)\|+\|E(b_3,T_3)\|}$$

Where:

$$
\begin{aligned}
E(a_7,T_7) =& \{(a_7, transactionEvent!pl_{transactionEvent!}, a_8),\\
& (a_7, chargeParams!pl_{chargeParams!}, a_5)\}\\
R(a_7,T_7) =& \{(a_7, requestStop?_{pl_{requestStop?}}, a_8),\\
& (a_7, chargeSchedule?pl_{chargeSchedule?}, a_5)\}\\
tau(a_7,T_7) =& \varnothing\\
Fw(a_7,T_7) =& E(a_7,T_7) \cup R(a_7,T_7)
\end{aligned}
$$

and

$$\begin{aligned}
R(b_3, T_3) =& \varnothing \\
E(b_3, T_3) =& \{(b_3, chargeSchedule!pl_{chargeSchedule!}, b_0)\} \\
tau(b_3, T_3) =& \varnothing \\
Fw(b_3, T_3) =& E(b_3, T_3)
\end{aligned}$$

The function $sum^1_{UR,\leftrightarrow}((a_7, b_3), E(a_7, T_7), R(b_3, T_3))$ calculates the sum of the best compatibility between $E(a_7, T_7)$ and $R(b_3, T_3)$. Since there is no compatible messages in (a_7, b_3)

$$sum^1_{UR,\leftrightarrow}((a_7, b_3), E(a_7, T_7), R(b_3, T_3)) = 0$$

The function $sum^1_{UR,\leftrightarrow}((b_3, a_7), E(b_3, T_3), R(a_7, T_7))$ is based upon the best compatibility between the $E(b_3, T_3)$ and $R(a_7, T_7)$.

$$\begin{aligned}
&sum^1_{UR,\leftrightarrow}((b_3, a_7), E(b_3, T_3), R(a_7, T_7)) \\
&= lab\_comp(chargeSchedule!, chargeSchedule?) * COMP^0_{UR,\leftrightarrow}[b_0, a_5]
\end{aligned}$$

From Equation 12, $lab\_comp(chargeSchedule!, chargeSchedule?) = 0.949$. Furthermore, $COMP^0_{UR,\leftrightarrow}[b_0, a_5] = 1$

$$\begin{aligned}
\Rightarrow& sum^1_{UR,\leftrightarrow}((b_3, a_7), E(b_3, T_3), R(a_7, T_7)) = 0.949 \\
\Rightarrow& obs\_comp^1_{UR,\leftrightarrow}(a_7, b_3) = \frac{0 + 0.949}{1 + 2} = 0.316
\end{aligned}$$

The state (a_7, b_3) does not have any incoming or outgoing τ transitions

$$\Rightarrow fw\_propag^k_{CN,\leftrightarrow}(a_5, b_3) = bw\_propag^1_{UR,\leftrightarrow}(a_5, b_3) = 0.316$$

The state compatibility of state (a_7, b_3) is calculated as follows:

$$\begin{aligned}
&state\_comp^1_{UR,\leftrightarrow}(a_7, b_3) \\
&= \frac{w_1 * fw\_propag^1_{CN,\leftrightarrow}(a_7, b_3) + w_2 * bw\_propag^1_{CN,\leftrightarrow}(a_7, b_3) + w_3 * nat(a_7, b_3)}{w_1 + w_2 + w_3}
\end{aligned}$$

with $w_1 = 5$, $w_2 = 3$, and $w_3 = 1$ and $nat(a_7, b_3) = 1$ because $a_7 \notin (I_a \cup F_a)$ & $b_3 \notin (I_b \cup F_b)$

$$\Rightarrow state\_comp^1_{UR,\leftrightarrow}(a_7, b_3) = \frac{5 * 0.316 + 3 * 0.316 + 1 * 1}{5 + 3 + 1} = 0.392$$

And the compatibility degree is calculated as follows:

$$COMP^1_{UR,\leftrightarrow}[a_7, b_3] = \frac{COMP^0_{UR,\leftrightarrow}[a_7, b_3] + state\_comp^1_{UR,\leftrightarrow}(a_7, b_3)}{2} = \frac{1 + 0.392}{2} = 0.696$$

