

A Cyber-Physical Systems-Based Approach for Intelligent Energy Resource Management in Electric Vehicle Charging Network

Ciccotelli Gabriele⁺, Iuliano Roberto^{*},

Abstract—In a context where electric vehicle (EV) adoption is rapidly increasing, the need for efficient energy resource management has become crucial. This study proposes a solution to optimise power delivery at EV charging stations, based on the integration of leader election algorithms and distributed computing. The solution developed and presented in this work represents a concrete application of the fundamental concepts in the modelling and analysis of cyber-physical systems (CPS) to enhance the efficiency of charging infrastructures, with potential benefits for sustainability and intelligent grid management.

I. INTRODUCTION

The growing adoption of electric vehicles (EVs) presents new challenges for energy resource management, particularly in handling power flows to charging stations. It is essential to ensure that power is distributed efficiently and proportionally among the various stations to avoid overloads and maintain acceptable charging times for all users.

To address this problem, several solutions have been explored that leverage the potential of cyber-physical systems—systems combining physical processes and computational elements to solve complex problems. In this context, an approach based on leader-election and distributed computing algorithms is proposed to optimise power distribution among charging stations.

The proposed method utilises a particular leader-election algorithm known as the Bully algorithm. The purpose of this algorithm is to identify the station with the highest unique identifier (UID) and designate it as the leader. Subsequently, based on this election, numerical values are assigned to each charging point, which are then employed by the Push-Sum algorithm to calculate the power share to be delivered from the generator at any given time.

This document will provide a detailed analysis of the proposed solution's implementation, aiming to ensure fair and dynamic energy management across EV charging networks.

II. SCENARIO OF INTEREST

Consider a scenario featuring a network of n charging points (CP) for electric vehicles, all powered by a common generator. Each CP can only connect to one EV at a time, allowing the EV to charge its battery by drawing power from the generator.

This system can be framed as a cyber-physical system (CPS), as well as a real-time system that must respond to external stimuli within specified time constraints.

Similarly, each charging point can be viewed as an individual CPS, characterised by a network module enabling it to communicate through message exchange with other charging points in the system. This communication method allows the CPs to coordinate and calculate the power quotas from the shared generator within a certain time limit. At any given moment, each CP's power quota must match the ratio of the generator's total output to the number of CPs currently connected to vehicles requesting power.

To implement this solution, two algorithms commonly referenced in distributed computing literature were selected:

- The Bully Algorithm: This algorithm is typically used for leader election within a network of computational nodes;
- The Push-Sum Algorithm: Usually employed for computing aggregates, such as averages, in a distributed system.

The Push-Sum algorithm was selected to calculate the distributed power quotas. Specifically, this algorithm calculates the proportion of total available power each CP can receive based on its status and the total number of active CPs at that time. For this, the network must start from a state where only one active CP node has a numerical value of 1, while all others are at zero. The Push-Sum algorithm then averages these values, and this calculation is repeated each time there's a state change in the network (e.g., a vehicle connecting or disconnecting, or a battery reaching full charge).

To establish this initial condition, the Bully algorithm was used to assign the leader node in the network a value of 1, leaving the other nodes at zero.

⁺Student in the *Analysis and Control of Cyber-Physical Systems* course, taught by Prof. L. Iannelli, DING, University of Sannio, email: gabrieleciccotelli98@gmail.com

^{*}Student in the *Analysis and Control of Cyber-Physical Systems* course, taught by Prof. L. Iannelli, DING, University of Sannio, email: r.iuliano@studenti.unisannio.it

A. Requirements Elicitation

It is necessary to identify a system that efficiently manages the distribution of power among various charging points for electric vehicles, powered by a common generator. The system must satisfy the following constraints:

- **Charging Point Management:** The system must support a network of charging points, each capable of serving a single electric vehicle at a time. Only active CPs, meaning those connected to a vehicle that requires power, should draw energy from the network.
- **Power Distribution:** The generator's power should be evenly divided among the CPs connected to a vehicle that requires power. At any moment, the power supplied to each of these CPs should equal the total available power divided by the number of CPs connected to a vehicle being charged.
- **Coordination Between Charging Points:** Whenever there is a state change in the network, active CPs must communicate to automatically recalculate the power quotas, adjusting to the new conditions.
- **Charging Conditions:**
 - A vehicle may only receive power if it is connected to a charging point.
 - A vehicle with a fully charged battery should no longer receive power.
 - If a vehicle's battery reaches full charge (SoC - State of Charge), no power should be supplied to it until the SoC drops below 95%.
 - A CP that is out of service should not supply power.
- **Total Power Limitation:** At any given time, the sum of the power supplied by all active charging points should never exceed the total power provided by the generator.

B. Informal Requirements Definition

From the previous analysis, the following functional requirements are derived and described in natural language:

- R1: An electric vehicle cannot request power unless it is connected to a charging point.
- R2: If a vehicle's battery is fully charged, it cannot request power.
- R3: If a vehicle's battery level is between 95% and 100% and was previously at 100%, it cannot request power until the State of Charge (SoC) drops below 95%.
- R4: The charging point can supply power only if a vehicle is connected.
- R5: The charging point cannot supply power if it is out of service.
- R6: The charging point supplies power to the vehicle only if the vehicle's battery is not fully charged.
- R7: At any moment, the sum of the power supplied by the charging points must be greater than or equal to zero and should never exceed the total power supplied by the generator.
- R8: At any moment, the sum of the power supplied by charging points without connected vehicles must be zero.

- R9: Within 4 seconds of the last change in a vehicle's power request, each active and functioning CP must supply power equal to the generator's output, divided by the number of active charging points.

III. IMPLEMENTATION OF THE SOLUTION

The proposed solution is centred on the following mechanisms:

- Each CP (or node) is associated with a UID and an initial numerical value v_i , set to zero, which will later be used to calculate power quotas through the Push-Sum algorithm:

$$\forall cp_i \in \mathcal{CP}, \exists (UID_i \in \mathbb{N}, v_i \in \mathbb{R}) \quad (1)$$

- Nodes connected to a vehicle requiring power form the set of active CPs, denoted as CPs_{active} .
- Only nodes within CPs_{active} participate in the leader election process, managed as follows:
 - The Bully algorithm is used to identify the leader node, $cp_{leader} \in CPs_{active}$, which is the node with the highest UID. This process also sets the leader node's variable $v = 1$, while other nodes remain at zero.
 - The leader election process is initiated whenever a network state change occurs due to any of the following events:
 - * An EV disconnects from the CP to which it was connected.
 - * An EV connects to an available CP and requests power.
 - * The battery of a connected EV reaches full charge.
 - Immediately after any of the above events, all v_i variables are reset to zero.

From these points, it is clear that the Bully algorithm is employed to ensure that, following any of the conditions listed above, only one node in CPs_{active} will have $v = 1$ within a specific time frame, designating it as the leader. Additionally, it is evident that:

- If $CPs_{active} = \emptyset$, the election process is not initiated, and all v_i variables remain zero.

This process continuously executes the Push-Sum algorithm, which is used to calculate the power quotas assigned to each $cp_i \in CPs_{active}$. Thus, this algorithm enables the nodes in the network to reach consensus. Once consensus is achieved, each EV connected to a CP can draw power from the charging point, provided the CP is not in an error state.

The mechanisms outlined here illustrate the goal of creating a system that avoids overloads and ensures a fair distribution of power to vehicles that request it.

IV. SYSTEM MODEL

Based on the scenario and approach described in previous sections, a system model was created. A model is a representation of the system, useful for evaluating its properties. It is a good abstraction of the system if it omits only non-essential details.

Considering that the identified system combines both continuous and discrete dynamics, it is classified as a hybrid (or modal) system. This classification highlights the coexistence of discrete and continuous characteristics: the system is defined by discrete states, called “modes,” and primarily continuous variables, which determine the system’s evolution. The continuous aspect represents physical processes, such as battery charging and discharging by the EVs and power delivery by the CPs. However, events that characterise the system and cause state transitions, such as the connection or disconnection of vehicles, are discrete and can be described with a state machine.

Given this, modal machines were utilised to model this hybrid system. Consequently, it is apparent that the system under consideration is a hybrid event-triggered system.

The modelling process followed a well-established engineering principle: complex systems can be described as compositions of simpler subsystems. Therefore, the first phase of this process involved decomposing the domain into multiple subsystems and modelling them accordingly.

A. Battery Model: Coulomb Counting Method

The Coulomb Counting Method, also known as the Ampere-hour (Ah) Counting Method, is a technique commonly used to estimate a battery’s State of Charge (SoC), which represents the amount of charge entering and exiting the battery over time. However, it is not one of the most accurate models, as several factors, such as temperature and current leakage, impact its accuracy.

Given equation (3), by differentiating both sides with respect to time, equation (4) is obtained:

$$\text{SoC}(t) = \text{SoC}(t_0) + \frac{1}{C_{\text{rated}}} \int_{t_0}^t I(t) dt \quad (2)$$

$$\text{SoC}(t) - \text{SoC}(t_0) = \frac{1}{C_{\text{rated}}} \int_{t_0}^t I(t) dt \quad (3)$$

$$\frac{d\text{SoC}(t)}{dt} = \frac{1}{C_{\text{rated}}} I(t) \quad (4)$$

Where:

- $\text{SoC}(t_0)$: Initial State of Charge
- C_{rated} [Ah]: Battery capacity
- $I(t)$ [A]: Charge/discharge current at time t

Considering the following dimensionless constants:

- η_{ch} : Battery charging efficiency
- η_{dis} : Battery discharging efficiency

The equations that describe the charging and discharging processes are as follows:

$$\frac{d\text{SoC}(t)}{dt} = \frac{\eta_{\text{ch}}}{C_{\text{rated}}} I_{\text{ch}}(t) \quad \text{con} \quad I_{\text{ch}}(t) > 0 \quad (5)$$

$$\frac{d\text{SoC}(t)}{dt} = \frac{1}{C_{\text{rated}} \cdot \eta_{\text{dis}}} I_{\text{dis}}(t) \quad \text{con} \quad I_{\text{dis}}(t) < 0 \quad (6)$$

To express the battery capacity C_{rated} in Wh, the numerator must contain power in W. Thus, the final equations for charging and discharging are respectively:

$$\frac{d\text{SoC}(t)}{dt} = \frac{I_{\text{ch}}(t) \cdot V_{\text{nom}} \cdot \eta_{\text{ch}}}{C_{\text{rated}}} \quad \text{con} \quad I_{\text{ch}}(t) > 0 \quad (7)$$

$$\frac{d\text{SoC}(t)}{dt} = \frac{I_{\text{dis}}(t) \cdot V_{\text{nom}}}{C_{\text{rated}} \cdot \eta_{\text{dis}}} \quad \text{con} \quad I_{\text{dis}}(t) < 0 \quad (8)$$

The differential equations were modelled in Simulink using gain and integrator blocks. Both equations use a nominal voltage value of $V_{\text{nom}} = 230$ [V] and an efficiency constant value of $\eta_{\text{ch}} = \eta_{\text{dis}} = \frac{9}{10}$ [3].

Since this model only represents charging and discharging processes when the vehicle is stationary and connected to the charging point, a discharge current value of $I_{\text{dis}} = -5$ [A] was selected. The models are shown in Fig. 1 and Fig. 2.

B. Charging Point Model

After identifying the key properties of the charging point system from the scenario, the system was represented using a modal machine, specifically a non-deterministic finite state machine (FSM).

1) *System States*: From the analysis, the following main operational states for a CP were identified:

- Normal State: The system remains in this state when no faults are present, meaning no conditions occur that would transition the system into an error state.
- Error State: This state represents a malfunction condition in which the system is out of service.

Further refining the normal operational state, additional sub-states were identified for modelling purposes:

- Idle State: The CP is not connected to any vehicle. It remains available for connection by any EV at any time.
- Connected State: An EV is connected to the charging point.

It was also observed that when an EV is connected to the CP, it may or may not be drawing power from the CP. Specifically, a vehicle with a fully charged battery ($\text{SoC} = 1$) may remain connected to the CP until the user disconnects it. Consequently, the “connected” state was further refined into the following sub-states:

- Powering State: The CP is supplying power to the connected vehicle.
- NotPowering State: The CP is not supplying power to the connected vehicle, such as when the vehicle is fully charged but remains connected.

The hierarchical structure of these states indicates that the system can be modelled as a hierarchical state machine, utilising the concept of state refinement.

The actor model of this system and the corresponding state machine are shown in Fig. 3. The implementation in Simulink Stateflow is shown in Fig. 5.

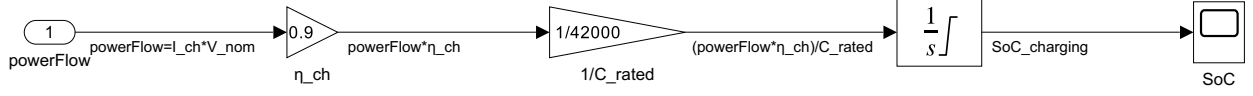


Fig. 1: Simulink diagram of the battery model Coulomb Counting Method (charging phase).



Fig. 2: Simulink diagram of the battery model Coulomb Counting Method (discharge phase).

2) *Signals and State Transitions*: Based on the actor model of the charging point system depicted in Figure 3, the defining signals of the system are as follows:

- Signal *pReq*: This is a continuous-time input signal.

$$pReq \in \mathbb{R}^+$$

It represents the vehicle's power request, with a value of 1 when the vehicle requests power from the CP and 0 otherwise.

- Signal *plug*: This is a continuous-time input signal.

$$plug \in \mathbb{R}^+$$

This signal represents the connection of a vehicle to the charging point, taking a value of 1 when a vehicle is connected and 0 otherwise.

- Signal *enablePowerFlow*: This is a continuous-time output signal.

$$enablePowerFlow \in \mathbb{R}^+$$

This signal represents the authorisation for the CP to deliver power to the connected vehicle. It takes a value of 1 when power delivery is authorised, and 0 otherwise. This signal is then used as a control signal to authorise the supply of a power quota, calculated by the component using the Push-Sum algorithm.

The overall system operates as a function that transforms the two input signals, so it can be defined as:

$$ChargingPoint : (\mathbb{R}^+)^2 \rightarrow \mathbb{R}^+$$

State transitions are driven by the input signals, while the output signal varies depending on the system's current mode. Details of these transitions and modes are provided in fig. 3 e 5.

3) *Non-Deterministic Behaviour*: In the charging point model, non-determinism is introduced by the possibility of "non-deterministic" transitions from the Normal state to the error state and vice versa. In other words, the vehicle can either remain operational in the Normal state or transition, unpredictably, to the Error state.

As shown in Figure 3, the state machine of the system is non-deterministic. For both the Normal and Error states, there are distinct transitions with guards that can both evaluate to true simultaneously. In the diagram in

Figure 3, the transitions that introduce non-determinism are highlighted in red.

This behaviour effectively results in the presence of a self-transition with a true guard on the Normal state, as well as a transition—also with a true guard—enabling the system to move from the Normal operating state to the Error state. The same logic applies to the Error state.

During simulation in Simulink Stateflow, the condition associated with the transition between Normal and Error was made probabilistic. From Figure 5, it can be observed that the transition from Normal to Error has the following guard:

```
[runPythonScriptRandGen <= (1 - exp(-(elapsed(sec))/300))]
```

Listing 1: Guard for the transition from the Normal to Error state in the charging point model implementation.

From the guard defined in code snippet 1, the following can be deduced:

- Invoking the `runPythonScriptRandGen()` function runs a Python script that generates a random value between 0 and 1 (inclusive) with a uniform probability density function (PDF). This means that every possible number in the $[0, 1]$ range is equally likely to occur. This external function was used because MATLAB's tools do not allow random values to include both bounds of the range;
- The value obtained from `runPythonScriptRandGen()` is compared to the output of function 9. The `elapsed(sec)` function returns the time, in seconds, since the last entry into the Normal state.

$$y = 1 - e^{\frac{-elapsed(sec)}{300}} \quad (9)$$

As shown in Figure 4, varying the denominator in function 9 alters the rate at which the function approaches the limit value of 1. For simulation purposes, the denominator was set to 300;

- The design of this guard ensures that the longer the system remains in the Normal state, the higher the probability of transitioning to the Error state.

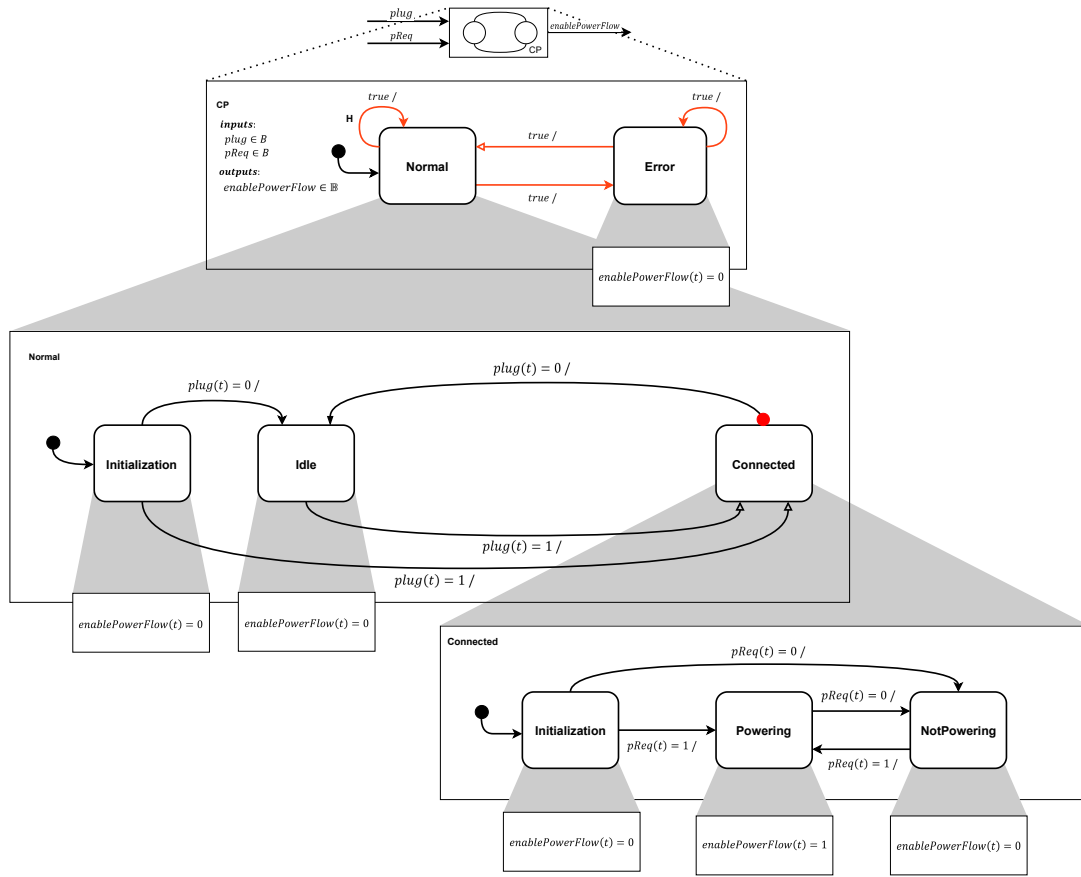


Fig. 3: Model of the charging point system. Above is the actor model. Below is the hierarchical finite-state machine and its mode refinements. Transitions that make the machine non-deterministic are shown in red. The self transition on the Normal state represents a historical transition (labelled H). In the state refinement of the Normal state, the transition from the Connected state to the Idle state is a preemptive transition.

