

Un approccio basato su sistemi cyberfisici per la gestione intelligente delle risorse energetiche in reti di punti di ricarica per veicoli elettrici

Ciccotelli Gabriele⁺, Iuliano Roberto^{*},

Abstract—In un contesto in cui la diffusione dei veicoli elettrici sta aumentando rapidamente, la necessità di una gestione efficiente delle risorse energetiche è diventata cruciale. Questo studio propone una soluzione per ottimizzare l'erogazione di potenza nelle stazioni di ricarica dei veicoli elettrici, basata sull'integrazione di algoritmi di elezione del leader e di calcolo distribuito. La soluzione sviluppata e presentata in questo lavoro rappresenta un'applicazione concreta dei concetti alla base della modellazione, e analisi di sistemi cyberfisici (CPS), al fine di migliorare l'efficienza delle infrastrutture di ricarica, con potenziali benefici in termini di sostenibilità e gestione intelligente delle reti energetiche.

I. INTRODUZIONE

La crescente adozione di veicoli elettrici (EV) pone nuove sfide alla gestione delle risorse energetiche, in particolare nella gestione dei flussi di potenza destinati alle stazioni di ricarica. È quindi essenziale garantire che la potenza sia distribuita in modo efficiente e proporzionale tra le diverse stazioni, per evitare sovraccarichi e assicurare tempi di ricarica accettabili per tutti gli utenti.

Per affrontare questo problema, sono state esplorate diverse soluzioni che sfruttano il potenziale dei sistemi cyberfisici, sistemi che combinano processi fisici ed elementi computazionali per risolvere problemi complessi. In questo contesto, si propone un approccio basato su algoritmi di *leader-election* e di *distributed computing*, per ottimizzare la distribuzione della potenza tra le stazioni di ricarica.

Il metodo proposto sfrutta un particolare algoritmo di *leader-election*, noto come *Bully algorithm*. L'obiettivo di tale algoritmo è quello di identificare la stazione con l'identificativo univoco (UID) più alto e renderlo leader. Successivamente, sulla base di tale elezione, vengono assegnati dei valori numerici a ciascun punto di ricarica, che sono poi impiegati dall'algoritmo *Push-Sum* per il calcolo dell'aliquota di potenza da erogare, rispetto a quella fornita dal generatore, nell'istante di tempo considerato.

Nel corso del presente documento verrà analizzata in dettaglio l'implementazione della soluzione proposta, il cui scopo è garantire una gestione equa e dinamica delle risorse energetiche nelle reti di ricarica dei veicoli elettrici.

II. SCENARIO DI INTERESSE

Si considera uno scenario in cui è presente una rete costituita da n punti di ricarica (CP) per veicoli elettrici, alimentati da un generatore di potenza comune. A ciascun CP può essere collegato un solo EV per volta, che

assorbendo potenza dallo stesso, è in grado di caricare la propria batteria.

Il sistema che si vuole realizzare può essere inquadrato nell'ottica dei *cyber-physical systems* come un CPS, nonché un sistema *real-time*, ossia un sistema che deve rispondere a sollecitazioni esterne rispettando determinati vincoli temporali.

Allo stesso modo, ogni punto di ricarica va inteso come un CPS caratterizzato, quindi, anche da un modulo di rete attraverso il quale è in grado di comunicare, tramite scambio di messaggi, con gli altri punti di ricarica del sistema. Tale metodo di comunicazione permette ai CP di coordinarsi e di calcolare, entro un certo limite di tempo, le aliquote di potenza del generatore comune. In particolare, in ogni istante di tempo, per ciascun CP, questa percentuale di potenza deve essere pari al rapporto tra la potenza erogata dal generatore principale, nell'istante considerato, e il numero di CP a cui è connesso un veicolo che richiede potenza nel medesimo istante.

A tal fine, per implementare la soluzione, si è pensato di sfruttare i seguenti due algoritmi, ampiamente noti in letteratura nell'ambito dei contesti distribuiti:

- L'algoritmo del Bullo, tipicamente utilizzato per l'elezione del leader in una rete di nodi computazionali;
- L'algoritmo *Push-Sum*, solitamente impiegato per il calcolo di aggregati, come la media.

L'algoritmo *Push-Sum* è stato utilizzato al fine di calcolare in maniera distribuita le sopracitate aliquote di potenza. Più precisamente, tale algoritmo permette di calcolare la percentuale della potenza complessiva erogabile da ogni CP, sulla base del relativo stato e del numero totale di CP attivi in quell'istante di tempo. A tal fine, è necessario che si parta da una condizione in cui, un solo nodo della rete dei CP attivi sia caratterizzato da un valore numerico pari a 1, mentre gli altri da un valore nullo. Quindi, tramite l'algoritmo *Push-Sum* viene calcolata la media di questi valori. Tale media deve essere ricalcolata ogni qualvolta si verifichi un cambiamento nello stato della rete (e.g., nel caso in cui un veicolo si colleghi o si disconnetta, oppure quando la batteria di un veicolo raggiunge la carica completa).

Per soddisfare la suddetta condizione iniziale è stato sfruttato l'algoritmo del Bullo, con il quale è stato possibile associare al nodo leader della rete il valore numerico 1, mantenendo i valori nulli negli altri nodi.

A. Elicitazione dei requisiti

Emerge la necessità di individuare un sistema, che ha l'obiettivo di gestire in modo efficiente la distribuzione di potenza tra diversi punti di ricarica per veicoli elettrici, alimentati da un generatore comune. In particolare, il sistema deve soddisfare i seguenti vincoli:

⁺Studente del corso di *Analisi e Controllo di Sistemi Cyberfisici*, del prof. L. Iannelli, DING, Università degli Studi del Sannio, email: gabrieleciccotelli98@gmail.com

^{*}Studente del corso di *Analisi e Controllo di Sistemi Cyberfisici*, del prof. L. Iannelli, DING, Università degli Studi del Sannio, email: r.iuliano@studenti.unisannio.it

- Gestione dei punti di ricarica: il sistema deve supportare una rete di punti di ricarica, ciascuno dei quali è in grado di servire un singolo veicolo elettrico alla volta. Solo i CP attivi, cioè quelli a cui è collegato un veicolo che richiede potenza, devono assorbire energia dalla rete;
- Distribuzione della potenza: la potenza del generatore deve essere ripartita equamente tra i punti di ricarica a cui è collegato un veicolo che richiede potenza. In ogni istante, la potenza fornita a ciascuno dei suddetti CP deve essere pari alla potenza totale disponibile, divisa per il numero di CP a cui è collegato un veicolo in carica;
- Coordinamento tra i punti di ricarica: ogni qualvolta si verifichi un cambiamento nello stato della rete, i CP attivi devono comunicare per ricalcolare automaticamente le quote di potenza, adattandosi alle nuove condizioni;
- Condizioni di ricarica:
 - Un veicolo può ricevere potenza solo se è collegato a un punto di ricarica;
 - Un veicolo con la batteria completamente carica non deve continuare a ricevere potenza;
 - Se la batteria di un veicolo connesso a un CP raggiunge il massimo livello di carica (SoC - *State of Charge*), non deve essere erogata potenza al veicolo finché il SoC della batteria non scende al di sotto del 95%;
 - Se un punto di ricarica è guasto, non deve poter erogare potenza.
- Limitazione della potenza totale: in ogni istante di tempo, la somma delle potenze erogate da tutti i punti di ricarica attivi non deve mai superare la potenza complessiva fornita dal generatore.

B. Definizione informale dei requisiti

Dall'analisi del paragrafo precedente, derivano le seguenti specifiche funzionali descritte in linguaggio naturale:

- R1: un veicolo elettrico non può richiedere potenza se non è connesso a un punto di ricarica;
- R2: se la batteria di un veicolo è carica, quest'ultimo non può richiedere potenza;
- R3: se la batteria ha un livello di carica compreso tra il 95% e il 100% e precedentemente questo era uguale al 100%, allora non potrà richiedere potenza finché il SoC non scende al di sotto del 95%;
- R4: il punto di ricarica può erogare potenza solo se ha un veicolo collegato;
- R5: il punto di ricarica non può erogare potenza se è guasto;
- R6: il punto di ricarica eroga potenza al veicolo, solo se quest'ultimo non ha la batteria carica;
- R7: in qualsiasi istante di tempo, la somma delle potenze erogate dai punti di ricarica deve essere maggiore o uguale a zero e non deve mai superare il valore di potenza fornito dal generatore;

- R8: in qualsiasi istante di tempo, la somma delle potenze erogate dai punti di ricarica ai quali non sono connessi veicoli deve essere nulla;
- R9: entro 4s dall'ultima variazione della richiesta di potenza di un veicolo, ogni CP attivo e non guasto deve erogare una potenza uguale a quella del generatore, diviso il numero di punti di ricarica attivi.

III. IMPLEMENTAZIONE DELLA SOLUZIONE

La soluzione ideata è imperniata sui seguenti meccanismi:

- A ciascun CP (o nodo) è associato un UID e un valore numerico v_i inizializzato a 0, che sarà successivamente utilizzato per calcolare le aliquote di potenza attraverso l'algoritmo *Push-Sum*:

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

- I nodi a cui è connesso un veicolo che richiede potenza, costituiscono l'insieme CPs_active ;
- Solo i nodi appartenenti a CPs_active partecipano al processo di elezione del leader, gestito come indicato nel punto successivo;
- L'algoritmo del Bullo viene utilizzato per individuare $cp_{leader} \in CPs_active$. Ossia, per individuare il nodo con UID maggiore e renderlo leader. Tale processo viene implicitamente sfruttato per impostare la variabile $v = 1$ del nodo cp_{leader} ;
- L'algoritmo di elezione viene avviato ogni qual volta si verifica un'alterazione dello stato della rete dei punti di ricarica, dettata dai seguenti eventi:
 - Un EV si disconnette dal CP a cui era connesso;
 - Un EV si connette a un CP libero e richiede potenza;
 - La batteria di un EV connesso e in carica giunge nello stato di carica completa.