The Calculation of $\mathbf{COMP^1_{UR,\leftrightarrow}[a_7, b_4]}$

$$obs\_comp^1_{UR,\leftrightarrow}(a_7, b_4) = \frac{sum^1_{UR,\leftrightarrow}((a_7,b_4),E(a_7,T_7),R(b_4,T_4))+sum^1_{UR,\leftrightarrow}((b_4,a_7),E(b_4,T_4),R(a_7,T_7))}{\|E(a_7,T_7)\|+\|E(b_4,T_4)\|}$$

Where:

$$\begin{aligned}
E(a_7, T_7) =& \{(a_7, transactionEvent!pl_{transactionEvent!}, a_8),\\
& (a_7, chargeParams!pl_{chargeParams!}, a_5)\}\\
R(a_7, T_7) =& \{(a_7, requestStop?_pl_{requestStop?}, a_8),\\
& (a_7, chargeSchedule?pl_{chargeSchedule?}, a_5)\}\\
tau(a_7, T_7) =& \varnothing\\
Fw(a_7, T_7) =& E(a_7, T_7) \cup R(a_7, T_7)
\end{aligned}$$

and

$$\begin{aligned}
R(b_4, T_4) =& \{(b_4, chargeParams?pl_{chargeParams?}, b_3),\\
& (b_4, transactionEvent?pl_{transactionEvent?}, b_5)\}\\
E(b_4, T_4) =& \{(b_4, chargeSchedule!pl_{chargeSchedule!}, b_0),\\
& (b_4, requestStop!pl_{requestStop!}, b_5)\}\\
tau(b_4, T_4) =& \varnothing\\
Fw(b_4, T_4) =& E(b_4, T_4) \cup R(b_4, T_4)
\end{aligned}$$

The function $sum^1_{UR,\leftrightarrow}((a_7, b_4), E(a_7, T_7), R(b_4, T_4))$ calculates the sum of the best compatibility between $E(a_7, T_7)$ and $R(b_4, T_4)$.

$$
\begin{aligned}
&sum^1_{UR,\leftrightarrow}((a_7,b_4),E(a_7,T_7),R(b_4,T_4))\\
=&lab\_comp(transactionEvent!,transactionEvent?) * COMP^0_{UR,\leftrightarrow}[a_8,b_5]\\
&+ lab\_comp(chargeParams!,chargeParams?) * COMP^0_{UR,\leftrightarrow}[a_5,b_3]
\end{aligned}
$$

From 13, $lab\_comp(transactionEvent!,transactionEvent?) = 0.833$
From 11, $lab\_comp(chargeParams!,chargeParams?) = 0.948$

$$
\begin{aligned}
COMP^0_{UR,\leftrightarrow}[a_8,b_5] = 1\\
COMP^0_{UR,\leftrightarrow}[a_5,b_3] = 1
\end{aligned}
$$

$$\Rightarrow sum^1_{UR,\leftrightarrow}((a_7,b_4),E(a_7,T_7),R(b_4,T_4)) = 0.833 + 0.948 = 1.781$$

The function $sum^1_{UR,\leftrightarrow}((b_4,a_7),E(b_4,T_4),R(a_7,T_7))$ is based upon the compatibility between the $E(b_4,T_4)$ and $R(a_7,T_7)$.

$$
\begin{aligned}
&sum^1_{UR,\leftrightarrow}((a_7,b_4),E(a_7,T_7),R(b_4,T_4))\\
=&lab\_comp(requestStop!,requestStop?) * COMP^0_{UR,\leftrightarrow}[b_5,a_8]\\
=&lab\_comp(chargeSchedule!,chargeSchedule?) * COMP^0_{UR,\leftrightarrow}[b_0,a_5]
\end{aligned}
$$