4) *Semantics of Hierarchical Composition, Equivalent Machine, and Order Reduction:* Regarding the hierarchical composition shown in Figure 3, the following points are highlighted:

- The refinement of the Normal state follows a depth-first semantic approach, meaning it reacts before its container state;
- For the refinement of the Connected state, a different semantic is applied. Specifically, the outgoing transition from this state is a preemptive transition. The guard of the preemptive transition is evaluated before the refinement reacts. If the guard evaluates as true, the refinement does not react. This semantic gives priority to variations in the $plug$ signal over the $pReq$ signal;
- The self-transition on the Normal state is a historical transition. This means that when the transition is enabled, the target refinement resumes from the state it was in previously (or from its initial state upon first entry);
- The following transitions:
 - Error \rightarrow Normal;
 - Idle \rightarrow Connected;
 - (Normal, Initialization) \rightarrow Connected;

are represented by hollow arrows, as they are reset transitions. When traversed, the target refinement resets to its initial state.

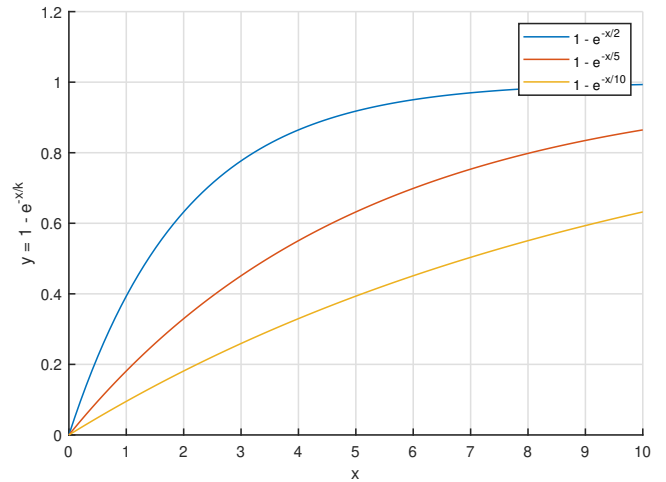


Fig. 4: Graph of the exponential function $y = 1 - e^{-x/k}$ for various k values: $k = 2$, $k = 5$, and $k = 10$. The curves demonstrate how the parameter k affects the rate at which the function approaches the limit value of 1.

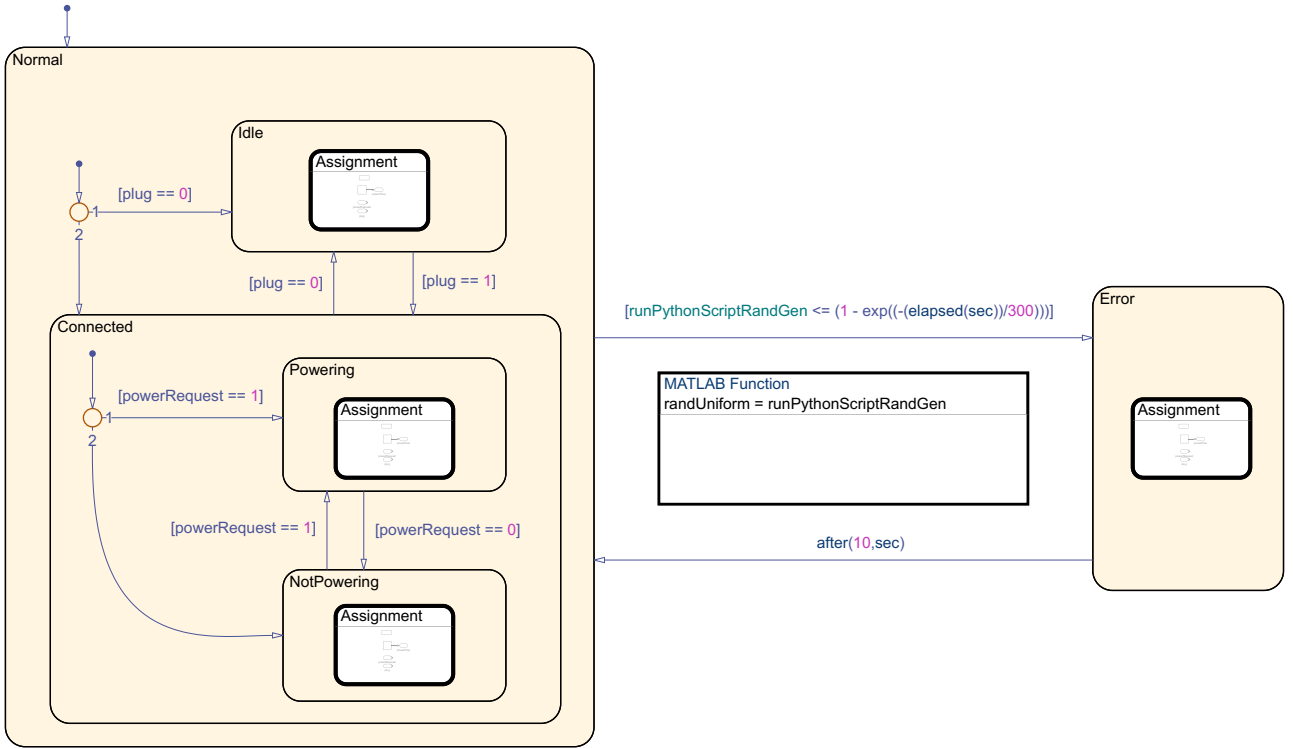


Fig. 5: Implementation of the charging point system model in Simulink & Stateflow.

The meaning of this hierarchy can be better understood by comparing the hierarchical state machine in Figure 3 with the equivalent "flattened" state machine shown in Figure 6a.

Since the Initialization state of the machine shown in Figure 6a does not possess state information or influence the outputs, an order reduction has been performed. The Initialization state collapses into the Idle state. The machine obtained after the order reduction is shown in Figure 6b.

In the implementation shown in Figure 5, it is evident that the (Normal, Initialization) state from the equivalent machine developed during the modelling phase (Figure 6) collapses into the connective junction block of the Normal state (Figure 5). Similarly, the (Normal, Connected, Initialization) state (Figure 6a) was implemented using the junction point of the Connected state.

The Stateflow tool provides solutions such as the one mentioned above, which help manage system complexity by avoiding the creation of initialisation states as initially envisioned in the model.

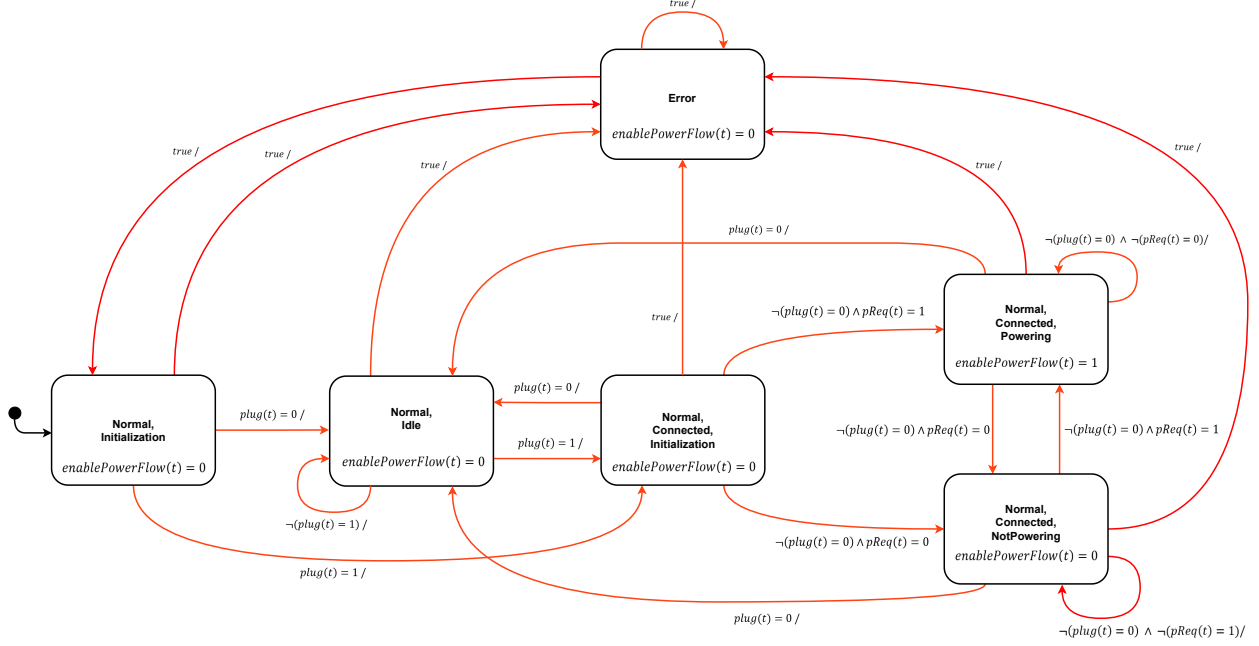
More precisely, through this "decision point", it is possible to evaluate which refinement state to reach immediately upon entering the container state. For instance, in Figure 5, based on the assigned priorities, the guard of the transition to the Idle state is evaluated first. If the guard ($plug == 0$) is verified, the transition is enabled, and the machine moves to the Idle state. In all other cases, the transition to the Connected state is enabled.



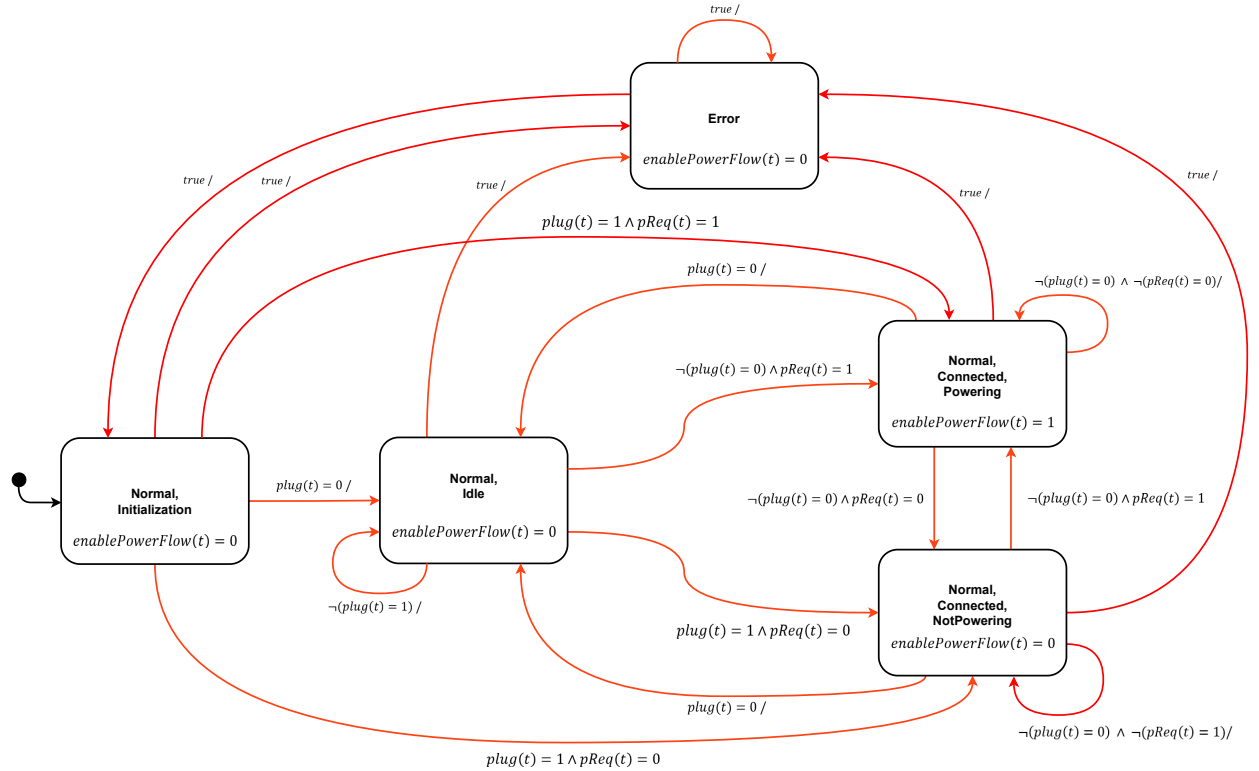
Fig. 7: Combining transitions and junctions to create branching paths.
Connective junction block.

5) *Characterisation of the Model*: Based on the way the charging point system model was designed, the state machine can be characterised as follows:

- **Event-triggered**: The system's dynamics (state evolution) are driven by events. The system reacts to external events such as a vehicle connecting or disconnecting from the charging point, as well as power requests from the vehicle itself;
- **Non-deterministic**: The update function is a set-valued function. For example, from the Normal state, given identical inputs, the system can either transition to the Error state or remain in the current state through a self-transition. A similar behaviour applies to the Error state;
- **A modal machine**: This is the model of a hybrid system, as the system combines:
 - **Discrete dynamics**: This relates to events that characterise the system and lead to state changes;
 - **Continuous dynamics**: This concerns continuous signals that govern the system's evolution. The system's input and output signals are continuous-time signals, meaning their evaluation and generation occur continuously over time. Therefore, the system operates in its current state in accordance with the active mode's specification.



(a) Equivalent FSM (flattened) of the point system model, shown in Fig. 3, obtained by solving the hierarchical composition. With the notation ‘Normal, Initialisation’, ‘Normal, Idle’, ‘Normal, Connected, Initialisation’, ‘Normal, Connected, Powering’ and ‘Normal, Connected, NotPowering’ indicate that the state machine is in the first indicated state and more precisely in the refinement state indicated in positions two and three (if present). For example, considering the state ‘Normal, Connected, Initialisation’ means to say that the state machine is in the Normal state, more precisely in the Connected state of its refinement, and in the Initialisation state of its further refinement.



(b) Equivalent FSM (flattened) of the point system model shown in fig. 3. The order reduction is performed on the equivalent machine shown in fig. 6a. Please refer to the caption of fig. 6a for the notation used in naming the states.

Fig. 6: Equivalent FSMs of the point of charge system model.
Regarding the characteristics of the input and output signals of the FSMs, please refer to fig. 3.

C. Electric Vehicle Model

When defining the electric vehicle (EV) model, the properties of interest within the analysed context were considered. The EV system has been represented using a modal machine, characterised by modes (i.e., discrete states, also referred to as bubbles) and continuous state variables.

1) *System States*: In the initial phase of model construction, the system states were identified. First, a normal operating state was defined, representing the standard operational condition of the vehicle. This was introduced to make the model flexible, allowing for the addition of an error state in future developments, which could be characterised based on specific context requirements.

Subsequently, the normal operating state was decomposed into:

- Unplugged: This state represents a condition where the EV is not connected to the CP;
- Plugged: This state represents a condition where the vehicle is connected to the charging point.

The Plugged state was further decomposed into:

- Completed: This state indicates the vehicle's battery is fully charged;
- Charging: This state indicates the vehicle's battery is in the charging phase.

Transitions between these states are governed by the battery's state of charge (SoC).

Next, a refinement of the Completed and Charging states was developed based on the Coulomb Counting Method battery model described in earlier chapters. Given the hierarchical nature of the states, the state machine representing the system can be classified as a hierarchical state machine.

The actor model of this system and its corresponding state machine are shown in Figure 8. The implementation in Simulink Stateflow is presented in Figure 9.

2) *Signals and State Transitions*: As shown in the actor model in Figure 8, the system is characterised by the following input and output signals:

- *pReq*: An output, continuous-time signal.

$$pReq \in \mathbb{B}^{\mathbb{R}^+} \quad (10)$$

- *plug*: An output, continuous-time signal.

$$plug \in \mathbb{B}^{\mathbb{R}^+} \quad (11)$$

- *pFlow*: An input, continuous-time signal representing the power flow into the vehicle.

$$pFlow \in \mathbb{R}^{\mathbb{R}^+} \quad (12)$$

Details of the *pReq* and *plug* signals are discussed in Section IV-B.2.

The system itself can be viewed as a function that transforms the input signal into the two output signals:

$$Electric\ Vehicle : \mathbb{R}^{\mathbb{R}^+} \rightarrow (\mathbb{B}^{\mathbb{R}^+})^2 \quad (13)$$

As described in the latter part of Chapter IV-B.2, the implementation of the models shown in Figure 11 also employs the connective junction block.

3) *Non-Deterministic Behaviour*: In the electric vehicle model, non-determinism is introduced by the unpredictability of the user's action to connect or disconnect the vehicle from a charging point.

In the state machine, this is reflected in the possibility of "non-deterministic" transitions between the Unplugged and Plugged states. In other words, at any given moment, the vehicle may either remain connected to the charging point (Plugged state) or, unpredictably, disconnect from it (Unplugged state). This makes the system model non-deterministic.

During simulation in Simulink Stateflow, the conditions associated with the transitions between Plugged and Unplugged states were evaluated probabilistically. From Figure 9, it can be observed that the transition from Unplugged to Plugged has the following guard:

```
[runPythonScriptRandGen <= (1 - exp(-(elapsed(sec))/10)))]
```

Listing 2: Guard for the transition from the Unplugged to Plugged state in the electric vehicle model implementation.

```
[(runPythonScriptRandGen <= SoC.EV) && (runPythonScriptRandGen <= (1 - exp(-(elapsed(sec))/100)))]
```

Listing 3: Guard for the transition from the Plugged to Unplugged state in the electric vehicle model implementation.