Immediatamente dopo il verificarsi dei sopraelencati eventi, tutte le variabili v_i sono azzerate.

Dai punti precedenti si evince che l'algoritmo del Bullo è sfruttato per fare in modo che, dopo il verificarsi di una delle condizioni sopraelencate, entro determinati limiti temporali, tra i nodi appartenenti a CPs_active sia presente un unico nodo con $v = 1$, ossia il leader. Appare anche evidente che:

- Se $CPs_active = \emptyset$, il processo di elezione non è avviato e tutte le variabili v_i rimangono uguali a 0.

Sotteso a tale processo è continuamente in esecuzione l'algoritmo *Push-Sum*, utilizzato per calcolare le aliquote di potenza destinate a ciascun $cp_i \in CPs_active$ e, quindi, per permettere ai nodi della rete di raggiungere il consenso. Tale algoritmo, per raggiungere il suo obiettivo, sfrutta le variabili v_i assegnate ai CP nei punti precedenti. Raggiunto il consenso, ciascun EV può assorbire potenza dal CP al quale è connesso, a meno che quest'ultimo non sia in uno stato di errore.

I meccanismi elencati evidenziano come si voglia realizzare un sistema, che eviti sovraccarichi e assicuri una distribuzione equa dei flussi di potenza verso i veicoli, che ne fanno richiesta.

IV. MODELLO DEL SISTEMA

A partire dallo scenario e dall'idea descritti nelle sezioni precedenti, si è proceduto con la creazione di un modello. Un modello è una rappresentazione del sistema, utile per valutare le proprietà dello stesso. E' considerabile una buona astrazione del sistema solo se omette esclusivamente i dettagli non essenziali dello stesso.

Considerato che il sistema individuato coniuga una dinamica continua con una discreta, lo stesso viene considerato un sistema ibrido (o modale). Con tale accezione, si vuole evidenziare il fatto che ci siano caratteristiche discrete e continue che coesistono. Cioè, il sistema è caratterizzato da stati discreti, che prendono il nome di "modi", e da variabili per lo più continue, dalle quali dipende l'evoluzione del sistema stesso. La parte continua del sistema è rappresentata dai processi fisici, ossia la carica e la scarica della batteria dei veicoli elettrici, e l'erogazione di potenza da parte dei CP ai veicoli. Tuttavia, gli eventi che caratterizzano il sistema e che ne comportano i cambiamenti di stato, come la connessione o la disconnessione dei veicoli, sono eventi discreti, che si possono descrivere con una macchina a stati. Sulla base di quanto affermato, al fine di modellare tale sistema ibrido sono state sfruttate le macchine modali. Quindi, appare evidente che il sistema considerato sia un sistema ibrido *event triggered*.

Il processo di modellazione si è sviluppato sfruttando un consolidato principio ingegneristico: i sistemi complessi possono essere descritti come composizioni di sistemi più semplici [4]. Per cui, la prima fase di tale processo ha previsto la scomposizione del dominio in più sottosistemi e la relativa modellazione.

A. Modello della batteria: Coulomb Counting Method

Il *Coulomb Counting Method*, noto anche come *Ampere - hour (Ah) Counting Method*, è una tecnica comunemente utilizzata per stimare lo stato di carica di una batteria, cioè la quantità di carica che entra ed esce da una batteria nel tempo [1]. Tuttavia, non è uno dei modelli più accurati poiché ci sono diversi fattori che affliggono il suo livello di precisione, come temperatura e dispersione di corrente [2].

Data l'equazione (3), derivando entrambi i termini rispetto al tempo, si ottiene l'equazione (4).

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

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

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

Con:

- $\text{SoC}(t_0)$: stato di carica iniziale;
- $C_{\text{rated}} [Ah]$: capacità della batteria;
- $I(t) [A]$: corrente di carica/scarica all'istante di tempo corrente.

Considerando anche le seguenti costanti adimensionali:

- η_{charge} : efficienza di carica della batteria;
- $\eta_{\text{discharge}}$: efficienza di scarica della batteria.

Risulta l'equazione che descrive il fenomeno di carica (5) e quella relativa al fenomeno di scarica (6):

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

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

Al fine di esprimere la capacità della batteria C_{rated} in Wh , è necessario che al numeratore sia presente una potenza espressa in W . Le due equazioni finali relative al fenomeno di carica e di scarica, quindi, risultano essere rispettivamente:

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

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

Le due equazioni differenziali sono state modellate in Simulink sfruttando i blocchi guadagno e integratore. In entrambe le equazioni, si è optato per un valore di tensione nominale pari a $V_{\text{nom}} = 230 [V]$ e per un valore delle costanti di efficienza pari a $\eta_{\text{charge}} = \eta_{\text{discharge}} = \frac{9}{10}$ [3]. Considerato che si ha interesse nel modellare esclusivamente i processi di carica e scarica quando il veicolo è fermo e connesso al punto di ricarica, nell'equazione relativa al fenomeno di scarica, si è scelto un valore di corrente $I_{\text{discharge}} = -5 [A]$. I modelli sono riportanti in fig. 1 e fig. 2.

B. Modello del punto di ricarica

Dopo aver derivato, dallo scenario considerato, le proprietà di interesse del sistema punto di ricarica, lo stesso è stato rappresentato sfruttando una macchina modale. In particolare, una macchina a stati finiti (FSM) non deterministica.

1) *Stati del sistema*: In primo luogo, analizzando lo scenario, relativamente a un CP, sono stati individuati i seguenti macro-stati di funzionamento:

- Stato di funzionamento normale (*Normal state*), in cui il sistema si trova quando non si verificano *fault*, ossia condizioni che determinano la transizione a uno stato di errore;
- Stato di errore (*Error state*), che rappresenta una condizione per cui il sistema risulti guasto.

Sulla base di ciò, lo stato di funzionamento normale è stato raffinato per comprendere quali fossero gli aspetti di interesse da modellare. A tal fine, sono stati individuati i seguenti stati:

- Stato libero (*Idle state*), che rappresenta la condizione per cui al CP non è connesso alcun veicolo. Il CP è libero e a esso può collegarsi, in un qualsiasi istante di tempo, un veicolo elettrico qualunque;
- Stato connesso (*Connected state*), al punto di ricarica è connesso un EV.

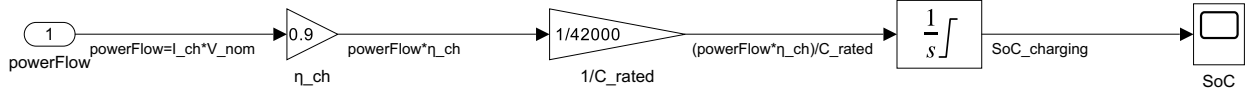


Fig. 1: Schema Simulink del modello della batteria *Coulomb Counting Method* (fase di carica).

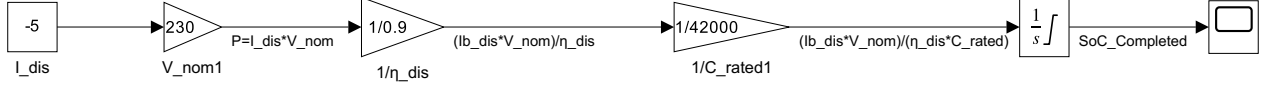


Fig. 2: Schema Simulink del modello della batteria *Coulomb Counting Method* (fase di scarica).

Infine, si è intuito che quando un EV è connesso al punto di ricarica, questo può richiedere o meno potenza al CP stesso. Infatti, un veicolo che completa la carica della propria batteria ($SoC = 1$) può continuare a essere connesso al CP, finché l'utente non disconnette il veicolo. Sulla base di tale osservazione, lo stato "connesso" è stato ulteriormente raffinato e sono stati individuati i seguenti stati:

- In alimentazione (*Powering state*): il CP si trova in tale stato quando eroga potenza al veicolo connesso;
- Non in alimentazione (*NotPowering state*): il CP si trova in tale stato quando non eroga potenza al veicolo connesso. E.g., quando il veicolo completa la sua carica e continua a essere collegato al CP.

Da come sono stati concepiti gli stati, appare evidente che il sistema possa essere modellato attraverso una macchina a stati gerarchica. Cioè sfruttando il concetto di *state refinement*.

L'*actor model* di tale sistema e la relativa macchina a stati sono riportati in fig. 3. La sua implementazione in Simulink Stateflow è riportata in fig. 4.

2) *Segnali e transizioni tra stati*: Dall'*actor model* del sistema punto di ricarica, riportato in fig. 3, si evince che i segnali caratterizzanti il sistema sono i seguenti:

- Segnale *pReq*, che è un segnale tempo continuo di ingresso.

$$pReq \in \mathbb{B}^{\mathbb{R}^+} \quad (9)$$

Il segnale rappresenta la richiesta di potenza da parte del veicolo. Questo assume valore pari a 1 quando il veicolo richiede potenza al CP e 0 altrimenti;

- Segnale *plug*, che è un segnale tempo continuo di ingresso.

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

Il segnale rappresenta la connessione di un veicolo al punto di ricarica. Questo assume valore pari a 1 quando un veicolo è connesso al CP, 0 in caso contrario;

- Segnale *enablePowerFlow*, che è un segnale tempo continuo di uscita.

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

Il segnale rappresenta l'abilitazione all'erogazione di potenza da parte del CP verso il veicolo connesso. Il segnale assume valore pari a 1 quando l'erogazione

è abilitata, 0 altrimenti. Tale segnale è stato poi utilizzato come segnale di controllo per abilitare l'erogazione di un'aliquota di potenza, calcolata dalla componente che si serve dell'algoritmo *Push-Sum*.