From Equation 12, $lab\_comp(chargeSchedule!,chargeSchedule?) = 0.949$. Additionally, $lab\_comp(requestStop!,requestStop?) = 1$ because of a perfect match.
And

$$
\begin{aligned}
COMP^0_{UR,\leftrightarrow}[b_5,a_8] = 1\\
COMP^0_{UR,\leftrightarrow}[b_0,a_5] = 1
\end{aligned}
$$

$$
\begin{aligned}
&\Rightarrow sum^1_{UR,\leftrightarrow}((b4,a_7),E(b_4,T_4),R(a_7,T_7)) = 1 + 0.949 = 1.949\\
&\Rightarrow obs\_comp^1_{UR,\leftrightarrow}(a_6,b_4) = \frac{1.781 + 1.949}{2+2} = 0.933
\end{aligned}
$$

The state (a_7,b_4) does not have any outgoing or incoming τ transition

$$\Rightarrow fw\_propag^k_{UR,\leftrightarrow}(a_6,b_4) = bw\_propag^1_{UR,\leftrightarrow}(a_6,b_4) = 0.933$$

The state compatibility of state (a_7, b_4) is calculated as follows:

$$state\_comp^1_{UR,\leftrightarrow}(a_7, b_4)$$
$$= \frac{w_1 * fw\_propag^1_{UR,\leftrightarrow}(a_7, b_4) + w_2 * bw\_propag^1_{UR,\leftrightarrow}(a_7, b_4) + w_3 * nat(a_7, b_4)}{w_1 + w_2 + w_3}$$

with $w_1 = 8$, $w_2 = 2$, and $w_3 = 1$ and $nat(a_7, b_4) = 1$ because $a_7 \notin (I_a \cup F_a)$ & $b_4 \notin (I_b \cup F_b)$

$$\Rightarrow state\_comp^1_{UR,\leftrightarrow}(a_7, b_4) = \frac{8 * 0.933 + 2 * 0.933 + 1 * 1}{8 + 2 + 1} = 0.939$$

And the compatibility degree is calculated as follows:

$$COMP^1_{UR,\leftrightarrow}[a_7, b_4] = \frac{COMP^0_{UR,\leftrightarrow}[a_7, b_4] + state\_comp^1_{UR,\leftrightarrow}(a_7, b_4)}{2} = \frac{1 + 0.939}{2} = 0.97$$

The Calculation of $\mathbf{COMP^1_{UR,\leftrightarrow}[a_7, b_1]}$, $\mathbf{COMP^1_{UR,\leftrightarrow}[a_7, b_2]}$, and $\mathbf{COMP^1_{UR,\leftrightarrow}[a_7, b_5]}$

By observing the Figure 18, the observable compatibility of the states (a_7, b_1), (a_7, b_2), and (a_7, b_5) are equal to zero because of dead lock. Therefore, their forward and backward compatibility is also zero.
The state compatibility is calculated as follows:

$$state\_comp^1_{UR,\leftrightarrow}(a_7, b_1) = \frac{w_3 * nat(a_7, b_2)}{w_1 + w_2 + w_3} = \frac{1 * 1}{5 + 2 + 1} = 0.125$$
$$state\_comp^1_{UR,\leftrightarrow}(a_7, b_2) = \frac{w_3 * nat(a_7, b_2)}{w_1 + w_2 + w_3} = \frac{1 * 1}{5 + 2 + 1} = 0.125$$
$$state\_comp^1_{UR,\leftrightarrow}(a_7, b_5) = \frac{w_3 * nat(a_7, b_5)}{w_1 + w_2 + w_3} = \frac{1 * 0}{5 + 3 + 1} = 0$$

and

$$COMP^1_{UR,\leftrightarrow}[a_7, b_1] = \frac{COMP^0_{UR,\leftrightarrow}[a_7, b_1] + state\_comp^1_{UR,\leftrightarrow}(a_7, b_1)}{2} = \frac{1 + 0.125}{2} = 0.563$$
$$COMP^1_{UR,\leftrightarrow}[a_7, b_2] = \frac{COMP^0_{UR,\leftrightarrow}[a_7, b_2] + state\_comp^1_{UR,\leftrightarrow}(a_7, b_2)}{2} = \frac{1 + 0.125}{2} = 0.563$$
$$COMP^1_{UR,\leftrightarrow}[a_7, b_5] = \frac{COMP^0_{UR,\leftrightarrow}[a_7, b_5] + state\_comp^1_{UR,\leftrightarrow}(a_7, b_5)}{2} = \frac{1 + 0}{2} = 0.5$$