From code snippet 3, it can be observed that when the vehicle is connected to the charging point, the likelihood of it disconnecting depends on both the elapsed time since the connection and the battery's state of charge (SoC). The higher the SoC, the more likely the vehicle is to disconnect from the charging point, making way for another vehicle.

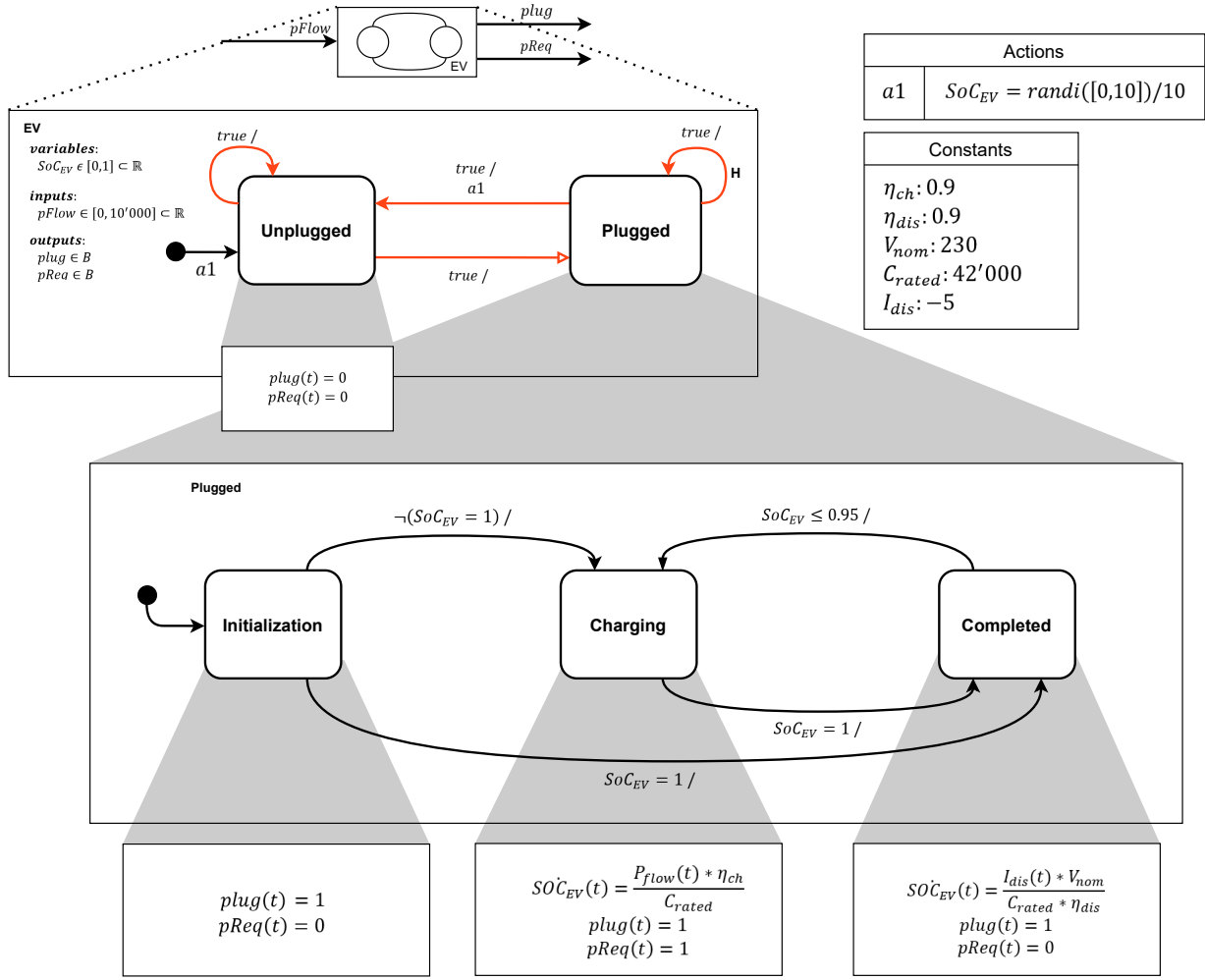


Fig. 8: Modal model of the electric vehicle system. The actor model of the system is shown above, while the hierarchical state machine and corresponding refinements of modes are depicted below. Transitions introducing non-determinism are highlighted in red. The self-transition on the Plugged state represents a historical transition (labelled H). The transition from Unplugged to Plugged is a reset transition.

Additionally, from Figure 9, it can be seen that when the transition from Plugged to Unplugged is triggered, it is as if a new vehicle is considered. In fact, the value of the SoC variable is reinitialised randomly.

Finally, it is worth noting that the probabilistic evaluation of these transitions is analogous to the process discussed in Chapter IV-B.3, concerning the charging point system model.

4) Regulating System Behaviour to Avoid Chattering:

In the EV system model, transitions between the Completed and Charging states are triggered based on the battery's state of charge (SoC). Without introducing a hysteresis band, an undesirable phenomenon can occur: when the battery's SoC reaches full charge in the Charging state, the system immediately transitions to the Completed state. However, if the SoC slightly decreases, the system rapidly switches back to the Charging state, resulting in continuous alternation between the two states. This phenomenon is known as chattering.

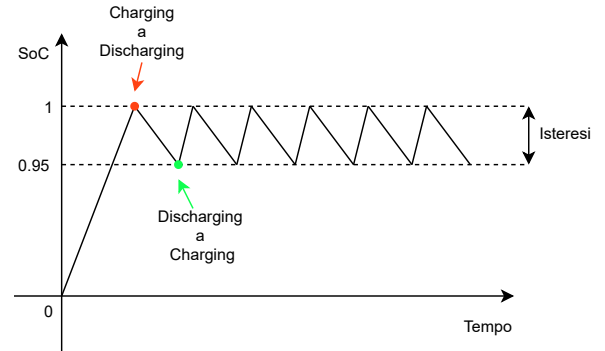


Fig. 10: Use of hysteresis to regulate the transition from the Completed to Charging state in the EV system model.

To prevent this undesirable behaviour, a hysteresis band was introduced to regulate state transitions. Specifically, the system remains in the Completed state when the battery's SoC is between 95% and 100%. Only when the SoC drops below 95% does the system transition from Completed to Charging. This hysteresis band ensures controlled transitions between the two states, eliminating the chattering phenomenon and improving the system's robustness.

Refer to Figure 10 for further illustration.

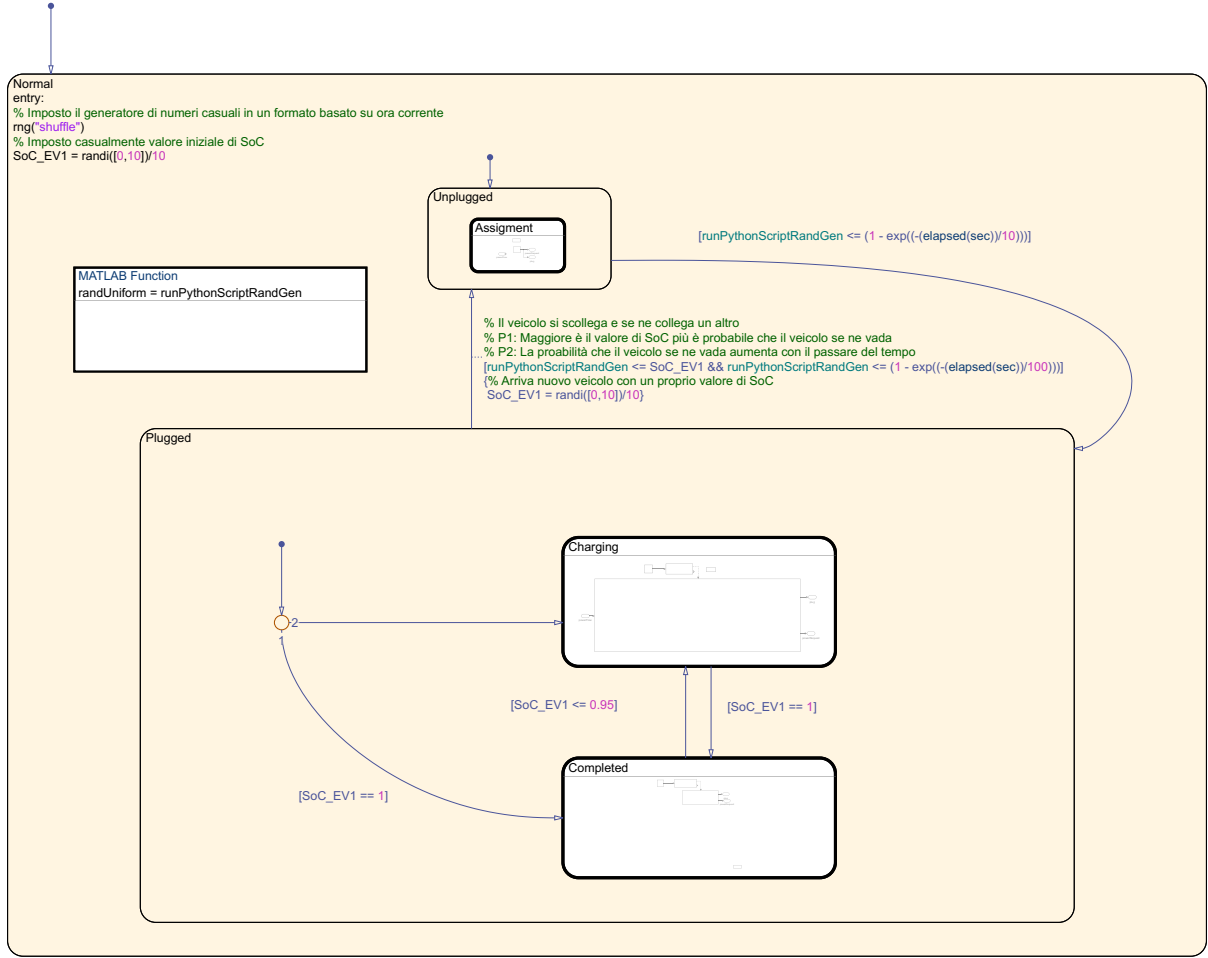


Fig. 9: Implementation of the electric vehicle system model in Simulink & Stateflow.

5) *Semantics of Hierarchical Composition, Equivalent Machine, and Order Reduction:* Regarding the hierarchical composition shown in Figure 8, the following points are highlighted:

- The refinement of the Plugged state follows a depth-first semantic approach, meaning it reacts before its container state;
- The self-transition on the Plugged state is a historical transition. This means that when the transition is enabled, the target refinement resumes from the state it was in previously;
- The transition from Unplugged to Plugged is represented by a hollow arrow, as it is a reset transition. When traversed, the target refinement resets to its initial state.

The significance of this hierarchy can be better understood by comparing the hierarchical state machine in Figure 8 with the equivalent "flattened" state machine shown in Figure 11a.

Since the Initialization state in the machine shown in Figure 11a does not contain state information or influence outputs, an order reduction was performed. The Initialization state collapses into the Unplugged state. The machine obtained after order reduction is shown in Figure 11b.

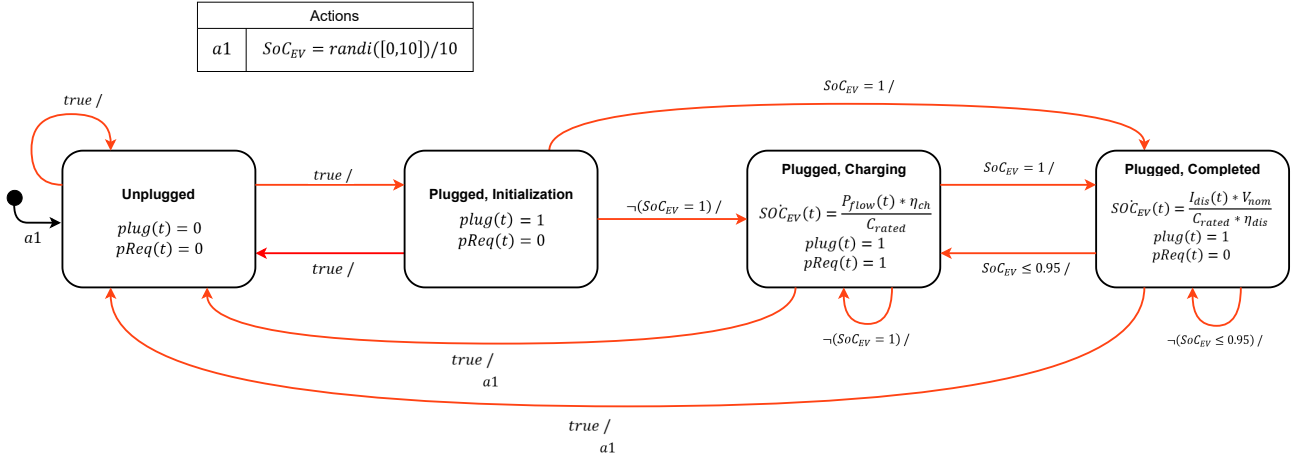
It should be noted that the resulting equivalent machine is a modal machine. Some states in the state machine

include both discrete state information (bubble) and the value of the continuous state variable SoC. In this case, there are $n = 3$ modes (bubbles) and a single continuous state variable SoC, which has a domain in the set of positive real numbers \mathbb{R}_+ (i.e., time) and a codomain in the subset $[0, 1] \subset \mathbb{R}$. Consequently, the variable can assume infinitely many possible values. Thus, the sets of modes and states of the machine are defined as follows:

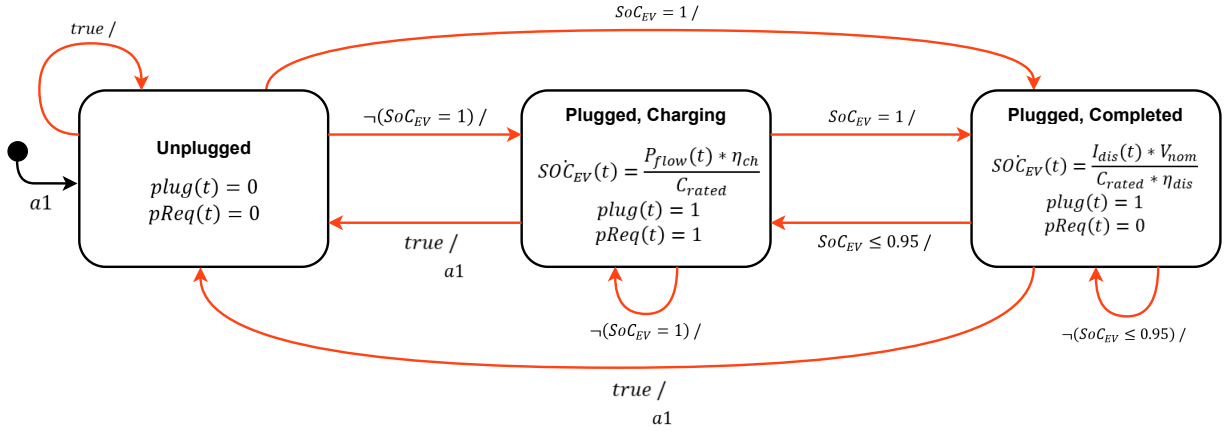
$$Modes = \left\{ \begin{array}{l} Unplugged, \\ (Plugged, Charging), \\ (Plugged, Completed) \end{array} \right\} \quad (14)$$

$$States = \left\{ \begin{array}{l} Unplugged, \\ (Plugged, Charging), \\ (Plugged, Completed) \end{array} \right\} \times \{[0, 1] \subset \mathbb{R}\} \quad (15)$$

This implies that $|States| = \infty$.



(a) Equivalent (flattened) state machine of the modal electric vehicle system model shown in Figure 8. The hierarchical composition is resolved. For notation regarding state names, refer to Figure 6a.



(b) Equivalent (flattened) state machine of the modal electric vehicle system model, obtained after order reduction. For notation regarding state names, refer to Figure 6a.

Fig. 11: Equivalent state machines of the modal electric vehicle system model. For input and output signal characteristics of the state machines, refer to Figure 8.

6) *Model Characterisation:* Based on the way the electric vehicle (EV) system model was designed, the state machine can be characterised as follows:

- **Event-triggered:** The dynamics of the system (state evolution) are driven by events. The system reacts to external events such as a vehicle connecting or disconnecting from a charging point, as well as changes in the battery's state of charge (SoC);
- **Non-deterministic:** The update function is a set-valued function. For instance, starting from the Unplugged state, the system can either transition to the Plugged state or remain in the current state through a self-transition. The same behaviour applies to the Plugged state;
- **A modal machine:** This represents a hybrid system, as the model integrates:
 - **Discrete dynamics:** These relate to events that characterise the system and cause state transitions;
 - **Continuous dynamics:** These involve continuous signals that govern the evolution of the system and the Coulomb Counting Method, which is

used to model the physical processes of charging and discharging the battery. Consequently, the evolution of the continuous state variable SoC is determined by the differential equations 7 and 8, as discussed in Section IV-A.

D. Composition of Electric Vehicle and Charging Point

The synchronous cascade composition of the EV and CP systems is shown in Figure 12a. This composition is achieved through the EV system's output signals, plug and pReq, which serve as inputs to the CP system. The two output ports of the EV system ($o1$ and $o2$) feed the input ports ($i1$ and $i2$) of the CP system. It is assumed that the data types of ports $o1$ and $o2$ are $V1$ and $V2$, respectively, and the types of input ports $i1$ and $i2$ are $V3$ and $V4$, respectively. Thus, the requirements for the validity of this composition are as follows:

$$V1 \subseteq V3 \quad (16)$$

$$V2 \subseteq V4 \quad (17)$$

In the specific case of the EV&CP state machine, both sets $V1$ and $V2$ coincide with the set of binary numbers \mathbb{B} . This ensures that each output produced by the electric vehicle system on ports $o1$ and $o2$ is accepted as input by the charging point system on ports $i1$ and $i2$. Based on this, it can be stated that there is a type check.

Although logically, the reaction of the CP system should occur after that of the EV system (as the CP system requires the outputs of the EV system as inputs), this composition is abstracted to a higher level where, in each reaction, both systems are treated as if they react instantaneously and simultaneously. This means the composite EV&CP system reacts as a single block, ensuring that, at each reaction, based on the input $pFlow$, the output $enablePowerFlow$ is immediately calculated. The entire process is visualised as if the reactions of both machines occur concurrently, without considering the internal sequentiality between EV and CP.

1) Synchrony and Reality: It is important to note that synchrony does not exist in reality. However, using synchronous models is advantageous because they provide a good representation of the system when delays, which are inevitably present in reality, do not affect the system's functionality. Thus, the results obtained would be the same as those observed if the systems were synchronous.