Il sistema complessivo opera una trasformazione sui due segnali di ingresso, per cui può essere visto come una funzione:

$$ChargingPoint : (\mathbb{B}^{\mathbb{R}^+})^2 \rightarrow \mathbb{B}^{\mathbb{R}^+} \quad (12)$$

Le transizioni tra gli stati sono pilotate dai segnali di ingresso. Il segnale di uscita varia a seconda del modo corrente in cui opera il sistema. I dettagli relativi a tali transizioni e ai modi sono riportati in fig. 3 e 4.

3) *Comportamento non deterministico*: Nel modello del punto di ricarica, il non determinismo è introdotto dalla possibilità di transitare in modo "non deterministico" dallo stato *Normal* allo stato di errore e viceversa. In altre parole, il veicolo può continuare a funzionare nello stato *Normal* oppure, in maniera indeterminata, passare allo stato *Error*.

Da fig. 3 si evince come la macchina a stati del sistema sia una macchina non deterministica. Infatti, per lo stato *Normal* e per lo stato *Error* sono presenti due distinte transizioni, con guardie che possono essere entrambe valutate come vere nella stessa reazione. Nel diagramma di fig. 3, le transizioni che rendono la macchina non deterministica sono colorate in rosso.

Questo comportamento si traduce, di fatto, nella presenza di una *self transition* con una guardia *true* sullo stato *Normal* e una transizione, anch'essa con una guardia *true*, che permette il passaggio dallo stato di funzionamento normale allo stato di errore. Discorso analogo per lo stato *Error*.

In fase di simulazione, in Simulink Stateflow, si è fatto in modo che la condizione associata alla transizione tra *Normal* ed *Error* sia valutata su base probabilistica. Infatti, da fig. 4 si evince come la transizione che va dallo stato *Normal* a *Error* abbia la seguente guardia:

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

Listing 1: Guardia della transizione tra lo stato di funzionamento normale e quello in errore nell'implementazione del modello punto di ricarica.

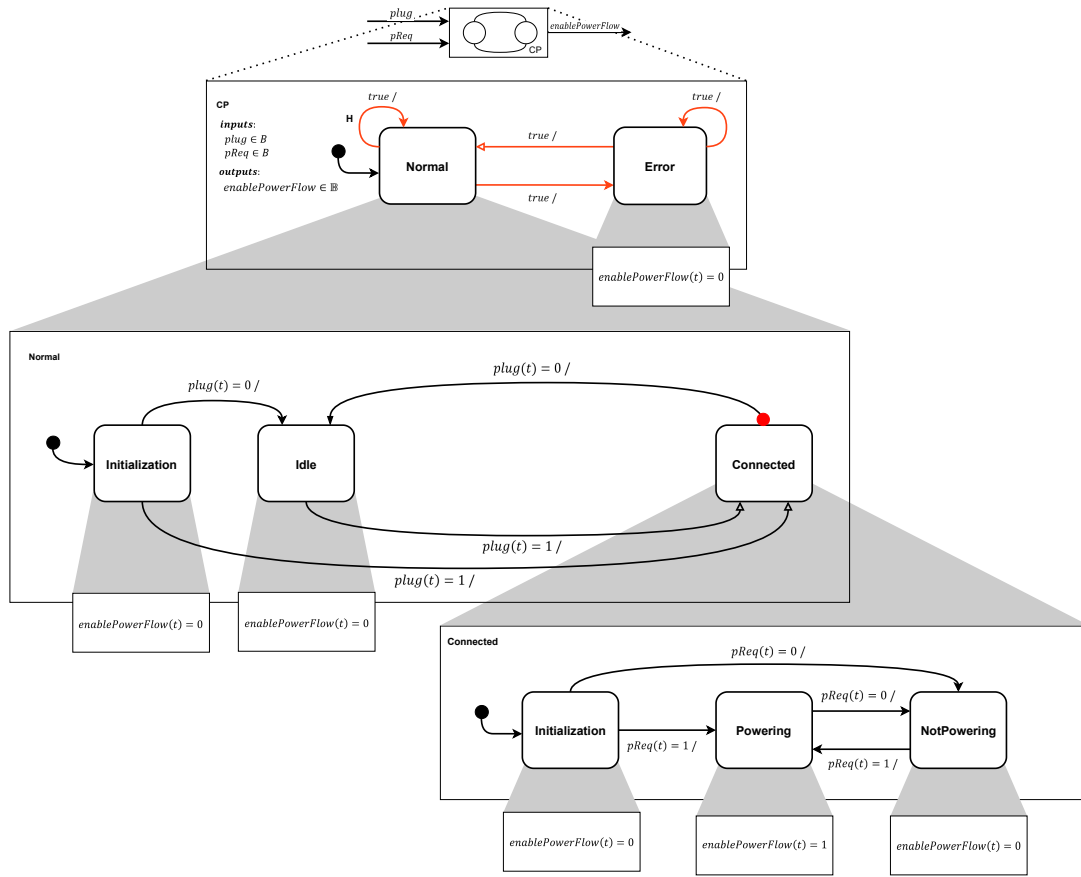


Fig. 3: Modello del sistema punto di ricarica. In alto l'*actor model*. Nella parte sottostante è rappresentata la macchina a stati finiti gerarchica e i relativi raffinamenti dei modi. Le transizioni che rendono la macchina non deterministica sono riportate in rosso. La *self transition* sullo stato *Normal* rappresenta una transizione storica (etichettata con *H*). Nello *state refinement* dello stato *Normal*, la transizione che va dallo stato *Connected* allo stato *Idle* è una *preemptive transition*.

Dalla guardia riportata nella porzione di codice 1 si evince che:

- Invocando la funzione `runPythonScriptRandGen()` viene invocato uno script Python che ritorna un valore compreso tra 0 e 1 (estremi inclusi) con una funzione di densità di probabilità (PDF) uniforme. Ciò significa che, a ogni invocazione della funzione, si ha la stessa probabilità di ottenere un qualsiasi numero incluso nell'intervallo $[0, 1]$. Per la generazione del numero casuale è stata invocata una funzione esterna a MATLAB, in quanto gli strumenti messi a disposizione dal *tool* non consentono di includere gli estremi nella generazione di tali valori casuali;
- Il valore ricavato dall'invocazione della funzione `runPythonScriptRandGen()` viene confrontato con quello restituito dalla funzione 13. La funzione `elapsed(sec)` restituisce il tempo espresso in secondi, dall'ultimo ingresso nello stato di funzionamento normale.

$$y = 1 - e^{-\frac{elapsed(sec)}{300}} \quad (13)$$

Dalla fig. 5 si evince come, al variare del valore del denominatore della funzione 13, vari la rapidità con cui la funzione si avvicina al valore limite di 1. Per motivi legati ai tempi di simulazione, nell'implementazione del modello, il denominatore della funzione è stato fissato a 300;

- Per come è concepita la guardia considerata, più passa il tempo, più aumenta la probabilità di transire nello stato di errore. A tal proposito, si faccia riferimento alla fig. 6. Nella generazione di tale grafico sono stati raccolti 100 campioni uniformi tra 0 e 1.

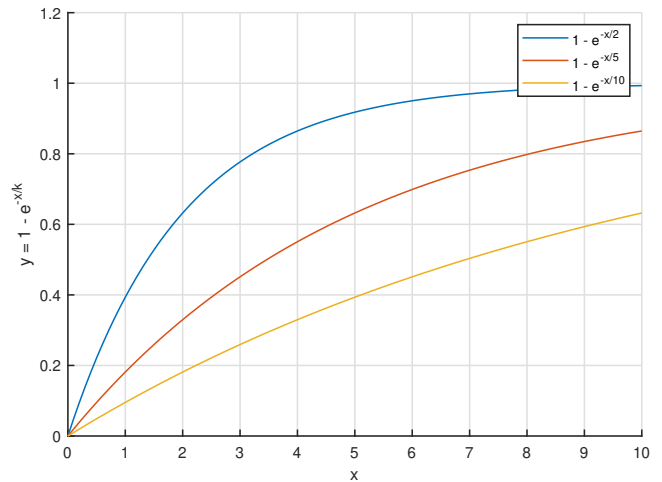


Fig. 5: Grafico della funzione esponenziale $y = 1 - e^{-x/k}$ per diversi valori di k : $k = 2$, $k = 5$, e $k = 10$. Le curve mostrano come il parametro k influisca sul comportamento della funzione esponenziale, evidenziando la rapidità con cui la funzione si avvicina al valore limite di 1.