The Calculation of $\mathbf{COMP^1_{UR,\leftrightarrow}[a_8, b_5]}$

By observing the Figure 18, it can be seen that (a_8, b_5) are final states.

$$\Rightarrow obs\_comp(a_8, b_5) = 1$$
$$\Rightarrow fw\_propag^k_{UR,\leftrightarrow}(a_8, b_5) = 0 \ \ \& \ \ bw\_propag^k_{UR,\leftrightarrow}(a_8, b_5) = 1$$

The state compatibility of state (a_8, b_5) is calculated as follows:

$$state\_comp^1_{UR,\leftrightarrow}(a_8, b_5) = \frac{w_1 * fw\_propag^1_{UR,\leftrightarrow}(a_8, b_5) + w_2 * bw\_propag^1_{UR,\leftrightarrow}(a_8, b_5) + w_3 * nat(a_8, b_5)}{w_1 + w_2 + w_3}$$

with $w_1 = 0$, $w_2 = 4$, and $w_3 = 1$ and $nat(a_8, b_5) = 1$ because $a_8 \in F_a$ & $b_5 \in F_b$

$$\Rightarrow state\_comp^1_{UR,\leftrightarrow}(a_8, b_5) = \frac{0 * 1 + 4 * 1 + 1 * 1}{0 + 4 + 1} = 1$$

And the compatibility degree is calculated as follows:

$$COMP^1_{UR,\leftrightarrow}[a_8, b_5] = \frac{COMP^0_{UR,\leftrightarrow}[a_8, b_5] + state\_comp^1_{UR,\leftrightarrow}(a_8, b_5)}{2} = \frac{1 + 1}{2} = 1$$

The Calculation of $\mathbf{COMP^1_{UR,\leftrightarrow}[a_8, b_0]}$, $\mathbf{COMP^1_{UR,\leftrightarrow}[a_8, b_1]}$, $\mathbf{COMP^1_{UR,\leftrightarrow}[a_8, b_2]}$, $\mathbf{COMP^1_{UR,\leftrightarrow}[a_8, b_3]}$, and $\mathbf{COMP^1_{UR,\leftrightarrow}[a_8, b_4]}$

By observing the Figure 18, it can be seen that (a_8, b_0), (a_8, b_1), (a_8, b_2), (a_8, b_3) ,(a_8, b_4) are in dead lock because a_8 has no emission messages. Hence their forward and backward compatibility is equal zero. Furthermore, those states do not the same nature, since one is final state and the other is neither final or initial state.

$$\begin{aligned}
\Rightarrow state\_comp^1_{UR,\leftrightarrow}(a_8, b_0) =& state\_comp^1_{UR,\leftrightarrow}(a_8, b_1) \\
=& state\_comp^1_{UR,\leftrightarrow}(a_8, b_2) \\
=& state\_comp^1_{UR,\leftrightarrow}(a_8, b_3) \\
=& state\_comp^1_{UR,\leftrightarrow}(a_8, b_4) \\
=& 0
\end{aligned}$$

$$\begin{aligned}\Rightarrow COMP^1_{UR,\leftrightarrow}[a_8, b_0] =& COMP^1_{UR,\leftrightarrow}[a_8, b_1]\\ =& COMP^1_{UR,\leftrightarrow}[a_8, b_2]\\ =& COMP^1_{UR,\leftrightarrow}[a_8, b_3]\\ =& COMP^1_{UR,\leftrightarrow}[a_8, b_4]\\ =& \frac{1+0}{2} = 0.5\end{aligned}$$