If, in reality, the delays lead to behaviours differing from synchronous ones, the synchronous computation model would no longer be appropriate. In such cases, asynchronous computation models must be introduced to account for the fact that the systems interact at different times.

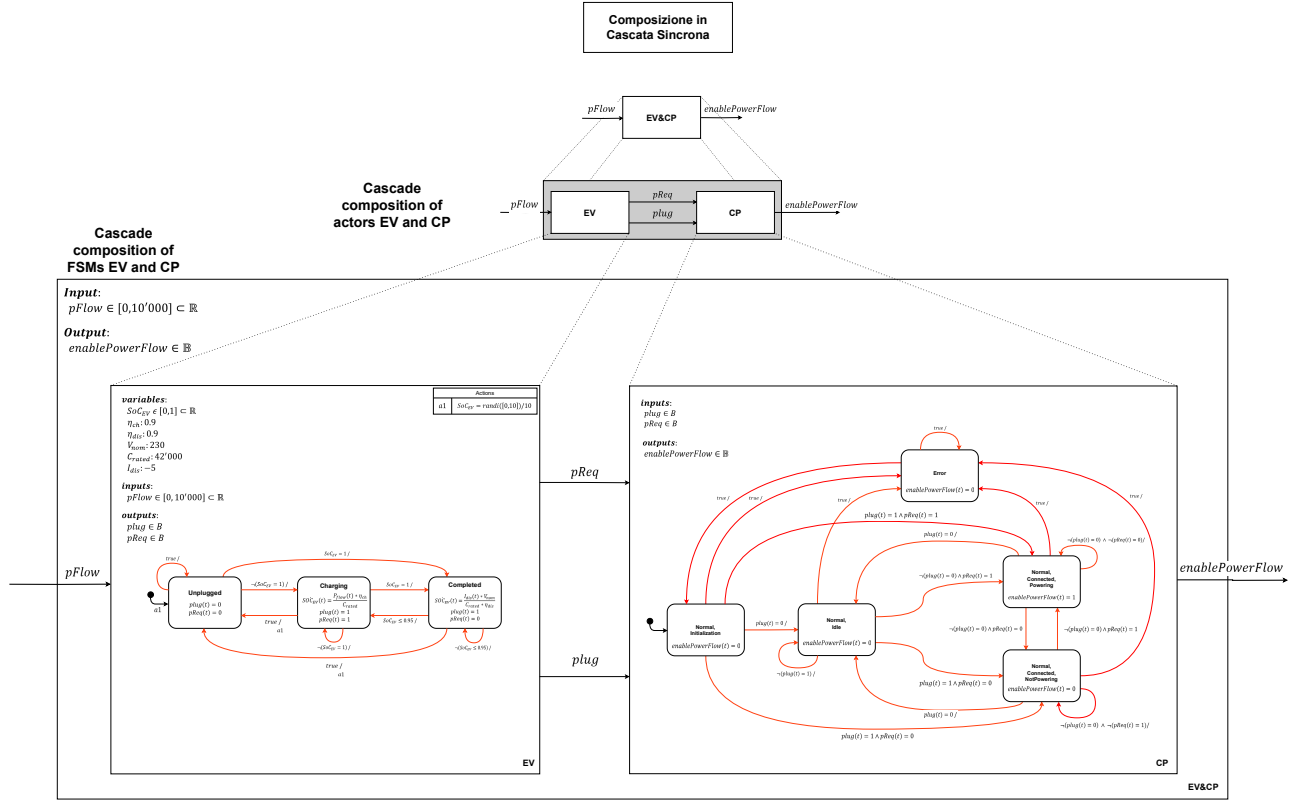
2) Equivalent State Machine of the Synchronous Cascade Composition: From the cascade composition of the two state machines, the equivalent state machine has been derived, represented graphically in Figure 13a and formally defined using the quintuple shown in Figure 14.

The state set of the equivalent machine corresponds to the Cartesian product of the state sets of the two machines. However, the EV state machine is characterised by an infinite number of possible states, and as a result, the state set of the equivalent machine will also have infinite cardinality. Among these, infinitely many states are due to the infinite values that the continuous state variable SoC can assume. Therefore, in the formal definition of the machine, only the states relevant to the analysis have been considered.

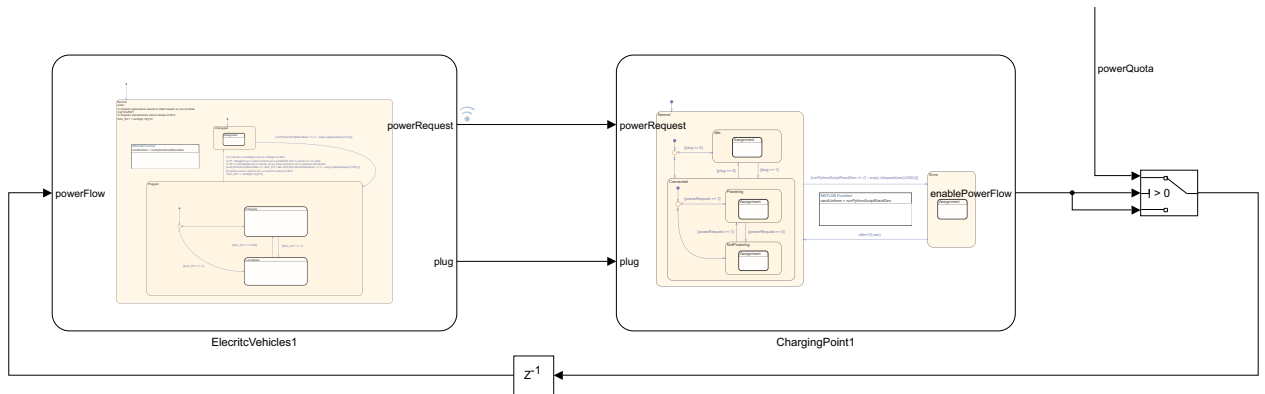
From the relevant states identified, the transitions that could be traversed were determined, excluding those that were unreachable. It is important to note that transitions occur simultaneously, even when one logically causes the other.

3) Implementation in Simulink Stateflow: In Figure 12b, the implementation of the model shown in Figure 12a within Simulink Stateflow is presented. It is important to note that the $enablePowerFlow$ signal is used to control a switch. The output signal from the switch, representing the power flow delivered by the considered charging point, passes through a delay block before being input into the EV system.

The presence of this delay block introduces a dynamic component into the system, thereby rendering it asynchronous.



(a) Model of the synchronous cascade composition of the electric vehicle and charging point systems.



(b) Implementation of the synchronous cascade composition in Simulink Stateflow.

Fig. 12: Cascade composition of EV&CP.

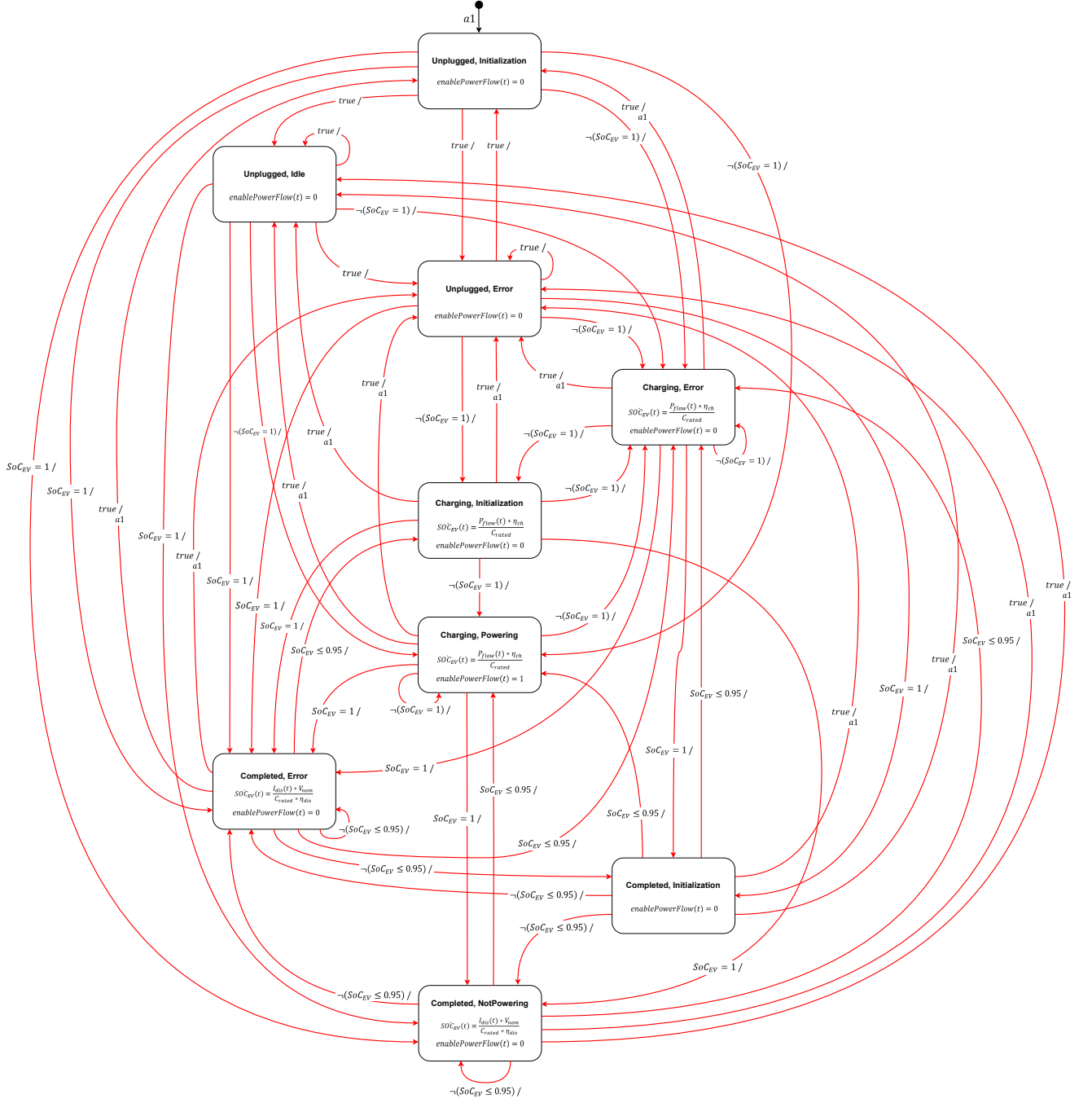


Fig. 13: Equivalent state machine of the synchronous cascade composition.
For input and output signal characteristics of the state machines, refer to Figure 12a.

- (a) Given the complexity of the graphical representation, the layout of the states and the representation of the corresponding transitions were optimised using the open-source tool Graphviz.

$$\begin{aligned}
\mathbf{States} &= \{(Unplugged, Initialization), (Unplugged, Idle), (Unplugged, Error), (Charging, Initialization), (Charging, Powering), (Charging, Error), \\
&\quad (Completed, Initialization), (Completed, NotPowering), (Completed, Error)\} \\
\mathbf{Inputs} &= \{pflow \in [0, 10'000] \subset \mathbb{R}\} \\
\mathbf{Outputs} &= \{enablePowerFlow \in \mathbb{B}\} \\
\mathbf{InitialState} &= \{(Unplugged, Initialization) \in \mathbf{States}\} \\
\mathbf{updateRelation}(s, i) &= \left\{ \begin{aligned}
&\{[(Unplugged, Idle), 0], [(Unplugged, Error), 0], [(Completed, Error), 0], [(Completed, NotPowering), 0]\} \\
&\quad \text{if } s = (Unplugged, Initialization) \wedge (SoC_{EV} = 1) \\
&\{[(Unplugged, Idle), 0], [(Unplugged, Error), 0], [(Charging, Error), 0], [(Charging, Powering), 1]\} \\
&\quad \text{if } s = (Unplugged, Initialization) \wedge \neg(SoC_{EV} = 1) \\
&\{[(Unplugged, Idle), 0], [(Unplugged, Error), 0], [(Completed, NotPowering), 0], [(Completed, Error), 0]\} \\
&\quad \text{if } s = (Unplugged, Idle) \wedge (SoC_{EV} = 1) \\
&\{[(Unplugged, Idle), 0], [(Unplugged, Error), 0], [(Charging, Powering), 1], [(Charging, Error), 0]\} \\
&\quad \text{if } s = (Unplugged, Idle) \wedge \neg(SoC_{EV} = 1) \\
&\{[(Unplugged, Error), 0], [(Unplugged, Initialization), 0], [(Completed, Error), 0], [(Completed, Initialization), 0]\} \\
&\quad \text{if } s = (Unplugged, Error) \wedge (SoC_{EV} = 1) \\
&\{[(Unplugged, Error), 0], [(Unplugged, Initialization), 0], [(Charging, Error), 0], [(Charging, Initialization), 0]\} \\
&\quad \text{if } s = (Unplugged, Error) \wedge \neg(SoC_{EV} = 1) \\
&\{[(Unplugged, Initialization), 0], a1\}, \{[(Unplugged, Error), 0], a1\}, [(Completed, Error), 0], [(Completed, Initialization), 0]\} \\
&\quad \text{if } s = (Charging, Error) \wedge (SoC_{EV} = 1) \\
&\{[(Unplugged, Initialization), 0], a1\}, \{[(Unplugged, Error), 0], a1\}, [(Charging, Initialization), 0], [(Charging, Error), 0]\} \\
&\quad \text{if } s = (Charging, Error) \wedge \neg(SoC_{EV} = 1) \\
&\{[(Unplugged, Error), 0], a1\}, \{[(Unplugged, Idle), 0], a1\}, [(Completed, Error), 0], [(Completed, NotPowering), 0]\} \\
&\quad \text{if } s = (Charging, Initialization) \wedge (SoC_{EV} = 1) \\
&\{[(Unplugged, Error), 0], a1\}, \{[(Unplugged, Idle), 0], a1\}, [(Charging, Powering), 1], [(Charging, Error), 0]\} \\
&\quad \text{if } s = (Charging, Initialization) \wedge \neg(SoC_{EV} = 1) \\
&\{[(Unplugged, Error), 0], a1\}, \{[(Unplugged, Idle), 0], a1\}, [(Completed, Error), 0], [(Completed, NotPowering), 0]\} \\
&\quad \text{if } s = (Charging, Powering) \wedge (SoC_{EV} = 1) \\
&\{[(Unplugged, Error), 0], a1\}, \{[(Unplugged, Idle), 0], a1\}, [(Charging, Powering), 1], [(Charging, Error), 0]\} \\
&\quad \text{if } s = (Charging, Powering) \wedge \neg(SoC_{EV} = 1) \\
&\{[(Unplugged, Error), 0], a1\}, \{[(Unplugged, Idle), 0], a1\}, [(Charging, Error), 0], [(Charging, Initialization), 0]\} \\
&\quad \text{if } s = (Completed, Error) \wedge (SoC_{EV} \leq 0.95) \\
&\{[(Unplugged, Error), 0], a1\}, \{[(Unplugged, Idle), 0], a1\}, [(Completed, Initialization), 0], [(Completed, Error), 0]\} \\
&\quad \text{if } s = (Completed, Error) \wedge \neg(SoC_{EV} \leq 0.95) \\
&\{[(Unplugged, Error), 0], a1\}, \{[(Unplugged, Idle), 0], a1\}, [(Charging, Error), 0], [(Charging, Powering), 1]\} \\
&\quad \text{if } s = (Completed, Initialization) \wedge (SoC_{EV} \leq 0.95) \\
&\{[(Unplugged, Error), 0], a1\}, \{[(Unplugged, Idle), 0], a1\}, [(Completed, Error), 0], [(Completed, NotPowering), 0]\} \\
&\quad \text{if } s = (Completed, Error) \wedge \neg(SoC_{EV} \leq 0.95) \\
&\{[(Unplugged, Error), 0], a1\}, \{[(Unplugged, Idle), 0], a1\}, [(Charging, Error), 0], [(Charging, Powering), 1]\} \\
&\quad \text{if } s = (Completed, NotPowering) \wedge (SoC_{EV} \leq 0.95) \\
&\{[(Unplugged, Error), 0], a1\}, \{[(Unplugged, Idle), 0], a1\}, [(Completed, Error), 0], [(Completed, NotPowering), 0]\} \\
&\quad \text{if } s = (Completed, NotPowering) \wedge \neg(SoC_{EV} \leq 0.95)
\end{aligned} \right.
\end{aligned}$$

Fig. 14: Formal definition of the equivalent machine relative to the synchronous cascade composition, realised considering only a relevant subset of the states. In this regard, refer to the sections [IV-C.5](#) and [IV-D.2](#)

E. Formal Definition of Specifications: Application of Signal Temporal Logic

Based on the requirements identified in Section II-B, and given that the specifications involve the values of continuous signals, the following specifications have been defined using the formalism of Signal Temporal Logic (STL):

$$\begin{aligned}
\varphi_1 &= G \{ [plug(t) = 0] \implies [pReq(t) = 0] \} \\
\varphi_2 &= G \{ [SoC(t) = 1] \implies [pReq(t) = 0] \} \\
\varphi_3 &= G \{ [(0.95 \leq SoC(t) \leq 1) \mathbf{S}(SoC(t^*) = 1)] \implies (pReq(t) = 0) \} \mid t^* < t \\
\varphi_4 &= G \{ [enablePowerFlow(t) = 1] \implies [plug(t) = 1] \} \\
\varphi_5 &= G \{ [errorState(t) = 1] \implies [enablePowerFlow(t) = 0] \} \\
\varphi_6 &= G \{ [enablePowerFlow(t) = 1] \implies [SoC(t) \neq 1] \} \\
\varphi_7 &= G \{ 0 \leq \left[\sum_{i=1}^n pFlow_i(t) \right] \leq generatorPower(t) \}, \quad n = \text{number of CPs} \\
\varphi_8 &= G \left\{ \left[\sum_{i=1}^{n_{idle}} pFlow_i(t) \right] = 0 \right\}, \quad n_{idle} = \text{number of CPs in Idle state} \\
\varphi_9 &= \bigvee_{i=1}^n \left[pReq_i^{actual}(t) \neq pReq_i^{previous}(t) \right] \implies F_{[0,4]} \left\{ \bigwedge_{i=1}^n \left[pFlow_i(t) = \frac{generatorPower(t) \cdot poweringState_{CP_i}(t)}{n_{CP_active}} \right] \right\}
\end{aligned}$$

Moreover, as evidenced by the update function shown in Fig. 14, the composition model (EV&CP) is non-deterministic. In fact, the update function is a set-valued function (set valued).