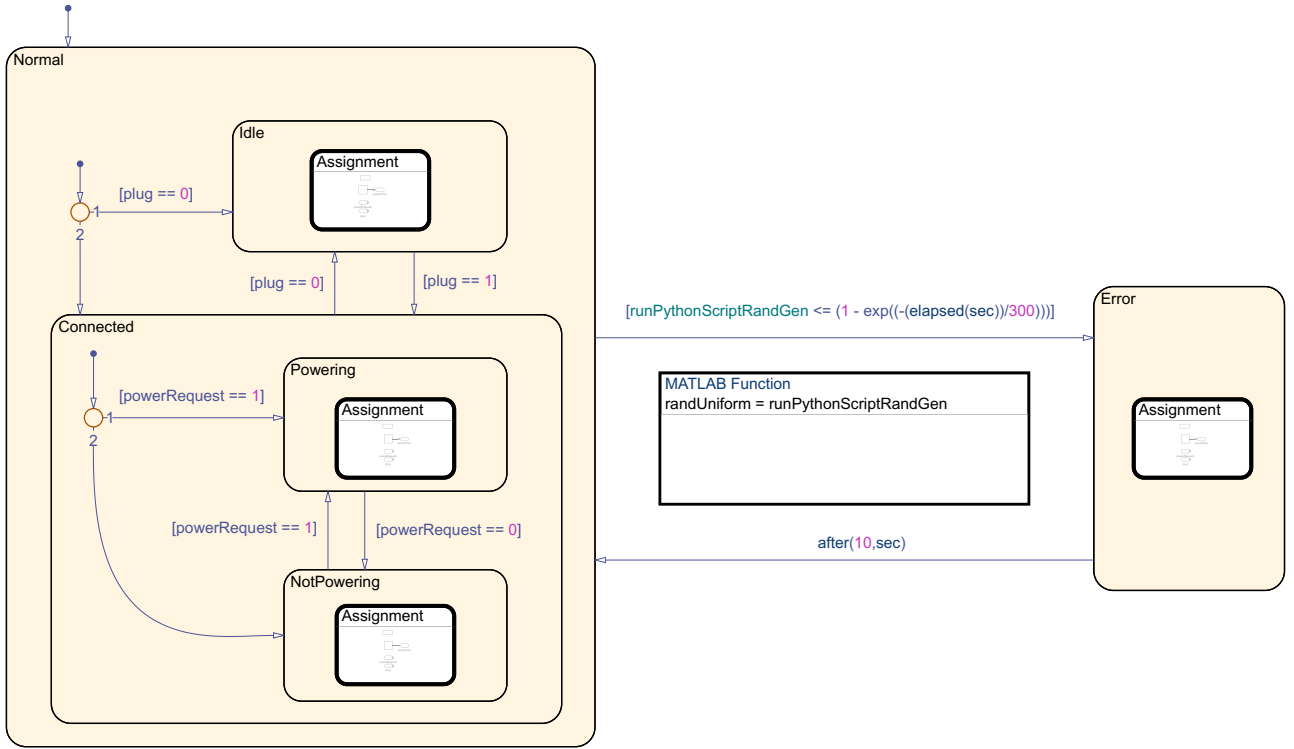


Fig. 4: Implementazione in Simulink & Stateflow del modello del sistema punto di ricarica.

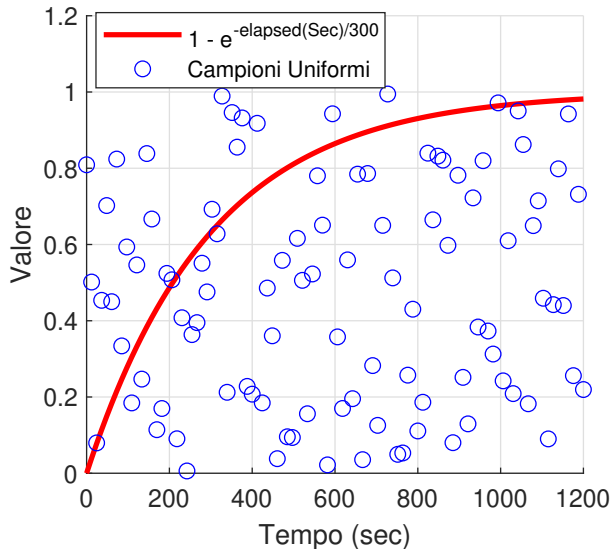


Fig. 6: Confronto tra campioni uniformi e la funzione esponenziale $y = 1 - e^{-\frac{elapsed(sec)}{300}}$. La curva rossa rappresenta la funzione esponenziale che cresce rapidamente per valori crescenti di tempo, mentre i punti blu rappresentano campioni generati in modo uniforme tra 0 e 1. Questo grafico illustra la probabilità di attivazione della transizione di stato (da **Normal** a **Error**) nel modello di macchina a stati CP, a seconda del tempo trascorso.

4) *Semantica della composizione gerarchica, macchina equivalente e riduzione d'ordine*: Relativamente alla composizione gerarchica riportata in fig. 3, si sottolinea che:

- Il raffinamento dello stato **Normal** è espresso attraverso una semantica *depth-first*, cioè reagisce prima

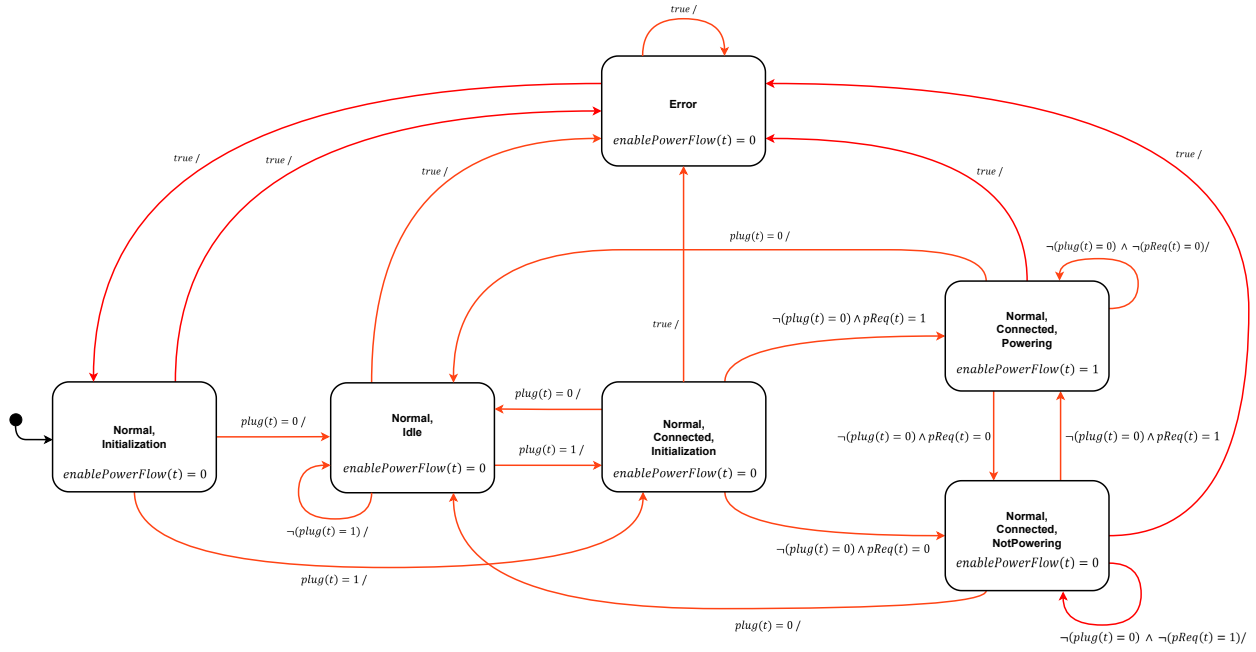
del suo stato container;

- Per il raffinamento dello stato **Connected**, è stata utilizzata una semantica diversa. Infatti, la transizione in uscita da tale stato è una *preemptive transition*. La guardia della *preemptive transition* è valutata prima che il raffinamento reagisca, se tale guardia è valutata come vera, allora il raffinamento non reagisce. Tale semantica serve a dare priorità alle variazioni del segnale *plug*, rispetto al segnale *pReq*;
- La *self transition* sullo stato **Normal**, è una transizione storica. Ciò significa che, quando è abilitata, il raffinamento di destinazione riprende dallo stato in cui si trovava precedentemente (o dal suo stato iniziale al primo ingresso);
- Le seguenti transizioni:
 - **Error** → **Normal**;
 - **Idle** → **Connected**;
 - (**Normal**, **Initialization**) → **Connected**;

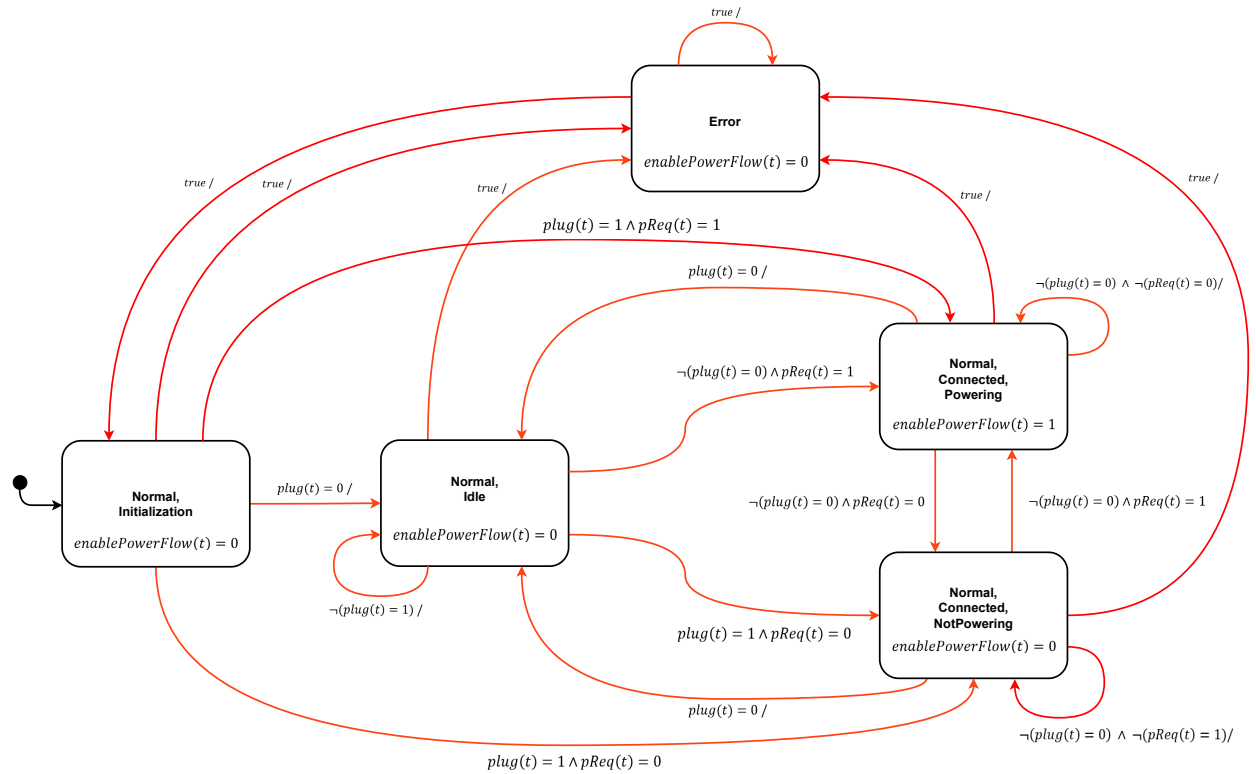
Sono rappresentate con una freccia dalla punta cava, in quanto trattasi di *reset transition*. Nel caso vengano percorse, infatti, il raffinamento di destinazione è azzerato al suo stato iniziale.