Appendix E - Code Listings

Listing 2: Example of data structures having ordered parameters

```
/**Finctional ISO 15118 data struture
 */
typedef struct
{
    int param1;
    float param2;
    char param3[64];
}iso15118_struture_b;

/**Finctional OCPP data struture
 */
typedef struct
{
    int param1;
    float param2;
    char param3[64];
}ocpp_struture_b;

iso15118_struture_b iso15118_data ={
    .param1 = 0xBEEFCAFE,
    .param2 = 3.14,
    .param3 = "hello wolrd"};
```

```
ocpp_struture_b ocpp_data = {0};
```

Listing 3: Example of data structures having unordered parameters

```
typedef base64 char;

/* Fictional ISO 15118 data structure */
typedef struct
{
    byte param1;
    double param2;
    base64 param3[64];
}iso15118_struture_b;

/* Fictional OCPP data structure */
typedef struct
{
    uint8_t param3[64];
    float param2;
    int param1;
}ocpp_struture_b;

void ConvertIso2Ocpp(iso15118_struture_b *iso, ocpp_struture_b *
   ocpp)
{
    ocpp->param1 = (int)iso->param1;
    ocpp->param2 = (double)iso->param2
    DecodeBase64(iso->param3,sizeof(iso->param3),ocpp->param3);
}
```

Listing 4: Format of the json file to describe the STS

```
{
    "graph_name":"NAME_OF_THE_GRAPH",
    "states":
    [
        {
            "state_name":"NAME_OF_STATE",
            "state_type":"STATE_TYPE",
                "transitions":
                [
                    {
                        "transition_name":"NAME_OF_THE_TRANSITION",
                        "transition_type":"TRANSITION_TYPE",
                        "params":["name:data_type", "name:data_type
                            ", "name:data_type"],
                        "next_state":"NAME_OF_STATE"
                    },
                ],
            },
            {
                "state_name":"NAME_OF_STATE",
                "state_type":"STATE_TYPE",
                "transitions":
                [
                    {
                        "transition_name":"NAME_OF_THE_TRANSITION",
                        "transition_type":"TRANSITION_TYPE",
                        "params":["name:data_type", "name:data_type
                            ", "name:data_type"],
                        "next_state":"NAME_OF_STATE"
                    },
                ],
            },
    ]
}
```

Typical commands of the Compatibility calculation application

Listing 5: Command to display help menu

```
python compatibility_calculation.py --help
```

Listing 6: Command to start the protocol compatibility calculation

```
python compatibility_calculation.py --graph iso_15118.json ocpp
    .json --iterate 10 --log_level none --output test.txt
```