For example, starting from the state (*Unplugged, Initialization*), upon the occurrence of the condition $SoC = 1$, there is an equal probability of transitioning to any of the following states:

- (*Unplugged, Idle*);
- (*Unplugged, Error*);
- (*Completed, Error*);
- (*Completed, NotPowering*).

Thus, the system's execution trace is not linear but has a tree structure (computation tree). This constitutes another reason why Linear Temporal Logic (LTL) could not have been used in this case.

F. Communication or Coordination Between System Components: Application of the Dataflow Model of Computation

This section describes how the charging points communicate through message exchange (or tokens) and coordinate their actions to meet the system's operational requirements. Specifically, in the considered system, communication between the CPs is necessary for the following purposes:

- To identify a leader;
- To achieve consensus.

The data flow among the CPs can be modelled using the Dataflow Model of Computation (MoC). Indeed, each charging point can be regarded as an actor that processes the received tokens and reacts based on the defined logic.

In this context, it is worth noting that the part of the system responsible for calculating power allocations,

which utilises the Push-Sum algorithm, has been modelled as shown in Fig. 15. Subsequently, the aforementioned model was implemented in Simulink using tools provided by the SimEvents library.

1) *Simulink SimEvents Library*: Initially, during the implementation of the model, the possibility of using the Dataflow Domain in Simulink was considered. However, due to the limited flexibility of this tool, the SimEvents library was chosen instead.

The SimEvents library provides a set of blocks that are particularly suited for implementing processes involving message exchange. These include:

- The Entity block, which enables the generation of events marked by a numerical value. For an event to be produced, this block must be appropriately triggered. In the considered scenario, the block generates a message;
- The Entity Multicast block, which facilitates notifying a group of recipients about the occurrence of an event. This component is responsible for sending the token;
- The Multicast Receive Queue block, which allows messages to be received from multiple sources. Additionally, this block emulates the behaviour of a FIFO queue, where the tokens are enqueued. During the implementation, one of these blocks was assigned to each CP;
- The Entity Terminator block, through which messages can be retrieved from the queue and processed.

2) *Charging Points Communicate to Achieve Consensus: Use of Synchronous Dataflow*: The interaction logic between the different CPs, aimed at achieving consensus on the percentage of power to draw from the global generator, can be understood by referring to the model shown in Fig. 15. This represents a dataflow model of the synchronous type.

Synchronous dataflow (SDF) refers to a constrained form of dataflow where each actor, in every reaction, consumes a fixed number of tokens on each input port and produces a fixed number of tokens on each output port.

As shown in Fig. 15, each actor produces and consumes 2 tokens per reaction:

- Two input tokens, which are the values v and w of the sending actor, representing the numerical value and the associated weight, used by the Push-Sum algorithm to compute the average;
- Two output tokens, which are the values v and w appropriately processed by the actor.

The balance equations of the model, excluding the feedback loop from actor CP3 to actor CP1, are as follows:

$$2q_{CP1} = 2q_{CP2} \quad (18)$$

$$2q_{CP2} = 2q_{CP3} \quad (19)$$

From these equations, the following homogeneous system of linear equations is derived:

$$\begin{cases} 2q_{CP1} - 2q_{CP2} = 0 \\ 2q_{CP2} - 2q_{CP3} = 0 \end{cases} \quad (20)$$

Thus, the coefficient matrix \mathbf{M} , which multiplies the vector of unknowns $[q_{CP1}, q_{CP2}, q_{CP3}]^T$, can be derived:

$$\mathbf{M} = \begin{bmatrix} 2 & -2 & 0 \\ 0 & 2 & -2 \end{bmatrix} \quad (21)$$

Such a system can be represented in matrix form as follows:

$$\mathbf{M}\mathbf{q} = 0 \quad (22)$$

It is necessary to identify a solution in the vector \mathbf{q} that satisfies the system.

Since the system is underdetermined, meaning the number of unknowns exceeds the number of equations, there are infinite solutions. This is further confirmed by the fact that the coefficient matrix (Eq. 21) is not of full column rank. Consequently, there exist infinite solutions beyond the trivial one. The trivial solution, although solving the system, would not be useful for the analysis as it represents the absence of scheduling.

Considering that the unknowns q_{CPi} must be integers, among the infinite solutions, the one that minimises the buffer size is as follows:

$$q_{CP3} = q_{CP2} = q_{CP1} = 1 \quad (23)$$

As demonstrated, the balance equations are satisfied, and it is therefore possible to predict the presence of limited-size buffers. This indicates that the model is consistent.

Furthermore, it can also be stated that there exists a scheduling (sequence of reactions of the machines composing the system) that allows the entire system to function. An admissible scheduling in this case is as follows:

$$CP1 \ CP2 \ CP3 \quad (24)$$

This analysis shows that, without considering the feedback, the model is consistent. That is, it is possible to

have limited buffers that ensure the infinite execution of the system.

The ability to perform such rigorous analyses is guaranteed by the "synchronous" attribute of dataflow. Indeed, it is certain that every time an actor reacts, the number of tokens produced/consumed is always the same.

However, in this case, consistency does not guarantee the existence of a scheduling because the feedback loop must also be considered.

The balance equations are satisfied, and the model is consistent, but as shown in Fig. 15, without initialisation on the feedback channel, a deadlock situation would occur. Consequently, the scheduling would not exist.

To resolve the deadlock, it is sufficient to introduce 2 initialisation tokens on the feedback channel with $v = 0$ and $w = 0$.

3) Charging Points Communicate to Elect a Leader: Use of Structured Dataflow: Regarding the coordination process among the CPs for the purpose of electing a leader, it should be noted that the implementation of the logic was directly realised in Simulink. The approach adopted in this case is based on structured dataflow. Specifically, the behaviour of each CP depends on the type of message received, particularly its value. The message is processed in the entity terminator, and based on its value, a specific action is performed. It is worth noting that this block allows the execution of code similar to a MATLAB function. Consequently, if necessary, it can trigger the transmission of new messages to other CPs. These actions are organised similarly to structured programming, where the system's behaviour is determined by a series of conditions.

As a result, the entire system's operation relies on these conditions, which uniquely determine the actions each charging point must perform. Each CP acts solely in response to the tokens received, interpreting their value.

G. Bully Algorithm: Leader Election

The Bully Algorithm is a commonly used solution in distributed systems to select a coordinator process among a set of processes. By imagining a distributed system as a network of interconnected machines, the algorithm enables the election of a network node as the leader, ensuring that all other nodes know when the leader has been elected and which node has been chosen. Additionally, if the coordinator fails, the algorithm restarts to initiate a new election.

For the algorithm to function correctly, each node in the network must be assigned a static unique identifier (UID) that cannot change over time. During each election, it is guaranteed that the active node with the highest UID will be elected as the leader.

The term "bully" in the algorithm's name arises from the fact that when a node detects that the leader is malfunctioning (e.g., it does not receive any response), it overrides lower-ranked nodes and "challenges" nodes

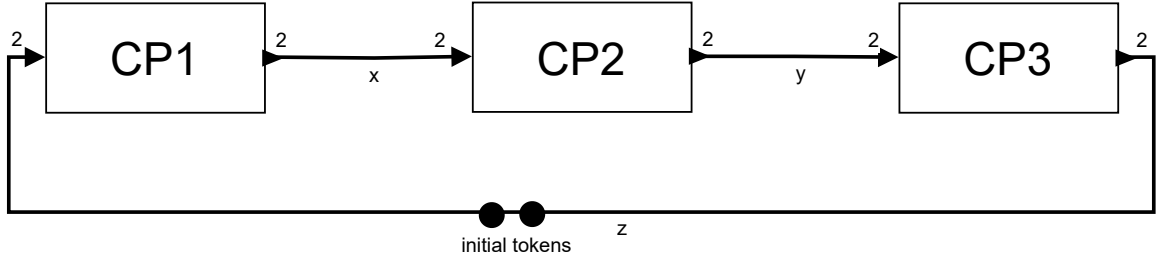


Fig. 15: Synchronous dataflow model of the interaction process among charging points, aimed at achieving consensus on the percentage of power allocated to each active CP.

with higher authority, i.e., nodes with higher UIDs, attempting to take control by sending a message to all nodes with UIDs higher than its own.

If a node with a higher UID receives the message and can become the leader, it responds to the node attempting to take control, indicating that it will now nominate itself, and in turn, sends a message to all nodes with UIDs higher than its own.

The algorithm continues in this manner until the node communicating its willingness to become the new leader receives no response or if the node attempting to take control has the highest UID, meaning there are no other nodes to contact.

When a node is elected, it declares itself the new leader, sending a message to all other nodes to inform them of the election.

In this work, the algorithm was used to satisfy the initial condition required to calculate the average power allocations that the charging points could deliver. Specifically, it was necessary for only one charging point to store the value 1, while all other CPs were assigned a null value.

To achieve this, during simulation, the functionality of the algorithm was replicated using Simulink and the SimEvents library described earlier. More specifically, for each CP, a MATLAB function was implemented to monitor changes in the pReq signal, outputted by EV. Whenever there was a change in energy demand from any connected vehicle, this MATLAB function communicated to all other CPs the need to intelligently redistribute resources. The CPs would then reset their numerical values used for power allocation calculations, consequently suspending power flow.

Each CP includes a "dead" module (dead_CP2 in Fig. 16) through which a message is sent to other charging points, informing them that the leader must be re-elected, i.e., the power flow distribution must be reorganised. When a CP receives and processes such a message, it resets its value and suspends power delivery.

At this point, each active CP requiring vehicle charging nominates itself as the new leader, communicating its UID to CPs with higher UIDs via a message. These higher-UID CPs process the message and, if active, respond using a dedicated response module (response_2_to_1 in Fig. 16), attempting in turn to take control and challenging CPs with higher UIDs.

The charging point that, after sending a message containing its UID as a nomination, receives no response within the 1s time limit, is elected as the new leader and communicates this to other charging points via a dedicated module (elected_CP2 in Fig. 16). It then sets its numerical value for power allocation calculations to $v = 1$.

H. Push-Sum Algorithm: Distributed Consensus

The Push-Sum algorithm is a well-known distributed protocol in the literature, typically used to compute aggregate functions [6], including the average of values across a network of nodes. Unlike the previously described Bully Algorithm, the Push-Sum algorithm operates as a continuously running calculation process, constantly updating the estimate of the average at various nodes in the network.

Consider a network consisting of N nodes arranged in a ring topology, where each node communicates with its immediate neighbour in a clockwise direction. Each node i possesses a numerical value v_i representing a local property, which in this case denotes the power allocation delivered by charging point CP_i . The goal of the algorithm is to provide all nodes with an accurate estimate of the average value $\bar{v} = \frac{1}{N} \sum_{i=1}^N v_i$ through a process of continuous distribution and updating.

1) *Algorithm Structure:* The Push-Sum algorithm can be divided into the following phases:

a) *Initialisation Phase:* Each node initialises two variables:

- v , representing the node's current value;
- w , representing the weight associated with this value, initially set to 1.

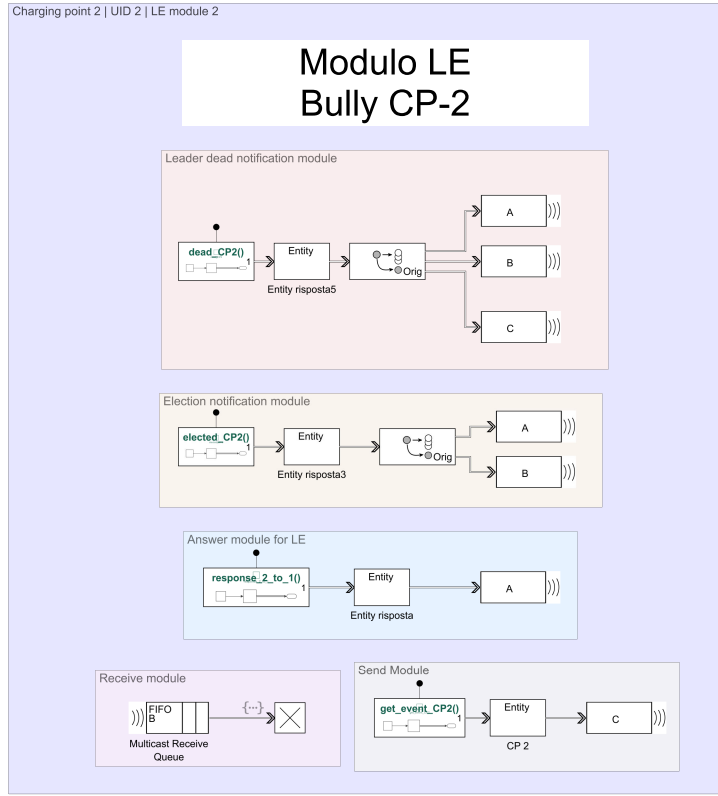


Fig. 16: Bully algorithm module for CP2

b) *Execution Phase:* For active nodes, i.e., those with a connected vehicle requesting power, the algorithm continuously executes a cycle, described in the following steps:

Algorithm 1 Push-Sum Averaging

- 1: **loop**
 - 2: wait(Δ) {Wait for a random time}
 - 3: $p \leftarrow \text{next peer}$ {Select the next node}
 - 4: sendPush($p, (x/2, w/2)$) {Send half of the value and weight}
 - 5: $x \leftarrow x/2$ {Update the local value}
 - 6: $w \leftarrow w/2$ {Update the local weight}
 - 7: **end loop**
-

Otherwise, if the node represents an inactive CP, these steps are not executed.

When a node receives a message (m) and is active, it processes the received value and weight, performing the following procedure:

Algorithm 2 onPush(m)

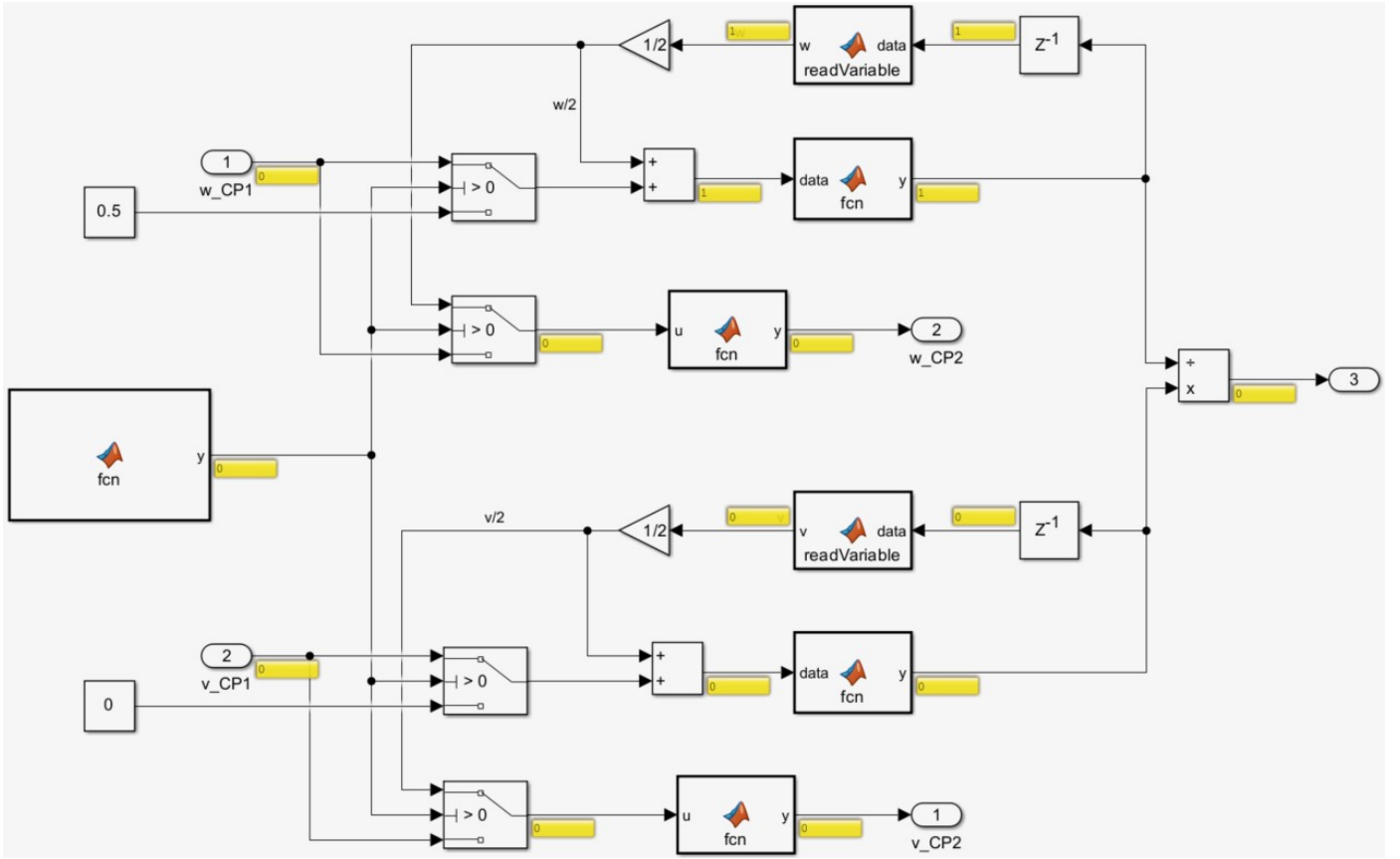
- 1: $x \leftarrow m.x + x$ {Add the received value to the local value}
 - 2: $w \leftarrow m.w + w$ {Add the received weight to the local weight}
-

Conversely, if the charging point receiving the message is inactive, it simply propagates the received message to its immediate neighbour in a clockwise direction. Thus, its behaviour is that of a simple propagator.