Il significato di tale gerarchia può essere compreso confrontando la macchina a stati gerarchica di fig. 3 con la macchina *flattened*, equivalente, riportata in fig. 7a.

Siccome lo stato **Initialization** della macchina riportata in fig. 7a non possiede informazioni di stato e non influenza il valore delle uscite, allora è stata effettuata una riduzione d'ordine. Lo stato **Initialization** collapsa nello stato **Idle**. La macchina ottenuta in seguito alla riduzione d'ordine è riportata in fig. 7b.



(a) FSM equivalente (*flattened*) del modello del sistema punto di ricarica, riportato in fig. 3, ottenuta risolvendo la composizione gerarchica. Con la notazione "Normal, Initialization", "Normal, Idle", "Normal, Connected, Initialization", "Normal, Connected, Powering" e "Normal, Connected, NotPowering" si indica che la macchina a stati si trova nel primo stato indicato e più precisamente nello stato del raffinamento indicato nelle posizioni due e tre (se presente). Ad esempio, considerare lo stato "Normal, Connected, Initialization" significa dire che la macchina a stati si trova nello Stato Normal, più precisamente nello stato Connected del suo raffinamento, e nello stato Initialization del suo ulteriore raffinamento.



(b) FSM equivalente (*flattened*) del modello del sistema punto di ricarica, riportato in fig. 3. Viene operata la riduzione d'ordine sulla macchina equivalente riportata in fig. 7a. Per la notazione utilizzata nel nominare gli stati si faccia riferimento alla didascalia di fig. 7a.

Fig. 7: FSM equivalenti del modello del sistema punto di ricarica.
Relativamente alle caratteristiche dei segnali di ingresso e di uscita delle FSM si faccia riferimento a fig. 3.

Dall'implementazione riportata in fig. 4, si evince come lo stato (*Normal*, *Initialization*) della macchina equivalente realizzata in fase di modellazione (fig. 7), durante la fase d'implementazione sia collassato nel blocco *connective junction* dello stato *Normal* (fig. 4). Lo stesso è avvenuto per lo stato (*Normal*, *Connected*, *Initialization*) (fig. 7a), il quale è stato implementato attraverso il punto di giunzione dello stato *Connected*.

Il tool Stateflow, infatti, mette a disposizione soluzioni come quella appena vista, che permettono di gestire al meglio la complessità del sistema, evitando, in questo caso, la creazione degli stati di inizializzazione previsti nel modello.

Più precisamente, attraverso questo "decision point", è possibile valutare quale stato del raffinamento raggiungere non appena si è entrati nello stato container. Infatti, il blocco in questione permette di verificare quale delle due transizioni uscenti abilitare, sulla base delle rispettive guardie.

Ad esempio, in fig. 4, sulla base delle priorità assegnate, viene prima valutata la guardia della transizione verso lo stato *Idle*. Se tale guardia ($plug==0$) è verificata, la transizione viene abilitata e la macchina passa nello stato *Idle*. In tutti gli altri casi, viene abilitata l'altra transizione, ossia quella verso lo stato *Connected*.



Fig. 8: Combinare transizioni e giunzioni per creare percorsi ramificati. Blocco *connective junction*.

5) *Caratterizzazione del modello*: Per come è stato concepito il modello del sistema punto di ricarica, tale macchina a stati risulta essere:

- *Event triggered*, poiché la dinamica del sistema (evoluzione del suo stato) è guidata da eventi. Il sistema, infatti, viene sollecitato da eventi come il collegamento o lo scollegamento del veicolo e dalle richieste di potenza da parte del veicolo stesso;
- Non deterministica, in quanto la *update function* è una funzione *set valued* (funzione multivalore). Infatti, a partire dallo stato *Normal*, a parità di ingresso, si può transitare nello stato *Error* o rimanere, tramite una *self transition*, nello stato corrente. Discorso analogo per lo stato *Error*;
- Una macchina modale, ossia il modello di un sistema ibrido. Infatti, il modello del sistema considerato coniuga:
 - Dinamica discreta: quella relativa agli eventi, che caratterizzano il sistema e che ne comportano i cambiamenti di stato;
 - Dinamica continua: quella relativa ai segnali continui, che regolano l'evoluzione del sistema. Infatti, i segnali in ingresso e in uscita dal sistema sono segnali che evolvono nel tempo continuo, di conseguenza, la loro valutazione

e generazione avviene in maniera continua nel tempo. Quindi, il sistema continua a vivere quando si trova in uno stato, secondo quanto specificato nel modo corrente.

C. Modello del veicolo elettrico

Nel definire il modello del veicolo elettrico, si sono considerate le proprietà di interesse nel contesto analizzato. Il sistema EV è stato rappresentato mediante una macchina modale che, come si vedrà nelle sezioni successive, è caratterizzato da modi (ossia gli stati discreti, anche detti *bubbles*) e da variabili di stato continue.

1) *Stati del sistema*: Nella fase iniziale della costruzione del modello, si è proceduto con l'identificazione degli stati. In primo luogo, è stato definito uno stato di funzionamento normale, che rappresenta una condizione di operatività standard del veicolo. Lo stesso è stato introdotto per rendere il modello flessibile, consentendo, in sviluppi futuri, l'aggiunta di uno stato di funzionamento in errore, caratterizzandolo secondo le esigenze del contesto. Successivamente, lo stato di funzionamento normale è stato scomposto in:

- *Unplugged*: condizione in cui il veicolo elettrico non è connesso al punto di ricarica;
- *Plugged*: condizione in cui il veicolo è connesso al CP.

Anche lo stato *Plugged* è stato a sua volta scomposto, identificando al suo interno:

- *Completed*: stato in cui la batteria del veicolo è completamente carica;
- *Charging*: stato in cui la batteria è in fase di carica.

La transizione tra questi stati è gestita in base allo stato di carica della batteria del veicolo stesso.

Successivamente, è stato considerato un raffinamento degli stati *Completed* e *Charging*, il quale è stato sviluppato sulla base del modello della batteria (*Coulomb Counting Method*), a cui si è fatto riferimento nei capitoli precedenti. Per come sono stati concepiti gli stati, si deduce che la SM rappresentate il sistema è una macchina gerarchica.

L'actor model di tale sistema e la relativa macchina a stati sono riportati in fig. 9. La sua implementazione in Simulink Stateflow è riportata in fig. 10.

2) *Segnali e transizioni tra stati*: Da come si evince nell'actor model riportato in fig. 9, i segnali di ingresso e uscita che caratterizzano il sistema sono i seguenti:

- Segnale $pReq$, che è un segnale di uscita, tempo continuo.

$$pReq \in \mathbb{B}^{R+} \quad (14)$$

- Segnale di uscita $plug$, anch'esso tempo continuo.

$$plug \in \mathbb{B}^{R+} \quad (15)$$

- Segnale di ingresso $pFlow$, che indica il flusso di potenza in ingresso al veicolo. Tale segnale è tempo continuo.

$$pFlow \in \mathbb{R}^{R+} \quad (16)$$

I dettagli dei segnali $pReq$ e $plug$ sono discussi nella sezione IV-B.2.