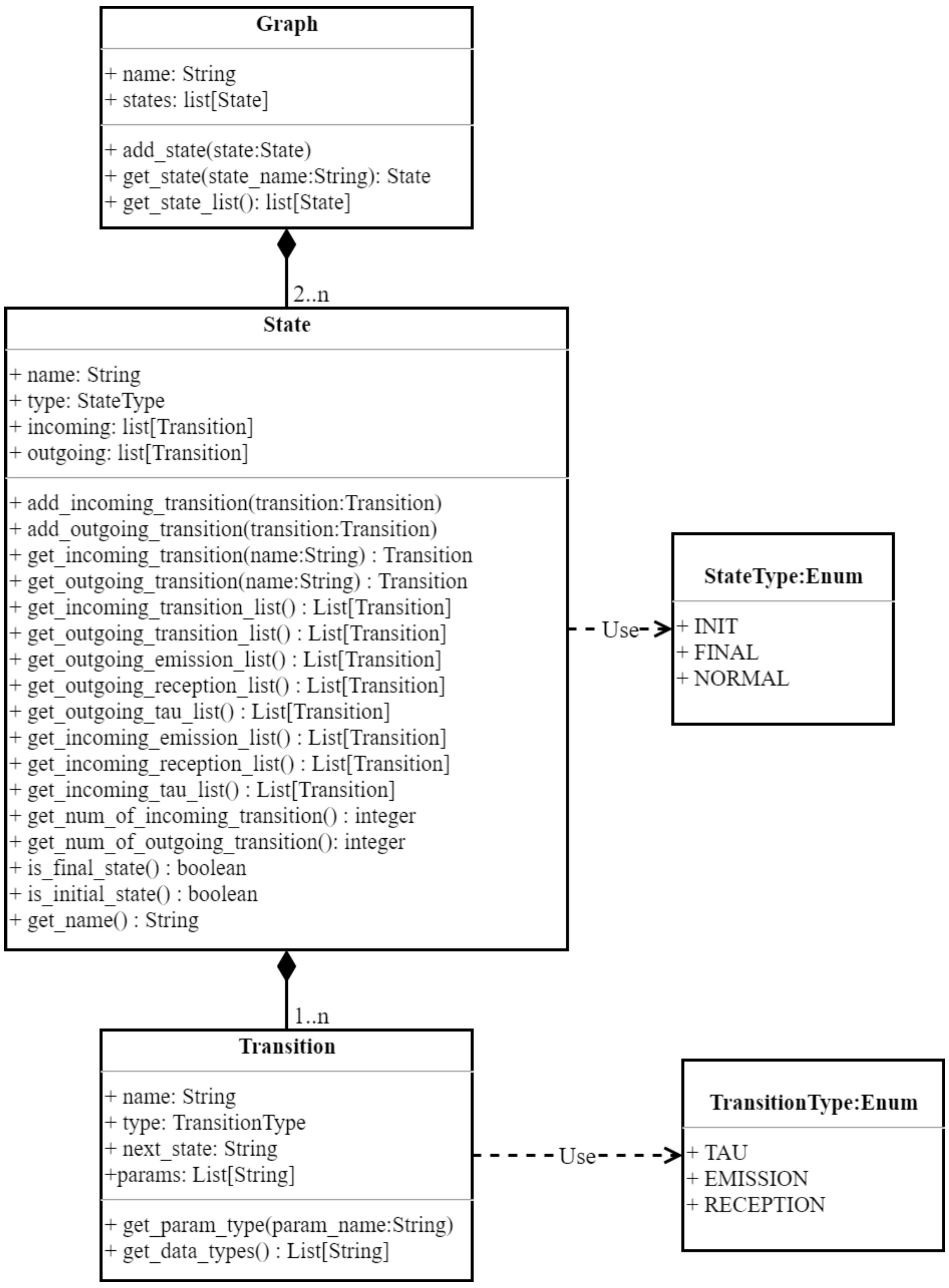

Graph
+ name: String
+ states: list[State]
+ add_state(state:State)
+ get_state(state_name:String): State
+ get_state_list(): list[State]
2..n
State
+ name: String
+ type: StateType
+ incoming: list[Transition]
+ outgoing: list[Transition]
+ add_incoming_transition(transition:Transition)
+ add_outgoing_transition(transition:Transition)
+ get_incoming_transition(name:String) : Transition
+ get_outgoing_transition(name:String) : Transition
+ get_incoming_transition_list() : List[Transition]
+ get_outgoing_transition_list() : List[Transition]
+ get_outgoing_emission_list() : List[Transition]
+ get_outgoing_reception_list() : List[Transition]
+ get_outgoing_tau_list() : List[Transition]
+ get_incoming_emission_list() : List[Transition]
+ get_incoming_reception_list() : List[Transition]
+ get_incoming_tau_list() : List[Transition]
+ get_num_of_incoming_transition() : integer
+ get_num_of_outgoing_transition(): integer
+ is_final_state() : boolean
+ is_initial_state() : boolean
+ get_name() : String
Use
StateType:Enum
+ INIT
+ FINAL
+ NORMAL
1..n
Transition
+ name: String
+ type: TransitionType
+ next_state: String
+params: List[String]
+ get_param_type(param_name:String)
+ get_data_types() : List[String]
Use
TransitionType:Enum
+ TAU
+ EMISSION
+ RECEPTION