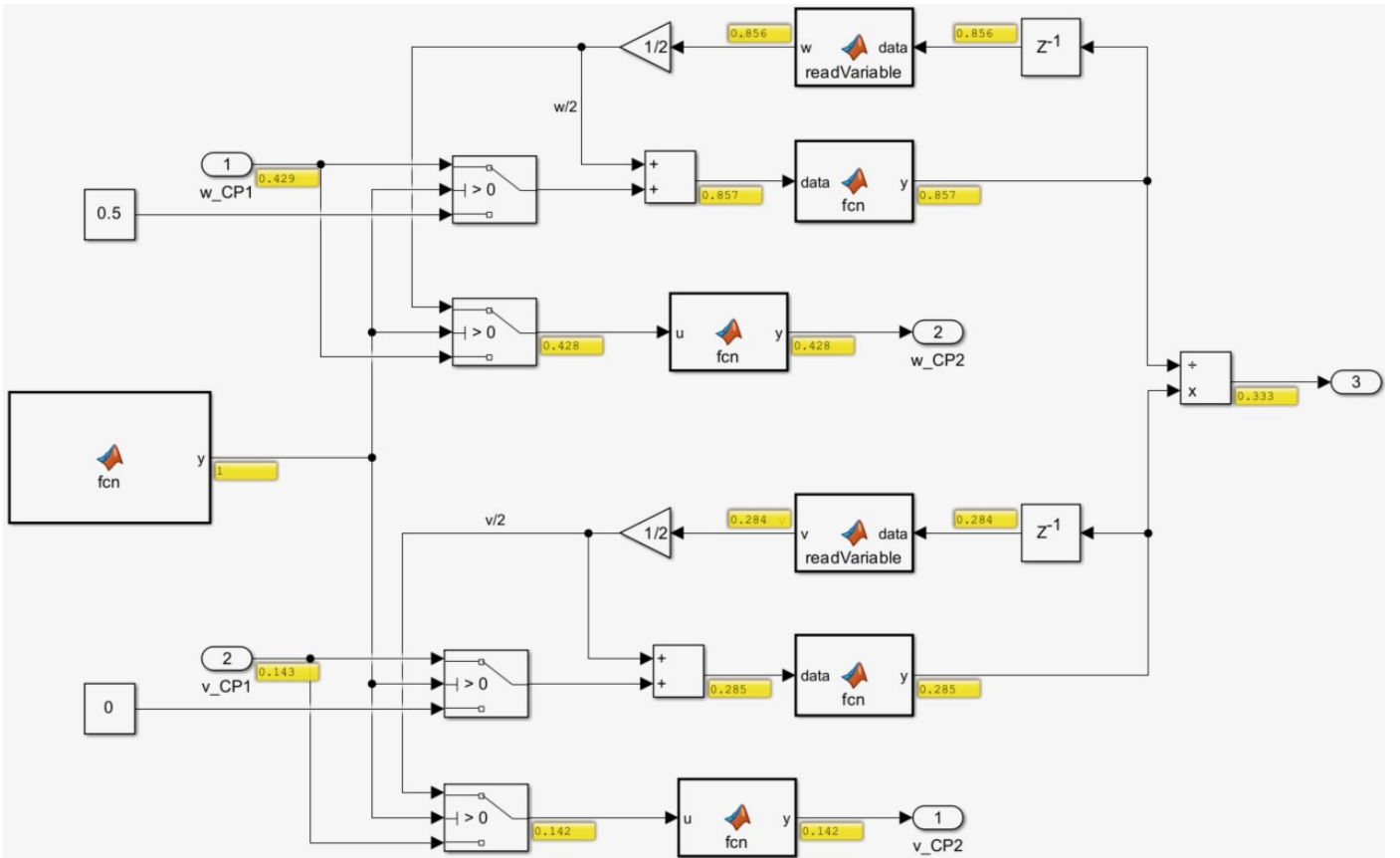
2) *Convergence to the Average Value:* Over time, each node dilutes its value and weight, distributing them among neighbouring nodes. As the process of sending and receiving continues, the values and weights spread across the network. After a sufficient number of iterations, the values and weights become evenly distributed among the nodes, enabling each node i to possess an accurate estimate of the network's average value, which is calculated as the ratio between v_i and w_i .

In other words, through the process of continuous dilution and redistribution, the Push-Sum algorithm ensures that every node in the network converges to the same estimate of the average value, regardless of the network topology. This algorithm is specifically designed to achieve convergence in distributed systems, i.e., networks with topologies dependent on the application context and therefore rarely represented as complete graphs.

During simulation, a mechanism was implemented such that if, between one sampling step and the next, the average value of a node remains constant within a tolerance factor of $p = 0.0001$, it is assumed that convergence to the average value has been achieved. Consequently, only when this condition is met can the resulting value be used to determine the percentage of power from the "global" generator that can be delivered by the corresponding CP.



(a) Values generated by the Push-Sum module prior to leader election.



(b) Values generated by the Push-Sum module after leader election with three active charging points.

Fig. 17: Simulink diagram of the Push-Sum module for CP2.
Labels indicate the values produced and sent over the respective channel.

V. MODEL IMPLEMENTATION IN MATLAB SIMULINK & STATEFLOW

After analysing and modelling the various characteristics of the system, its complete implementation was carried out using Simulink and Stateflow tools.

A. Differences Between the Model and Its Implementation

As can be observed from the Stateflow diagrams of the state machines and their composition (Figs. 5 and 9), differences exist between the model and its implementation. These divergences are mainly due to:

- The use of blocks designed to reduce model complexity, made available by the tool. An example is the use of the connective junction block;
- The use of probabilistic guards for transitions, which make the state machines non-deterministic. For further explanation, refer to Chapter IV-B.3.

B. Simulink & Stateflow Diagram of the Push-Sum Algorithm

Figure 17 shows the Simulink diagram of the Push-Sum module for charging point CP2. The labels on the connections between blocks indicate the value transmitted through the respective communication channel at a specific simulation step. It should be noted that the network consists of three nodes arranged in a ring topology, as shown in Fig. 15, and each of these three nodes is characterised by an identical Push-Sum module.

In Fig. 17a, the values computed at a time when the leader has not yet been elected are shown. From the MATLAB function on the left, a null numerical value is output, which controls the four switch blocks, allowing them to propagate null values of v and w to the next charging point CP3.

This MATLAB function generates an enable signal for the switch blocks, equal to 1 when a leader is present in the network and 0 otherwise.

The delay block Z^{-1} at the top right outputs an initial value of 1, which is continually halved as part of the algorithm illustrated in the diagram 1.

Two conditions must, however, be satisfied:

- 1) Immediately after leader election, as mentioned in IV-H.1.a, all nodes must start from an initial phase where their weights w are set to 1;
- 2) The average value estimate computed by each node, defined as:

$$\frac{v_i}{w_i} \quad (25)$$

must remain null until a leader is elected.

To achieve this, the weight value 1 input to the Gain block, after being halved, is summed with the output of the constant block 0.5.

This ensures that the value $w = 1$ remains continuously in circulation until a leader is elected. Meanwhile, the delay block at the bottom outputs an initial value of 0.

As a result, until a leader is elected, the average value estimate computed by each CP_i remains consistently null:

$$\frac{v_i}{w_i} = \frac{0}{1} = 0 \quad (26)$$

In Fig. 17b, the computed values at a time when the leader has already been elected, three CPs are active, and the correct average value estimate of 0.333 has been reached, are shown. Once the leader is elected, the MATLAB function on the left produces a constant value of 1, enabling the switch blocks to pass the values of v and w from CP1. These values are then used both to update CP2's internal variables and to compute a correct average value estimate.

The six additional MATLAB functions are used to:

- Re-establish the initial condition of the Push-Sum algorithm following the leader's failure, i.e., reset all v_i variables and set all w_i to 1;
- Compensate for delays between the leader's failure and the generation of a subsequent event to reinitialise the outputs of the described blocks.

C. Simulink & Stateflow Diagram of the System

Figure 18 shows the Simulink and Stateflow diagram of the entire system under consideration. For simulation purposes, a network of three charging points organised in a ring topology was chosen.

The operational logic of the diagram essentially involves the following phases:

- At the start of the simulation, the behaviour of a group of vehicles connecting to the network of CPs is simulated. Specifically, for each CP, it is probabilistically determined whether an EV is connected to it or not. Additionally, the initial *SoC* value of its battery is randomly set. These values are graphically displayed in the simulation environment using a display block placed to the left of each EV and a gauge indicator (horizontal gauge block) located beneath each vehicle;
- When a vehicle connects to a CP and its corresponding *pReq* signal goes high, the lamp to the right of the vehicle turns green; otherwise, it lights up red;
- Whenever there is a change in the value of the *pReq* signal, the difference between its previous and current values is captured by a dedicated MATLAB function, which initiates the Bully Algorithm for leader election, as explained in Section IV-G. It is important to note that any change in the *pReq* signals input to each CP triggers the election process, which is necessary to achieve consensus on the power allocation for each charging point;
- Each CP measures the time elapsed since the transmission of its candidacy message using its internal clock. It is assumed that the system operates in a synchronous distributed environment, where "synchronous" is used in a different sense compared to its use in the context of controllers. Here, it implies the ability to define an upper bound on transmission delay. Specifically, it is assumed that the maximum observable delay, within which a CP must receive a response after sending a candidacy message, is 1s. If no responses are received, according to the logic of the Bully Algorithm, the CP can declare itself the leader. This enables updating the variables used to achieve consensus;

- Finally, the Push-Sum algorithm, which runs continuously, determines consensus achievement. Specifically, the "Push-Sum Algorithm Module CP_i " in Fig. 18 outputs a value between 0 and 1, representing the percentage of power from the generator that can be delivered by the individual CP_i . This value is multiplied by the power output of the generator, which is assumed constant over time. The result of this multiplication is then fed into a switch, controlled by the *enablePowerFlow* signal.

D. Limitations of the Model

The developed model is based on several assumptions and simplifications, which are discussed in the previous chapters and summarised here for clarity:

- The network to be modelled is represented as a directed graph of order three with a ring topology. It should be noted that the proposed model implementation does not offer flexibility regarding the addition of nodes to the network or modifications to the topology itself;
- The communication network between CPs is assumed to be free from failures and reliable, meaning that communication between CPs is never interrupted or altered;

- The network of charging points is considered a synchronous distributed system, as described in Section V-C of this document. A distributed system, in this context, refers to a network of nodes that cooperate and communicate solely through message exchange;
- The composition between the EV and CP state machines is synchronous. It is reiterated that perfect synchrony does not exist in reality, as described in Section IV-D.1 of this document;
- The necessary conditions for the Bully Algorithm to function are always met. For example, each node in the network must be labelled with a unique identifier.

These are the main assumptions that enable the model to operate. However, there are also other, less significant and less evident assumptions. This emphasises the awareness of the model's limitations. Nevertheless, this approach aligns well with the primary contextual requirement: to use the tools and concepts learned during the course "Analysis and Control of Cyber-Physical Systems" with awareness.

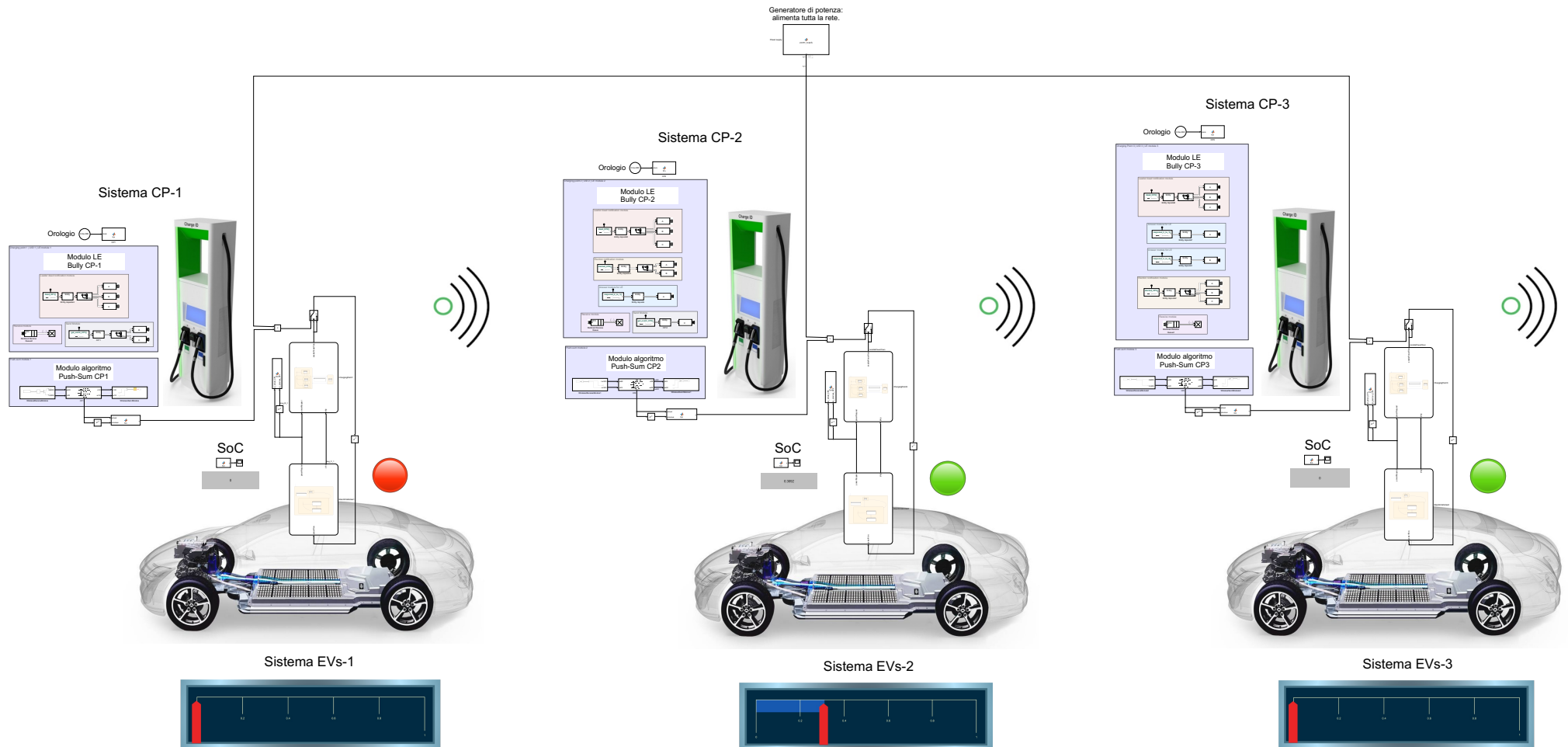


Fig. 18: Simulink and Stateflow diagram of the complete system.

VI. FORMAL VERIFICATION OF SPECIFICATIONS - STL

For the verification of specifications, formally defined using STL, the Simulink Test tool was utilised. With Simulink Test, it is possible to create tests for entire systems or isolated components. Requirement-based assessments can be defined starting from temporal logic such as STL, specifying test inputs, expected outputs, and tolerances.

It should be noted that Simulink Test does not support the use of the implication operator or the temporal operators specific to STL. Therefore, to verify the described specifications, they were adapted by referencing the operators supported and provided by the tool itself. For example, the implication operation was represented using the following equivalent operation:

$$A \implies B = \neg(A) \vee B \quad (27)$$

This equivalence is demonstrated by the following truth table:

A	B	$A \implies B$	$\neg(A)$	$\neg(A) \vee B$
T	T	T	F	T
T	F	F	F	F
F	T	T	T	T
F	F	T	T	T

The other tools provided by Simulink Test and used are:

- **Bounds Check**, which refers to a type of test that verifies whether signals or variables in a Simulink model respect certain limits (bounds) during simulation execution. This type of check is essential for identifying any values that exceed expected ranges, signalling potential errors or abnormal behaviours in the system. For example, this test was used to describe Specification 7, shown in Table VIIa;
- **Trigger Response**, which refers to a mechanism for triggering a specific response or behaviour in a system based on certain conditions occurring during the simulation. This test was used to describe Specification 9 (Table IXa).

It is worth noting that during the definition and verification of the specifications, signals not anticipated during the model development phase were conceived. These signals are:

- $SoC \in \mathbb{A}^{\mathbb{R}^+}$, with $\mathbb{A} = \{x \in \mathbb{R} \mid 0 \leq x \leq 1\}$, represents the battery charge level of a generic EV. This signal was used in Specifications 2 (Table IIa) and 6 (Table VI);
- $completedState \in \mathbb{B}^{\mathbb{R}^+}$ assumes the value 1 when the state machine of a generic CP is in the *Completed* state, and 0 otherwise. This signal was used in Specification 3 (Table IIIb);
- $errorState \in \mathbb{B}^{\mathbb{R}^+}$ assumes the value 1 when the state machine of a generic CP is in the *Error* state, and 0 otherwise. This signal was used in Specification 5 (Table Va);
- $generatorPower = 10000$ represents the power provided by the network generator that supplies all CPs. It is assumed that this value is constant over

time. This signal was used in Specifications 7 (Table VIIa) and 9 (Table IXa);

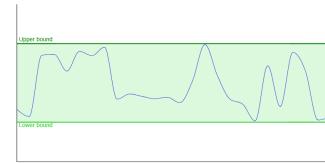
- $pS_{i_idle} \in \mathbb{R}^{\mathbb{R}^+}$ is the result of the multiplication between the signal $idleState \in \mathbb{B}^{\mathbb{R}^+}$, which indicates whether CP_i is in the *Idle* state or not, and the signal $pFlow \in \mathbb{R}^{\mathbb{R}^+}$. When the charging point is in the *Idle* state, the corresponding $idleState$ signal is 1, so the multiplier output will be zero only if the power delivered by the CP ($pFlow$) is also zero. If the charging point i has a connected vehicle, the pS_{i_idle} signal (multiplier output) will still be zero because the corresponding $idleState$ signal will be 0. This signal was used in Specification 8 (Table VIIIa);
- $pReq_i^{actual}$ represents the current value of the $pReq$ signal for CP_i . This signal was used in Specification 9 (Table IXa);
- $pReq_i^{previous}$ represents the delayed value of the $pReq$ signal for CP_i by one step. This signal was used in Specification 9 (Table IXa).

Due to the limitations of the Simulink Test tool, it was not possible to verify the assessment related to Specification $S_{EV_#3}$.

For convenience, to aid in understanding the formalisms used in defining the specifications, the details listed above have also been included in the captions of the tables presented on the following pages.

1) *Limitations of the Specification Verification Process*: Simulink Test verifies specifications only during simulation time, which means that if a specific condition or specification does not occur within the simulated time frame, the test will fail to detect it. This limitation may arise due to various reasons, including:

- **Insufficient simulation duration**: If the simulation does not run long enough for the condition requiring specification verification to arise, the specification will not be adequately tested;
- **Incomplete model coverage**: If the events or conditions leading to a specification failure do not occur during the simulation, it will be impossible to verify that the system reacts correctly to such events.



(a) Visual representation of the specification $S_{CPS_#1}$.



(b) Visual representation of the specification $S_{CPS_#3}$.

Fig. 19: Visual representations of certain specifications defined using the Simulink Test tool.