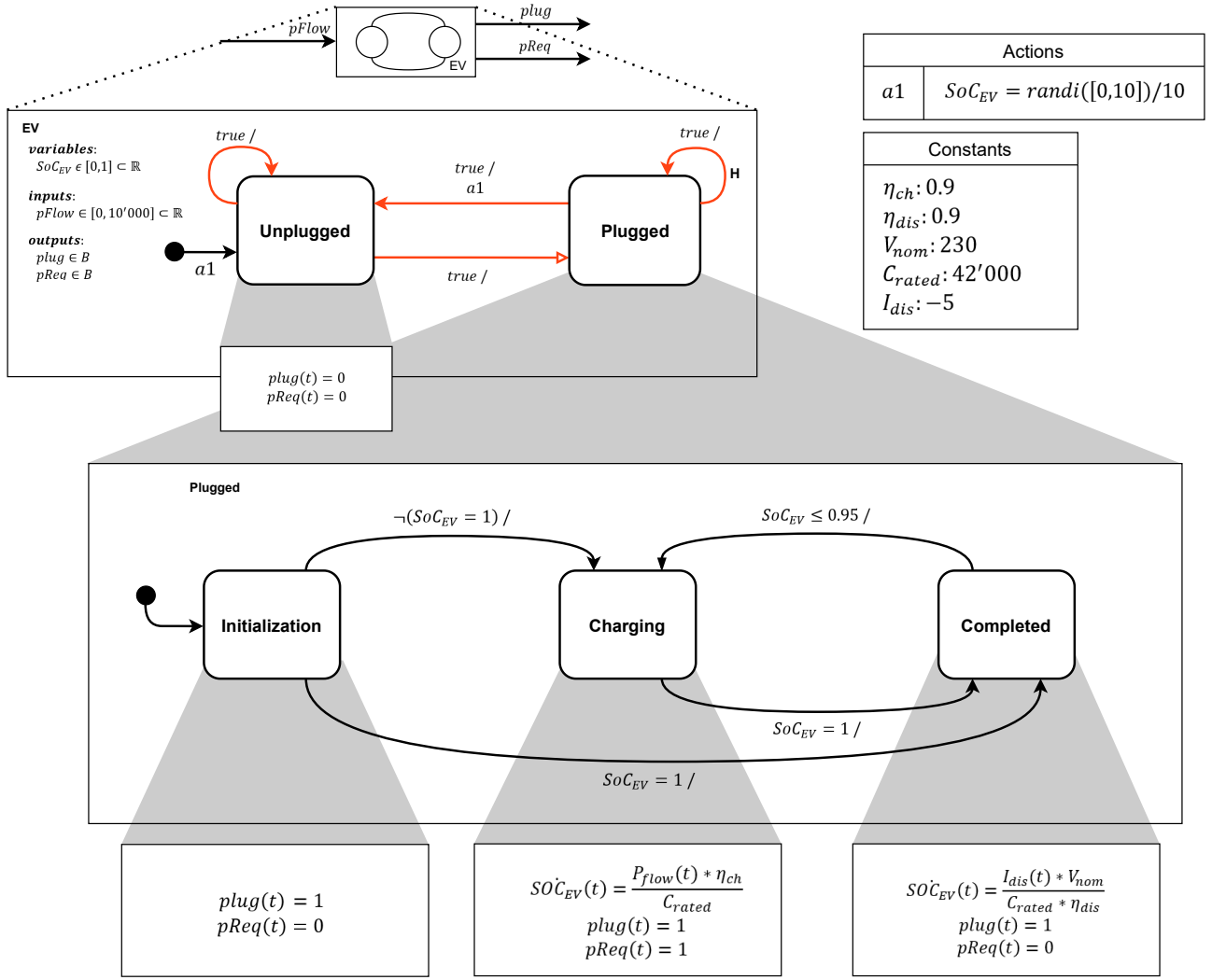


Fig. 9: Modello modale del sistema veicolo elettrico. In alto l'actor model del sistema. Nella parte sottostante è rappresentata la macchina a stati gerarchica e i relativi raffinamenti dei modi. Le transizioni che rendono la macchina non deterministica sono riportate in rosso. La self transition sullo stato Plugged rappresenta una transizione storica (etichettata con H). La transizione da Unplugged a Plugged è una reset transition.

Il sistema stesso può essere considerato come una funzione, che opera una trasformazione sul segnale di ingresso e produce i due segnali di uscita:

$$Veicolo\ elettrico: \mathbb{R}^+ \rightarrow (\mathbb{B}^+)^2 \quad (17)$$

Come descritto nella parte finale del capitolo IV-B.2, anche nell'implementazione dei modelli di fig. 12 si è optato per l'utilizzo del blocco *connective junction*.

3) *Comportamento non deterministico*: Nel modello del veicolo elettrico, il non determinismo è introdotto dall'imprevedibilità con cui un utente può collegare o scollegare il veicolo a o da un CP.

Nella macchina a stati ciò si riflette nella possibilità di transitare in modo "non deterministico" dallo stato *Unplugged* allo stato *Plugged* e viceversa. In altre parole, in ogni istante di tempo, il veicolo può sia continuare a essere connesso al punto di ricarica (stato *Plugged*), oppure, in maniera indeterminata, scollegarsi dal CP (stato *Unplugged*), il che rende il modello del sistema non deterministico.

In fase di simulazione, in Simulink Stateflow, si è fatto in modo che le condizioni associate alle transizioni tra

Plugged e *Unplugged* siano valutate su base probabilistica. Infatti, da fig. 10 si evince come la transizione che va dallo stato *Unplugged* a *Plugged* abbia la seguente guardia:

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

Listing 2: Guardia della transizione tra lo stato di funzionamento Unplugged e Plugged nell'implementazione del modello veicolo elettrico.

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

Listing 3: Guardia della transizione tra lo stato di funzionamento Plugged e Unplugged nell'implementazione del modello veicolo elettrico.

Dalla porzione di codice 3 si nota che, quando il veicolo è connesso alla colonnina, la probabilità che lo stesso si disconnetta, oltre a dipendere dal tempo trascorso da quando il veicolo si è collegato, dipenda anche dal *SoC* della sua batteria. Infatti, maggiore è il valore di *SoC* più è probabile che il veicolo si scolleghi dal CP e ne arrivi un altro.

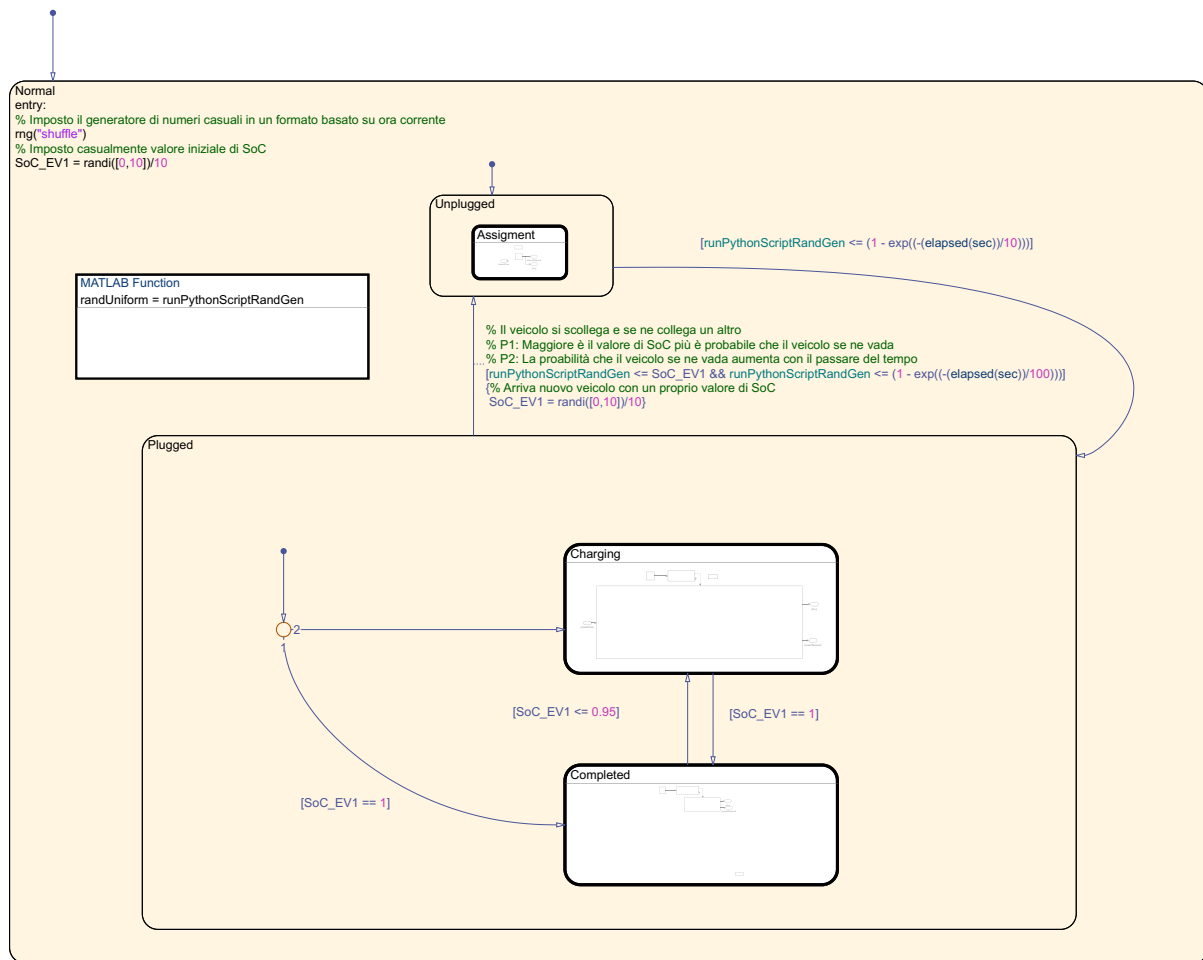


Fig. 10: Implementazione in Simulink & Stateflow del modello del sistema veicolo elettrico.

Inoltre, da fig. 10, è possibile osservare che, quando la transizione che va dallo stato *Plugged* ad *Unplugged* è abilitata, è come se si considerasse un nuovo veicolo. Infatti, il valore della variabile *SoC* viene ricalcolato in maniera casuale.

Infine, si sottolinea che la valutazione di tali probabilità può essere compresa analizzando quanto affermato nel capitolo IV-B.3, a proposito del modello del sistema punto di ricarica.

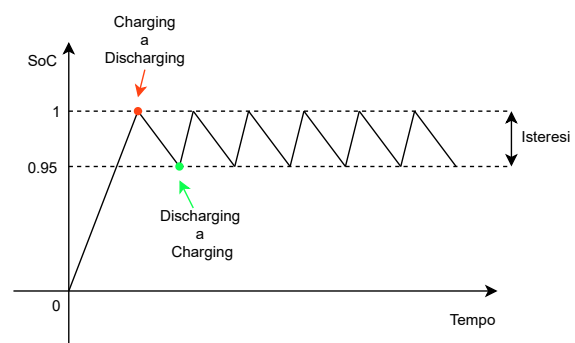


Fig. 11: Utilizzo dell'isteresi per regolare la transizione dallo stato *Completed* allo stato *Charging* nel modello del sistema veicolo elettrico.