Figure 35: Class diagram of the Graph, State, and Transition

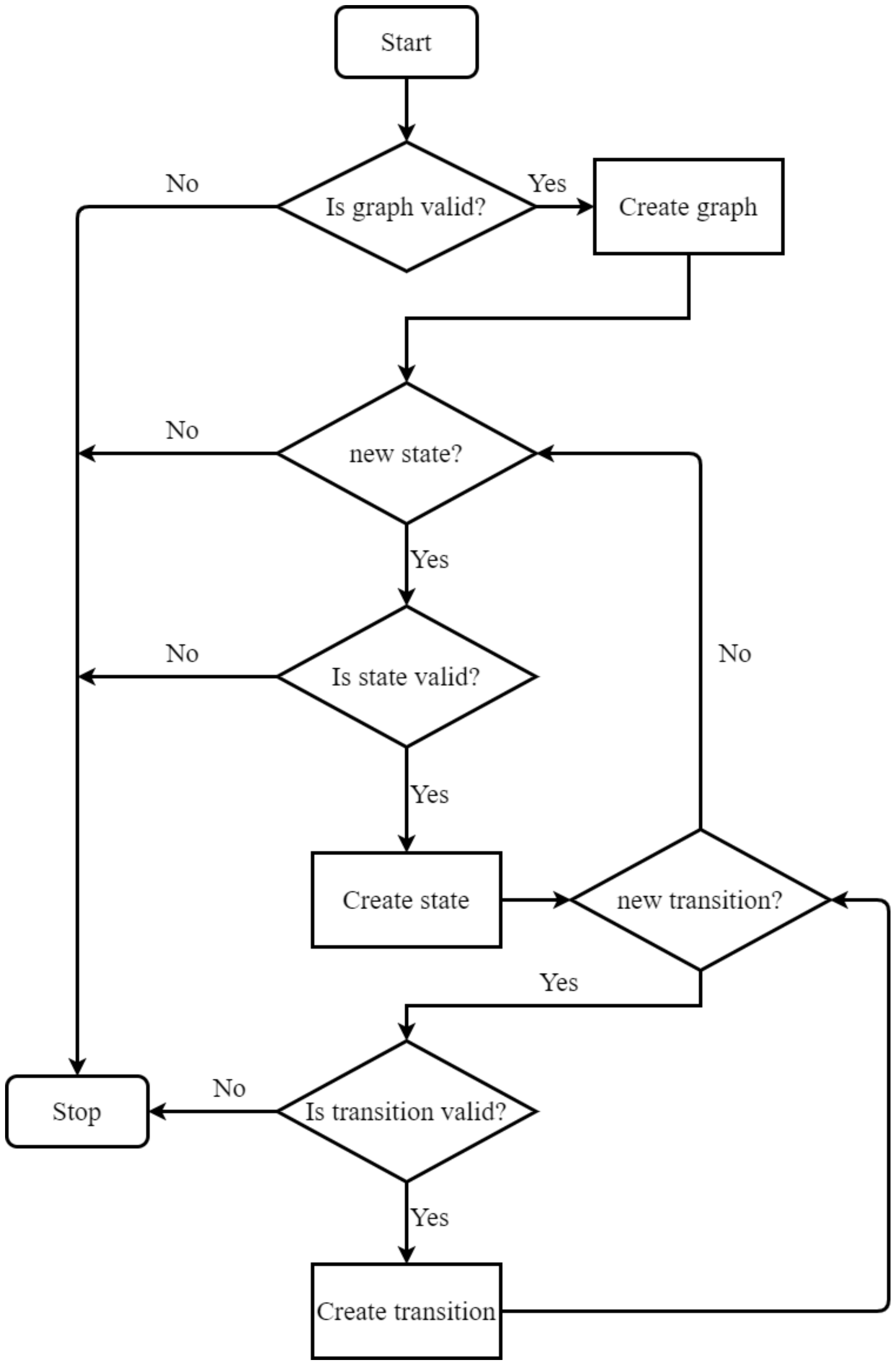


Figure 36: Flow chart of the Parser of the Compatibility calculation tool