Specification 1	
Natural Language	An electric vehicle cannot request power unless it is connected to a charging point
STL - φ_1	$G \{ [plug(t) = 0] \implies [pReq(t) = 0] \}$
Simulink Test - S_EV_#1	At any point of time $\{ [\neg(plug == false)] (pReq == false) \}$

TABLE I: Specification 1 expressed in different formats.

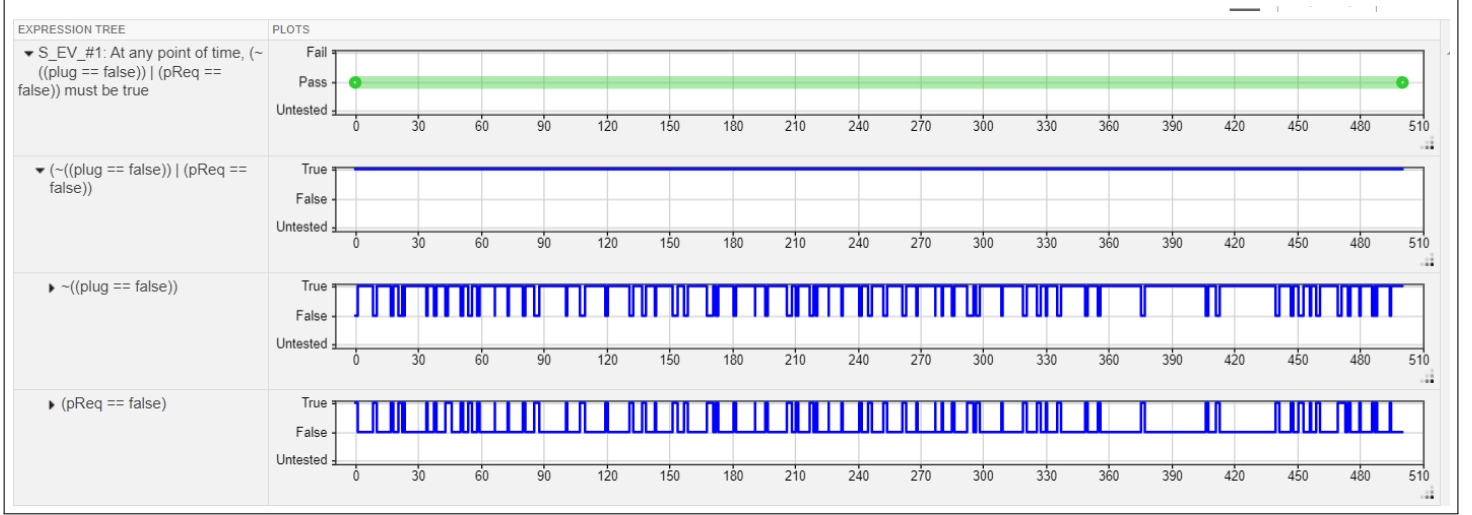


Fig. 20: Verification of the S_EV_#1 specification in Simulink Test.

Specification 2	
Natural Language	If a vehicle's battery is fully charged, it cannot request power
STL - φ_2	$G \{ [SoC(t) = 1] \implies [pReq(t) = 0] \}$
Simulink Test - S_EV_#2	At any point of time $\{ [\neg(SoC_EV == 1)] (pReq == false) \}$

TABLE II: Specification 2 expressed in different formats.

- (a) The signal $SoC \in \mathbb{A}_+^{\mathbb{R}}$ where $\mathbb{A} = \{x \in \mathbb{R} \mid 0 \leq x \leq 1\}$ and represents the value of the battery charge level of the vehicle considered.

Specification 3	
Natural Language	If a vehicle's battery level is between 95% and 100% and was previously at 100%, it cannot request power until the State of Charge (SoC) drops below 95%
STL - φ_3	$G \{ [(0.95 \leq SoC(t) \leq 1) \mathbf{S}(SoC(t^*) = 1)] \implies (pReq(t) = 0) \} \mid t^* < t$
Simulink Test - S_EV_#3 (unverified assessment)	At any point of time, if <i>completedState</i> == <i>true</i> becomes true, then, with a delay of at most 0.1 seconds, <i>pReq</i> == <i>false</i> must stay true until <i>SoC</i> ≥ 0.95 and <i>plug</i> == <i>true</i>

TABLE III: Specification 3 expressed in different formats.

- (a) The STL formula can be read as follows: 'It must always be verified that the fact that $0.95 \leq SoC(t) \leq 1$ since it was equal to 1, at some time in the past, implies that there is no demand for power until the value of *SoC* falls below 0.95.
- (b) In writing the assessment in Simulink Test, consideration was given to the fact that, when a value of $SoC(t) == 1$ is reached, if the vehicle is connected ($plug == true$), it will enter the Completed state and remain in that state as long as the value of *SoC* of the same vehicle remains above 0.95. In fact, the signal $completedState(t) \in \mathbb{B}^{\mathbb{R}_+}$ takes value 1 if one is in the Completed state, 0 otherwise.

Specification 4	
Natural Language	The charging point can supply power only if a vehicle is connected
STL - φ_4	$G \{ [enablePowerFlow(t) = 1] \implies [plug(t) = 1] \}$
Simulink Test - S_CP_#1	At any point of time $\{ [\neg(enablePowerFlow == true)] (plug == true) \}$

TABLE IV: Specification 4 expressed in different formats.

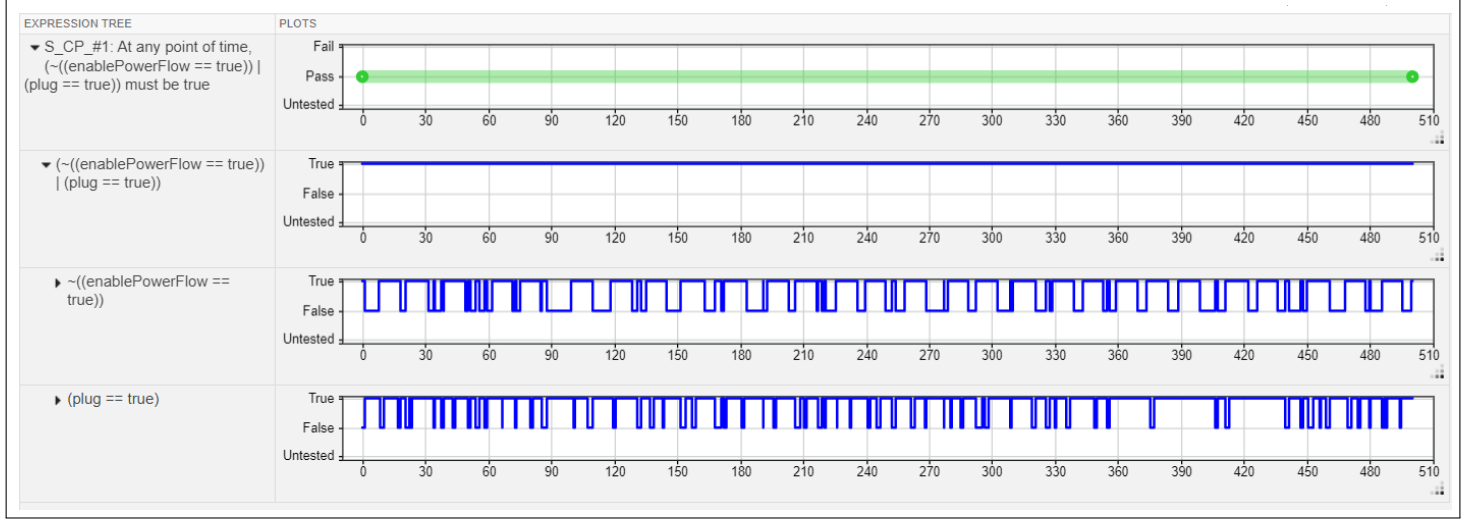


Fig. 21: Verification of the S_CP_#1 specification in Simulink Test.

Specification 5	
Natural Language	The charging point cannot supply power if it is out of service
STL - φ_5	$G \{ [errorState(t) = 1] \implies [enablePowerFlow(t) = 0] \}$
Simulink Test - S_CP_#2	At any point of time $\{ \neg(errorState == true) (enablePowerFlow == false) \}$

TABLE V: Specification 5 expressed in different formats.

- (a) The signal $errorState(t) \in \mathbb{B}^{R+}$ was only used in the specification validation phase. It takes the value 1 when the charging point is in an error state, otherwise it is 0.

Specification 6	
Natural Language	The charging point delivers power to the vehicle, only if the vehicle's battery is not fully charged
STL - φ_6	$G \{ [enablePowerFlow(t) = 1] \implies [SoC(t) \neq 1] \}$
Simulink Test - S_EV&CP_#1	At any point of time, $\{ \neg(enablePowerFlow == true) (SoC \neq 1) \}$ must be true.

TABLE VI: Specification 6 expressed in different formats.

Specification 7	
Natural Language	At any moment, the sum of the power supplied by the charging points must be greater than or equal to zero and should never exceed the total power supplied by the generator
STL - φ_7	$G\{0 \leq [\sum_{i=1}^n pFlow_i(t)] \leq generatorPower(t)\}, \quad n = \text{number of CPs}$
Simulink Test - S_CPS_#1	At any point of time, $pFlow_1 + pFlow_2 + pFlow_3$ must be greater than or equal to 0 and less than or equal to $generatorPower$

TABLE VII: Specification 7 expressed in different formats.

(a) The signal $pFlow_i(t)$ represents the power rate delivered by CP_i . The signal $generatorPower$ has been assumed constant and equal to $10'000$.

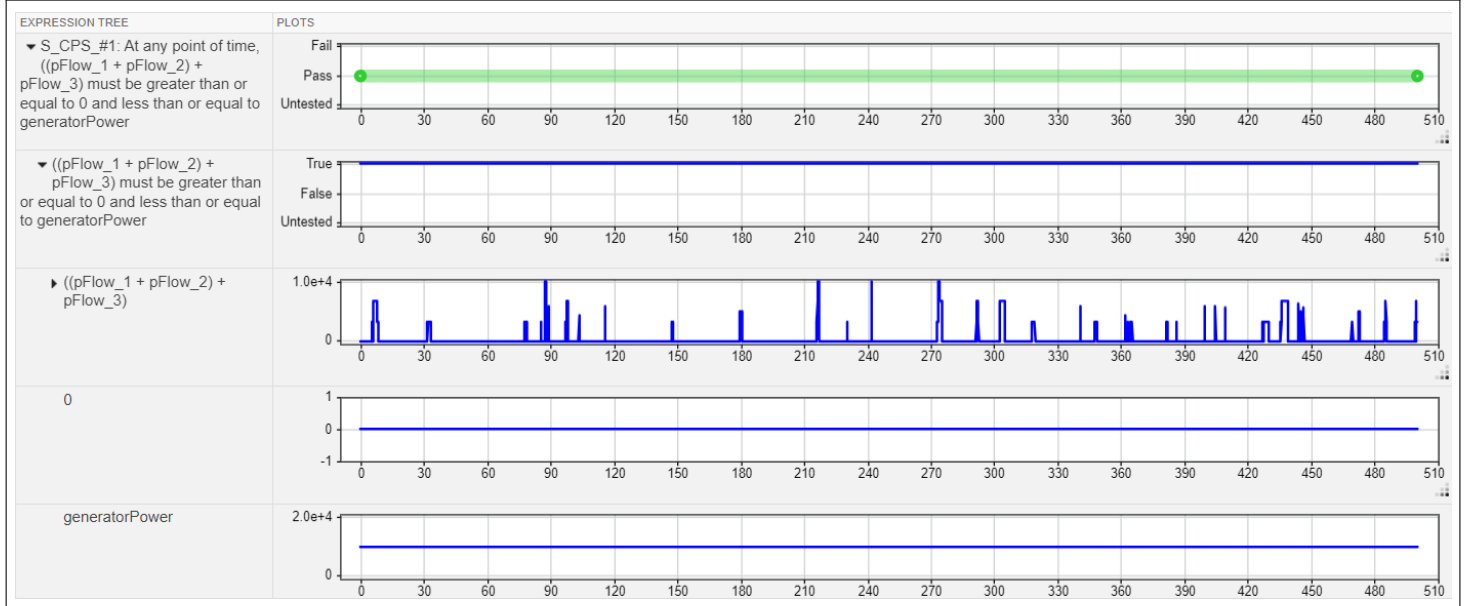


Fig. 22: Verification of the S_CPS_#1 specification in Simulink Test.

Specification 8	
Natural Language	At any moment, the sum of the power supplied by charging points without connected vehicles must be zero
STL - φ_8	$G\{[\sum_{i=1}^{n_{idle}} pFlow_i(t)] = 0\}, \quad n_{idle} = \text{number of CPs in idle state Idle}$
Simulink Test - S_CPS_#2	At any point of time, $pS_1.idle + pS_2.idle + pS_3.idle == 0$ must be true.

TABLE VIII: Specification 8 expressed in different formats.

(a) The signal $pS_{i.idle}(t) \in \mathbb{R}^+$ is the result of the multiplication between the signal $idleState(t) \in \mathbb{R}^+$ which indicates whether the CP_i is in the *Idle* state or not, and the signal $pFlow(t) \in \mathbb{R}^+$. When the charging point is in the *Idle* state, the corresponding signal $idleState(t)$ will be equal to 1, so the output of the multiplier will be null only if the power delivered by the CP ($pFlow(t)$) is null. If, on the other hand, the charging point i has a vehicle connected, then the related signal $pS_{i.idle}(t)$ (output of the multiplier) will still be null, since the related signal $idleState(t)$ will be 0. Note that the signal $pS_{i.idle}(t)$ has been modelled in this way in order to find a strategy to detect CPs in the *Idle* state.

Specification 9	
Natural Language	Within 4s of the last change in a vehicle's power demand, each active and non-faulty CP must deliver power equal to that of the generator divided by the number of active charging points
STL - φ_9	$\bigvee_{i=1}^n \left[pReq_i^{actual}(t) \neq pReq_i^{previous}(t) \right] \implies$ $F_{[0,4]} \left\{ \bigwedge_{i=1}^n \left[pFlow_i(t) = \frac{generatorPower(t) \cdot poweringState_{CP_i}(t)}{n_{CP_active}} \right] \right\}$
Simulink Test - S_CPS_#3	At any point of time, if $(pReq_1_a \neq pReq_1_p) ((pReq_2_a \neq pReq_2_p) (pReq_3_a \neq pReq_3_p))$ becomes true then, with a delay of at most 4 seconds, $((pFlow_1 == \alpha) \& (pFlow_2 == \beta) \& (pFlow_3 == \gamma)) ((pReq_1_a \neq pReq_1_p) ((pReq_2_a \neq pReq_2_p) (pReq_3_a \neq pReq_3_p)))$ must be true

TABLE IX: Specification 9 expressed in different formats.

(a) The term $\alpha(t) = \frac{generatorPower(t) \cdot poweringState_{CP1}(t)}{numero\ di\ CP\ attivi\ all'istante\ t}$ and represents the power rate delivered by the charging point CP1 at time instant t . It is highlighted, that when such a CP does not have a connected vehicle requesting power, the numerator cancels as $poweringState_{CP1}(t) = 0$. In fact, the state machine of the CP is in the state *Powering* if and only if $plug(t) = 1$ and $pReq(t) = 1$ result. Furthermore, if the numerator is not zero, it is divided by the number of active CPs, which are those participating in the consensus process. The same reasoning applies for the signals $\beta(t)$ and $\gamma(t)$.

VII. EXPERIMENTAL RESULTS

When designing a system model, such as a cyber-physical system, the primary objective is to enable detailed analysis. The model is not only created to represent the system but also to serve as a tool for performing evaluations that would be impractical or risky to conduct directly on the real system.

To this end, various simulations were carried out to assess specific aspects of the system. These simulations allow the analysis of the model's behaviour under different scenarios and conditions, verifying how the CPS reacts in critical or particularly complex situations. Through these simulations, it is possible to gain a deeper understanding of the system's performance, safety, and reliability, confirming the validity of the model before its real-world application.

In particular, for the case considered, several graphical views of the system's behaviour were produced and are presented in Figures 23, 24, 25, and 26.

Figure 23 illustrates how the charging points reach consensus concerning the power demands of the vehicles. Graphically, the consensus process is represented by the "funnel effect," where the functions progressively converge towards the same final value. It is noted that, in the considered context, consensus is achieved when all active charging points possess the same value, calculated using the Push-Sum algorithm. This value represents the power allocation from the global generator assigned to them. An analysis of the graph reveals that:

- The funnel effect lasts longer when the power requests from vehicles to their respective CPs remain unchanged. The funnel effect ceases when there is a change in power requests;
- From a performance perspective, it would be desirable to minimise the time required to reach consensus. This is influenced by communication latencies, network topology, and the number of nodes;
- The system has a vulnerability: if there is a continuous change in the power request state for a given CP, consensus would never be reached. Technically, this results in the continuous triggering of the Bully Algorithm, causing repeated leader election processes and resetting the estimates calculated by the Push-Sum algorithm.

Below is an interpretation of the system's behaviour during the simulation window from 0s to 5s (Figure 23). This time window reveals that:

- Charging point 1 receives a power request, as indicated by the *pReq-CP1* signal "rising";
- Charging point 2 receives a power request, as indicated by the *pReq-CP2* signal "rising";
- Charging point 3 receives a power request, as indicated by the *pReq-CP3* signal "rising";
- After leader election, the estimates from the Push-Sum modules begin to oscillate until consensus is reached (4.5s);
- Finally, the system's behaviour is disrupted by the termination of the power request from the vehicle connected to charging point 1.

Figures 24, 25, and 26 show how the *SoC* of a vehicle connected to a given CP increases during time intervals where consensus is achieved, i.e., when the estimates from the Push-Sum module converge. From an analysis of Figure 26, the following observations can be made:

- At the start of the simulation, the battery charge level is stationary at 40%;
- At 4.5s, consensus is achieved, and the *SoC* begins to increase (the battery charges);
- During the next two convergence attempts, consensus is not reached, causing the *SoC* to remain stationary at 52%;
- Shortly after 10s, a sudden change in the *SoC* occurs, corresponding to the connection of a new vehicle with a new charge level;
- At 15.4s, consensus is achieved, and the battery begins to charge, as seen from the evolution of the *SoC*;
- At 22.8s, a new vehicle is considered, not yet connected to the CP. Its battery charge level is 100%. The vehicle connects to the CP at 25s, but since its *SoC* is at maximum, no power is supplied, and the battery begins to discharge.