4) *Regolare il comportamento del sistema per evitare il fenomeno del chattering:* Nel modello del sistema EV, le transizioni tra gli stati *Completed* e *Charging* vengono abilitate in base alle condizioni relative allo stato di carica della batteria. Senza l'adozione di una banda di isteresi, si verifica un fenomeno indesiderato: quando il *SoC* della batteria raggiunge il livello di carica completa nello stato *Charging*, il sistema passa immediatamente allo stato *Completed*. Tuttavia, se il livello di carica scende anche minimamente, il sistema ritorna rapidamente allo stato *Charging*, generando una continua alternanza tra i due stati. Questo fenomeno è noto come *chattering*.

Per evitare tale fenomeno indesiderato, è stata introdotta una banda di isteresi che regola il passaggio tra gli stati. In particolare, il sistema rimane nello stato *Completed* quando il *SoC* della batteria si trova tra il 95% e il 100%. Solo quando il *SoC* scende al di sotto del 95%, il sistema passa dallo stato *Completed* allo stato *Charging*. Questa banda di isteresi assicura che le transizioni tra i due stati avvengano in modo controllato, eliminando il fenomeno del *chattering* e migliorando la robustezza del sistema. A tal proposito si faccia riferimento a fig. 11.

5) *Semantica della composizione gerarchica, macchina equivalente e riduzione d'ordine*: Relativamente alla composizione gerarchica riportata in fig. 9, si sottolinea che:

- Il raffinamento dello stato *Plugged* è espresso attraverso una semantica *depth-first*, cioè reagisce prima del suo stato container;
- La *self transition* sullo stato *Plugged* è una transizione storica. Ciò significa che quando la transizione storica è abilitata, il raffinamento di destinazione riprende dallo stato in cui si trovava precedentemente;
- La transizione da *Unplugged* a *Plugged* è rappresentata con una freccia dalla punta cava, in quanto trattasi di *reset transition*. Nel caso venga percorsa, il raffinamento di destinazione è azzerato al suo stato iniziale.

Il significato di tale gerarchia può essere compreso confrontando la macchina a stati gerarchica di fig. 9 con la macchina *flattened*, equivalente, riportata in fig. 12a.

Siccome lo stato *Initialization* della macchina riportata in fig. 12a non possiede informazioni di stato e non influenza il valore delle uscite, allora è stata effettuata una riduzione d'ordine. Lo stato *Initialization* collassa nello stato *Unplugged*. La macchina ottenuta in seguito alla riduzione d'ordine è riportata in fig. 12b.

Si tenga presente che la macchina equivalente ottenuta è una macchina modale. Alcuni stati della SM, infatti, oltre a includere un'informazione relativa allo stato discreto (*bubble*) nel quale si trova la macchina, include anche il valore che possiede la *continuous state variable SoC*. In particolare, nel caso in analisi, sono presenti $n = 3$ modi (*bubbles*) e una sola variabile di stato continua *SoC*, la quale ha come dominio l'insieme dei numeri reali positivi \mathbb{R}_+ (ossia il tempo) e, come codominio, il sottoinsieme $[0, 1] \subset \mathbb{R}$. Di conseguenza, la variabile può assumere infiniti possibili valori. Quindi, gli insiemi dei modi e degli stati della macchina, sono così definiti:

$$Modi = \{Unplugged, (Plugged, Charging), (Plugged, Completed)\}$$

$$Stati = \{Unplugged, (Plugged, Charging), (Plugged, Completed)\} \times \{[0, 1] \subset \mathbb{R}\}$$

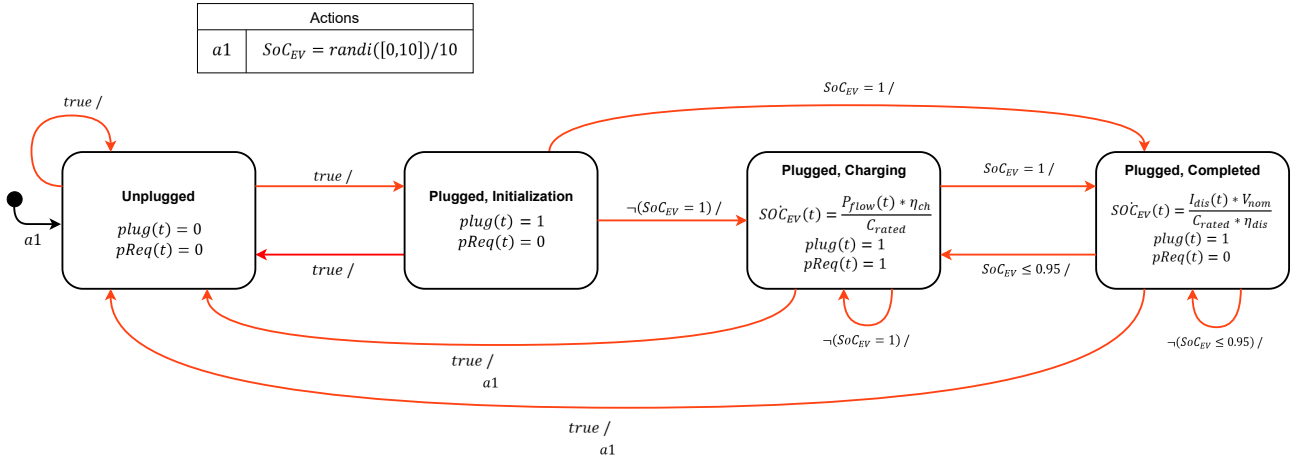
Ciò implica che $|Stati| = \infty$.

6) *Caratterizzazione del modello*: Per come è stato concepito il modello del sistema veicolo elettrico, tale macchina a stati risulta essere:

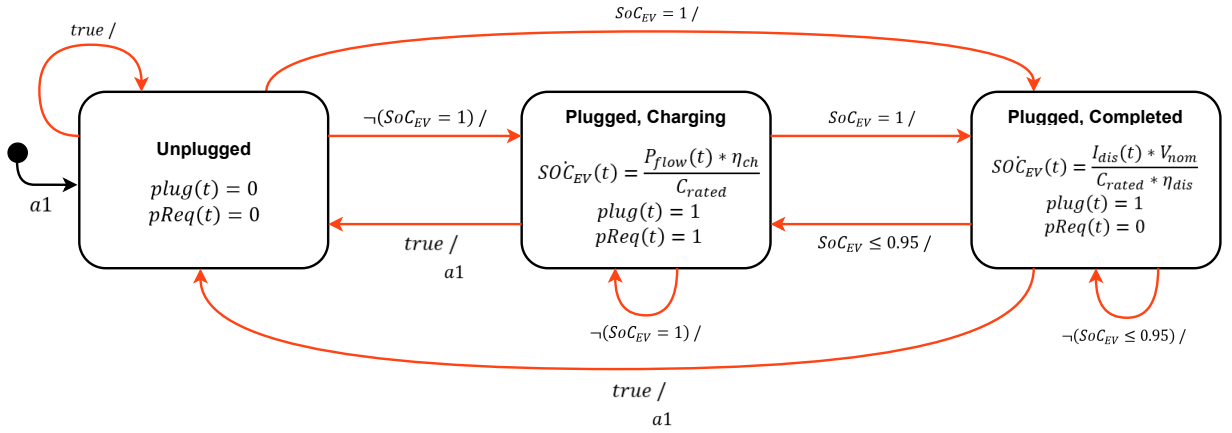
- *Event triggered*, poiché la dinamica del sistema (evoluzione del suo stato) è guidata da eventi. Il sistema, infatti, viene sollecitato da eventi, generati dal non determinismo e dalle variazioni del valore di *SoC*;
- Non deterministica, in quanto la *update function* è una funzione *set valued* (funzione multivalore). Infatti, a partire dallo stato *Unplugged* si può transitare nello stato *Plugged* o rimanere, tramite una *self transition*, nello stato corrente. Discorso analogo per lo stato *Plugged*;
- Una macchina modale, ossia il modello di un sistema ibrido. Infatti, il modello del sistema considerato coniuga dinamica continua e discreta.

Più precisamente:

- Dinamica discreta: quella relativa agli eventi, che caratterizzano il sistema e che ne comportano i cambiamenti di stato;
- Dinamica continua: quella relativa ai segnali continui, che regolano l'evoluzione del sistema e dal *Coulomb Counting Method*, utilizzato per modellare il fenomeno fisico relativo alla carica e alla scarica della batteria e, quindi, per calcolare il valore della variabile continua *SoC*. Tale modello, infatti, come discusso nella sezione IV-A si basa su l'utilizzo delle equazioni differenziali 7 e 8 che, quindi, determinano l'evoluzione del sistema nel tempo continuo.



(a) Macchina a stati equivalente (*flattened*) del modello modale del sistema veicolo elettrico, riportato in 9. Si risolve la composizione gerarchica. Per la notazione utilizzata nel nominare gli stati si faccia riferimento alla didascalia di fig. 7a.



(b) Macchina a stati equivalente (*flattened*) del modello modale del sistema veicolo elettrico, ottenuta operando la riduzione d'ordine. Per la notazione utilizzata nel nominare gli stati si faccia riferimento alla didascalia di fig. 7a.

Fig. 12: Macchine a stati equivalenti del modello modale del sistema veicolo elettrico. Relativamente alle caratteristiche dei segnali di ingresso e di uscita delle SM si faccia riferimento a fig. 9.