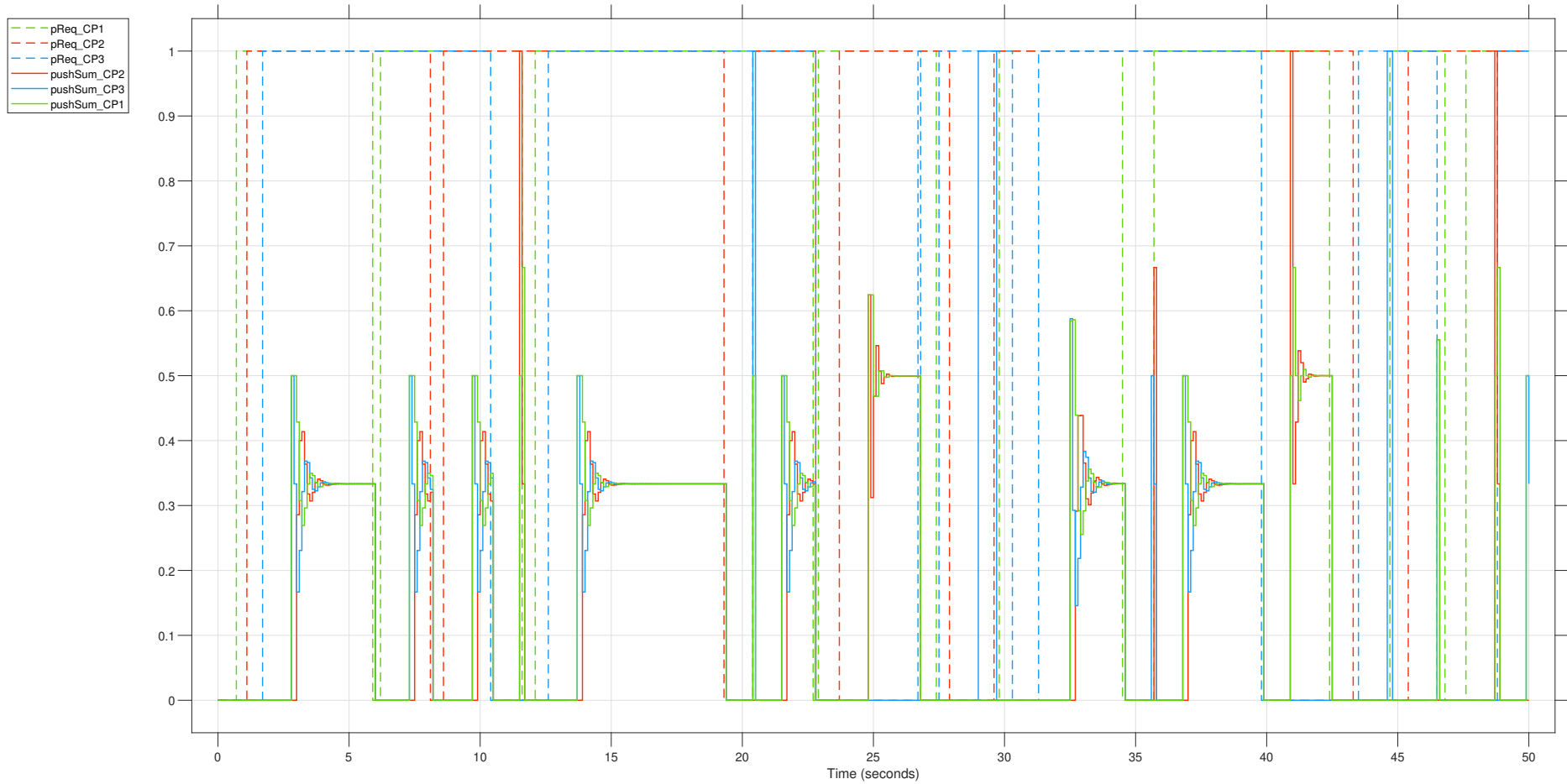


Fig. 23: Graph showing how the charging points reach consensus concerning the power demands of vehicles. The solid lines represent the average value estimates calculated by the Push-Sum modules. The dashed lines represent the power requests. Colours distinguish the different nodes (charging points) in the network.

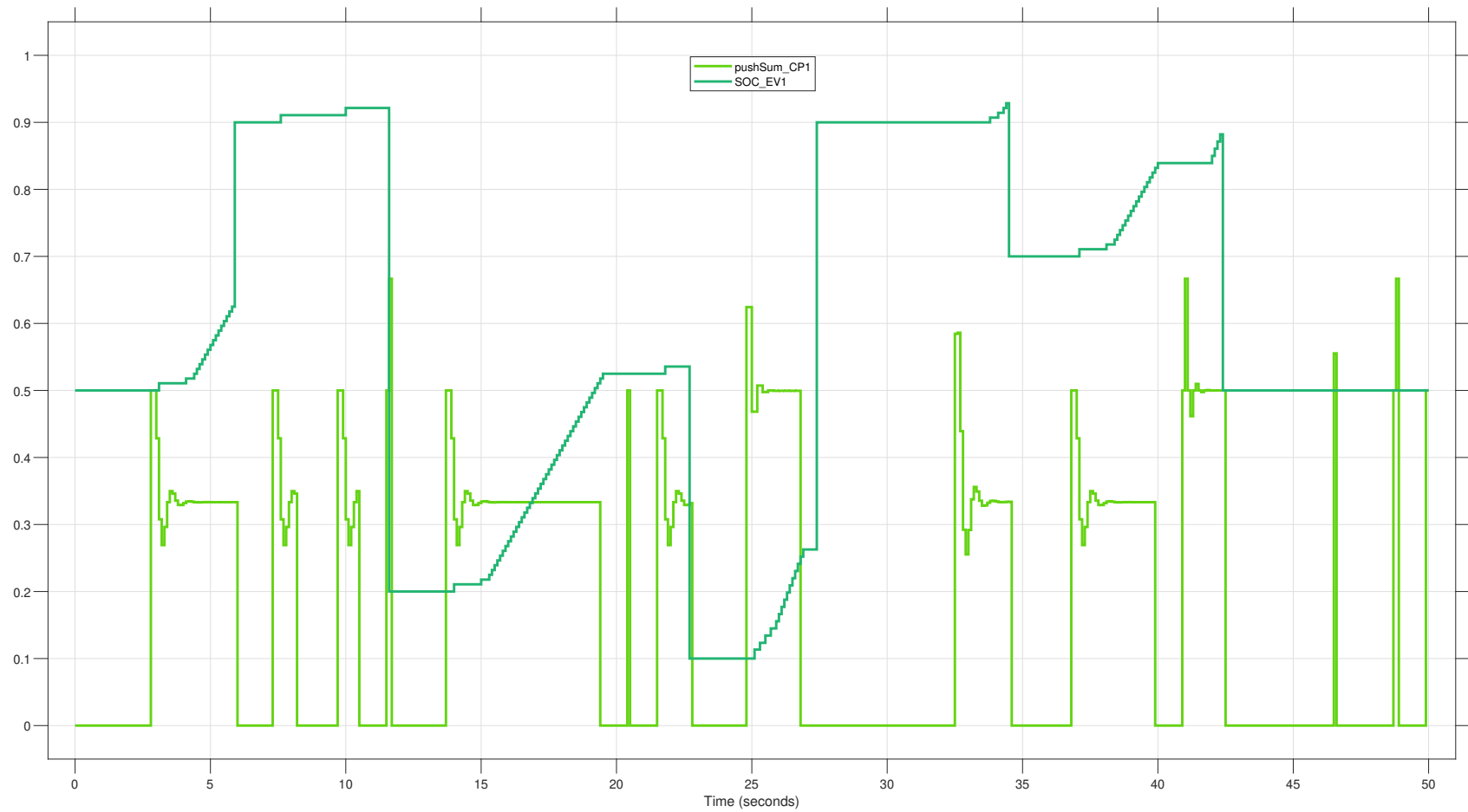


Fig. 24: Graph showing the *SoC* value of the vehicle connected to CP1 (dark-coloured function) in relation to the average value estimate calculated by the Push-Sum module of charging point 1 (light-coloured function).

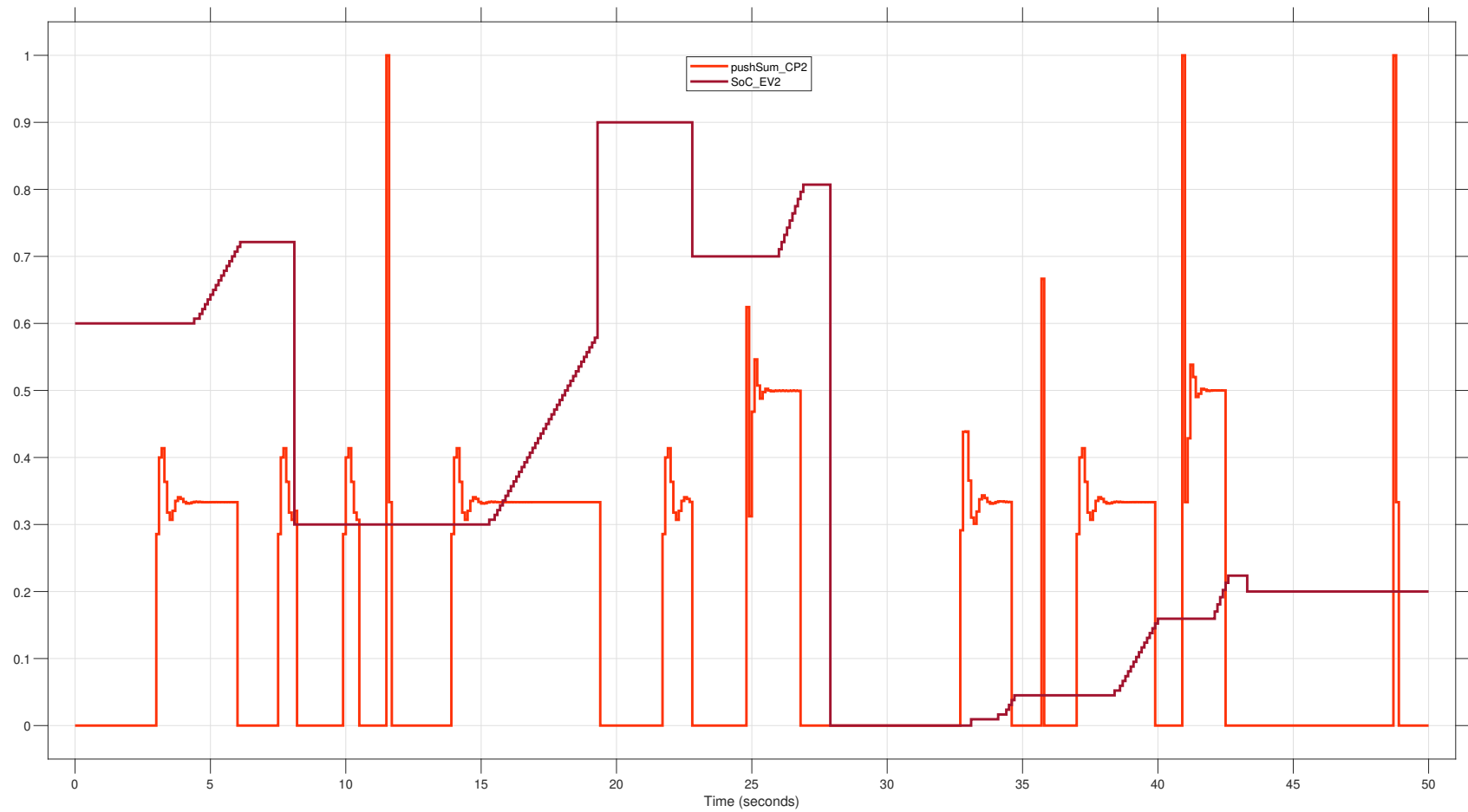


Fig. 25: Graph showing the *SoC* value of the vehicle connected to CP2 (dark-coloured function) in relation to the average value estimate calculated by the Push-Sum module of charging point 2 (light-coloured function).

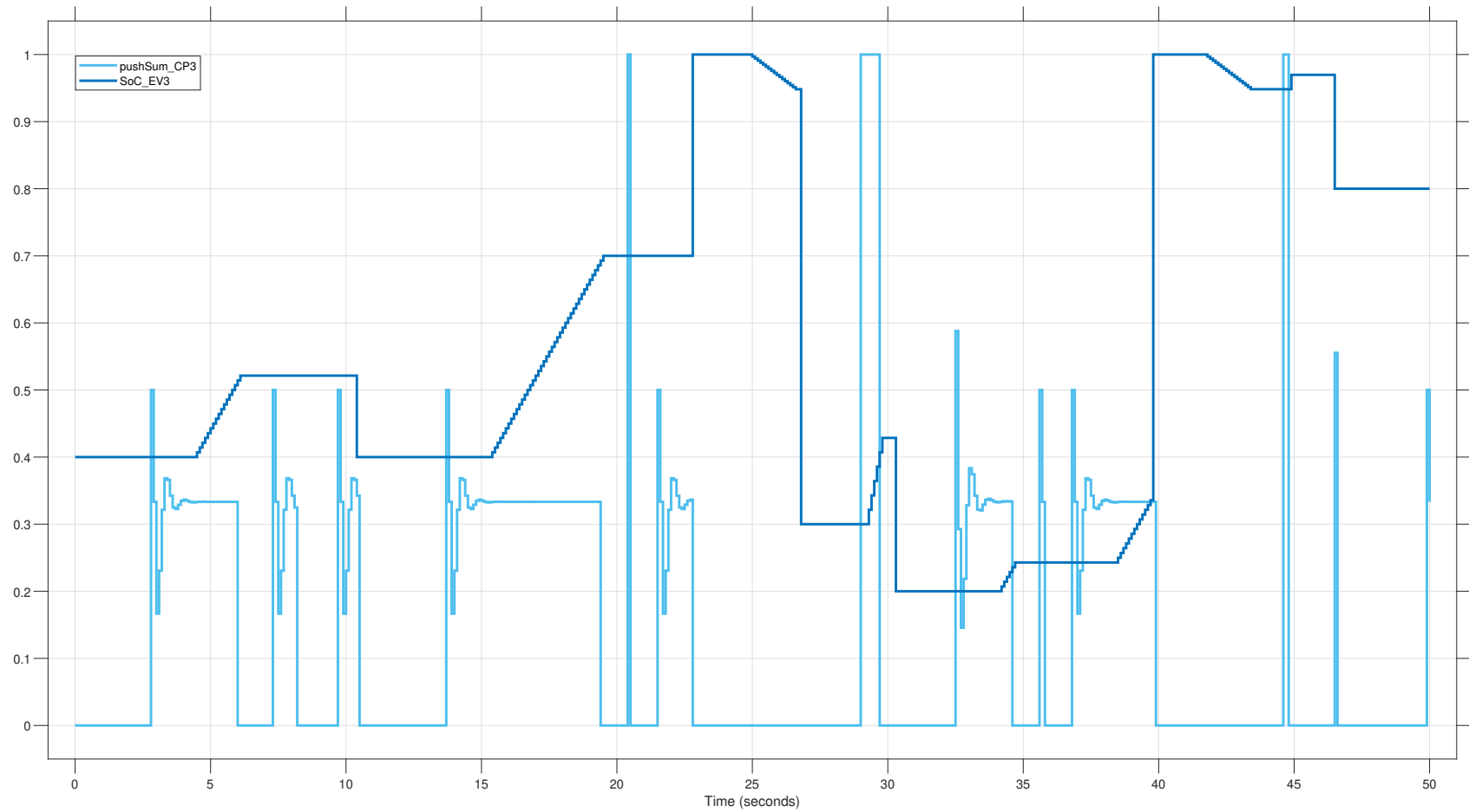


Fig. 26: Graph showing the *SoC* value of the vehicle connected to CP3 (dark-coloured function) in relation to the average value estimate calculated by the Push-Sum module of charging point 3 (light-coloured function).

VIII. CONCLUSIONS AND FUTURE WORK

The work discussed in this document allowed for the evaluation of the potential of concepts and knowledge acquired during the "Analysis and Control of Cyber-Physical Systems" course. During the model implementation phase, the team had to devise solutions to overcome various challenges that arose while applying the theoretical concepts learned. Consequently, the team demonstrated the ability to seek necessary knowledge and apply it to overcome the obstacles encountered, adhering to the philosophy stated at the beginning of the course: *"You can't teach people everything they need to know. The best you can do is position them where they can and what they need to know when they need to know it. (Seymour Papert)"*.

Future developments of the proposed solution could address various aspects of the project, including:

- Extending the base system models, namely those of the electric vehicle and the charging point, and detailing them according to contextual needs;
- Modifying the model implementation to allow for a network of non-homogeneous charging points. Additionally, it would be interesting to evaluate how different network topologies affect the convergence time of the Push-Sum algorithm used to achieve consensus on the power allocations each CP must deliver;
- Implementing the dataflow model using alternative solutions to the one employed in this document (Simulink SimEvents library). It was noted that, in complex scheme implementations, the library does not provide sufficient tools to reduce the effort of the implementation process.

In conclusion, it is emphasised that the ability to manage complex scenarios was acquired through the knowledge gained. Indeed, despite necessary simplifications, it was possible to structure a solution to a problem that the development team had never encountered before.

The entire project, including test files, has been made available on GitHub in a repository accessible [here](#).

REFERENCES

- [1] Analog Devices, "A Closer Look at State of Charge (SOC) and State of Health (SOH) Estimation Techniques for Batteries," Analog Devices Technical Articles, March 2023. [Online]. Available at: <https://www.analog.com/en/resources/technical-articles/a-closer-look-at-state-of-charge-and-state-health-estimation-tech.html>.
- [2] K. S. Ng, C. S. Moo, Y. P. Chen, and Y. C. Hsieh, "Enhanced Coulomb counting method for estimating state-of-charge and state-of-health of lithium-ion batteries," *Applied Energy*, vol. 86, no. 9, pp. 1506–1511, Sep. 2009. [Online]. Available at: <https://doi.org/10.1016/j.apenergy.2008.11.021>.
- [3] E-Station. Guide to electric cars. Available at: <https://www.e-station.it/guida-alle-auto-elettriche.html>.
- [4] E. A. Lee and S. A. Seshia, *Introduction to Embedded Systems - A Cyber-Physical Systems Approach*, Second Edition, MIT Press, 2017.
- [5] Simulink Test - MATLAB. More information on the library can be found at: <https://it.mathworks.com/products/simulink-test.html>.
- [6] Márk Jelasity, *Gossip-based Protocols for Large-scale Distributed Systems*, DSc Dissertation, Szeged, 2013.