D. Composizione veicolo elettrico e punto di ricarica

La composizione in cascata sincrona dei due sistemi EV e CP, è mostrata in fig. 13a. La stessa è realizzata attraverso i segnali di uscita del sistema EV: *plug* e *pReq*, i quali fungono da ingressi per il sistema CP. Le due porte di uscita del sistema EV (*o1* e *o2*) alimentano le porte di ingresso (*i1* e *i2*) del sistema CP. Si assume che la tipologia dei dati delle porte *o1* e *o2* siano rispettivamente *V1* e *V2*, e il tipo delle porte di ingresso *i1* e *i2* siano rispettivamente *V3* e *V4*. Quindi, i requisiti affinché tale composizione risulti valida sono i seguenti:

$$V1 \subseteq V3 \quad (18)$$

$$V2 \subseteq V4 \quad (19)$$

Dove, nel caso specifico della SM EV&CP, gli insiemi *V1* e *V2* coincidono entrambi con l'insieme dei numeri binari \mathbb{B} . Questo assicura che ciascuna uscita prodotta dal sistema veicolo elettrico, sulle porte *o1* e *o2*, sia accettata in ingresso dal sistema punto di ricarica, sulle porte *i1* e *i2*. Sulla base di questo, è possibile affermare che ci sia un *type check*.

Nonostante dal punto di vista logico la reazione del sistema CP debba avvenire successivamente a quella di EV (poiché CP richiede le uscite di EV come ingressi), attraverso la composizione appena descritta ci si pone a un livello di astrazione superiore nel quale, a ogni reazione, è come se entrambi i sistemi reagissero istantaneamente e simultaneamente. Ciò significa che il sistema composito EV&CP reagisce come un unico blocco, facendo in modo che, a ogni reazione dello stesso, sulla base dell'ingresso *pFlow*, venga immediatamente calcolata l'uscita *enablePowerFlow*. Il tutto avviene senza dover considerare la sequenzialità interna tra EV e CP, permettendo di visualizzare l'intero processo come se le reazioni delle due macchine fossero contemporanee.

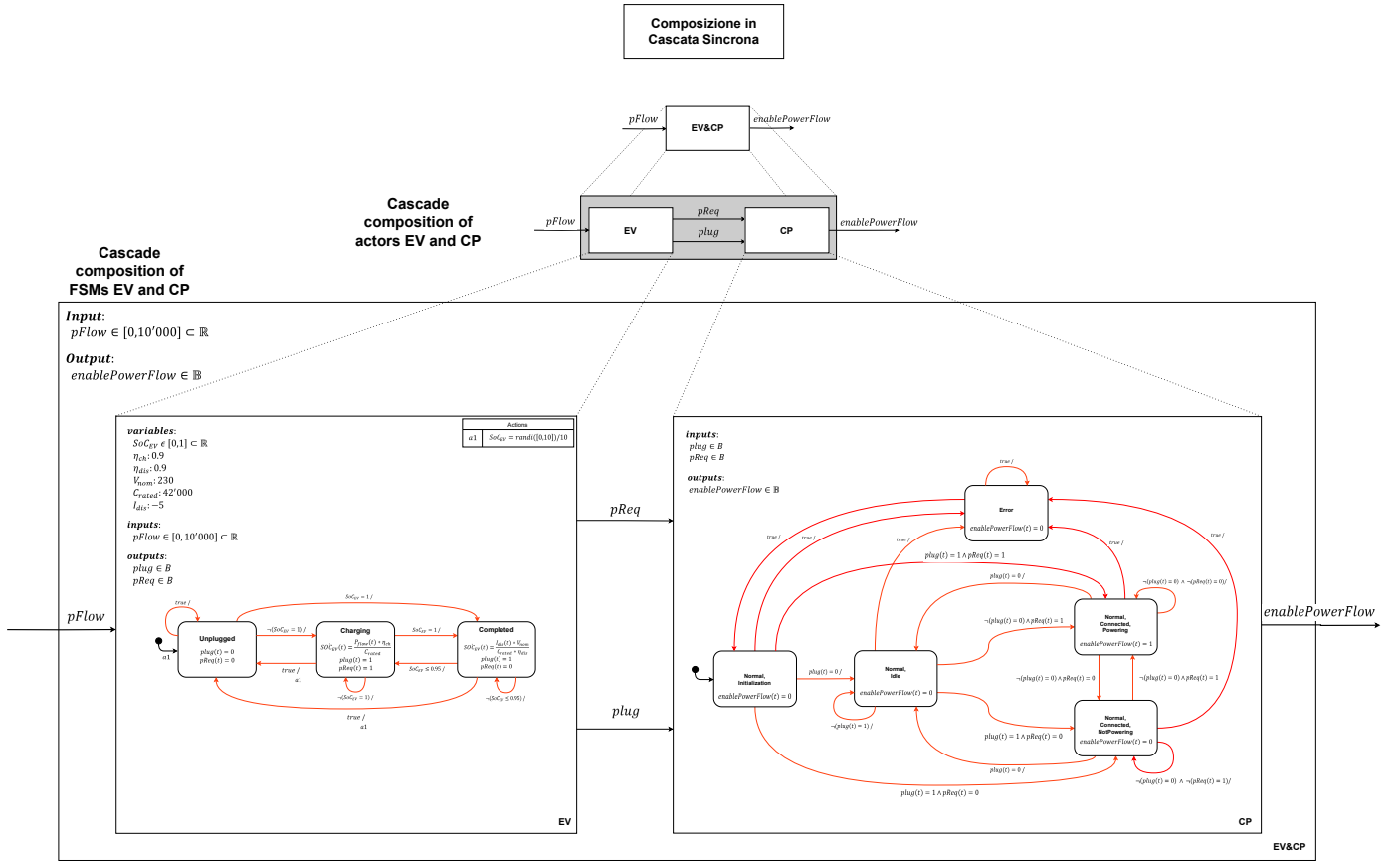
1) *Sincronismo e realtà*: Si evidenzia che, nella realtà, il sincronismo non esiste. Tuttavia, è vantaggioso utilizzare modelli sincroni. Questo perché, tali modelli, sono una buona rappresentazione del sistema nel caso in cui i ritardi, inevitabilmente presenti nella realtà, non inficino sul funzionamento del sistema. Quindi, i risultati ottenuti sono gli stessi che si osserverebbero se i sistemi fossero sincroni.

Qualora i ritardi, nella realtà, determinino comportamenti diversi da quelli sincroni, il modello di computazione sincrona non risulterebbe più adeguato. In tal caso, è necessario introdurre modelli di computazione asincroni, che tengano conto del fatto che i sistemi interagiscono effettivamente in tempi diversi.

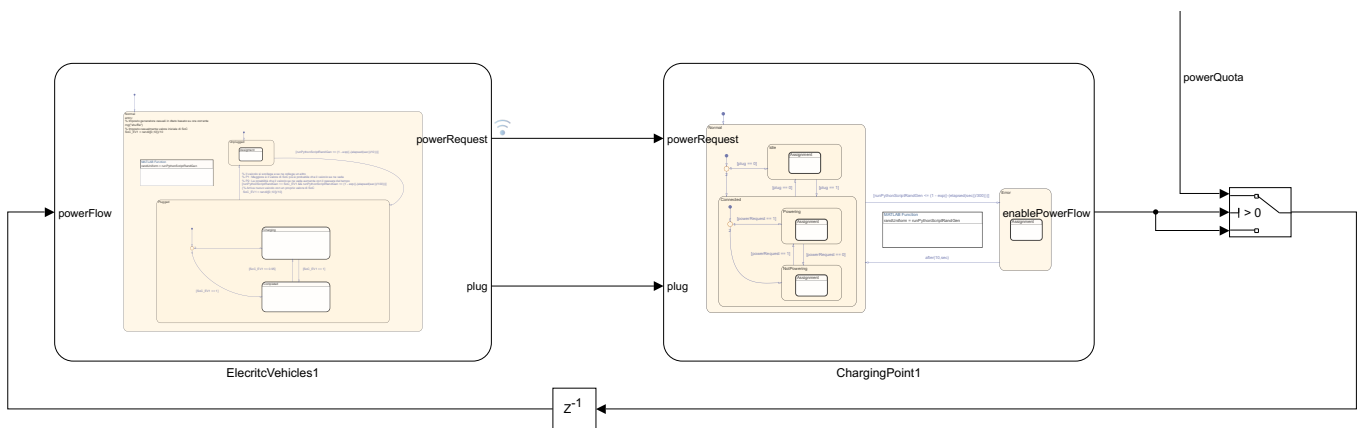
2) *Macchina equivalente della composizione in cascata sincrona*: A partire dalla composizione in cascata delle due macchine a stati, è stata ricavata la relativa macchina equivalente, rappresentata graficamente in fig. 14a e definita formalmente attraverso la quintupla riportata in fig.15.

L'insieme degli stati della macchina equivalente corrisponde al prodotto cartesiano degli insiemi degli stati delle due macchine. La macchina a stati EV, però, è caratterizzata da un numero infinito di possibili stati, di conseguenza anche l'insieme degli stati della macchina equivalente avrà cardinalità infinita. Tra questi, infatti, rientrano infiniti stati a causa degli infiniti valori che può assumere la *continuous state variable SoC*. Per questo motivo, nella definizione formale della macchina sono stati considerati solo gli stati utili all'analisi. A partire dagli stati utili considerati, è stato determinato quali fossero le transizioni percorribili, escludendo, di conseguenza, quelli irraggiungibili. E' importante notare che le transizioni sono simultanee, anche quando una causa logicamente l'altra.

3) *Implementazione in Simulink Stateflow*: In fig. 13b è riportata l'implementazione in Simulink Stateflow del modello di fig. 13a. E' importante notare che il segnale *enablePowerFlow* permette di pilotare uno *switch*, come spiegato nella sezione V-C del documento. Il segnale di uscita dello *switch*, che rappresenta il flusso di potenza erogato dal CP considerato, prima di entrare in ingresso al sistema EV, attraversa un blocco *delay*. La presenza di tale blocco è significativa poiché introduce una componente dinamica nel sistema, il che lo rende un sistema non sincrono.



(a) Modello della composizione in cascata sincrona dei sistemi veicolo elettrico e punto di ricarica.



(b) Implementazione della composizione in cascata sincrona in Simulink Stateflow.

Fig. 13: Composizione in cascata EV&CP.

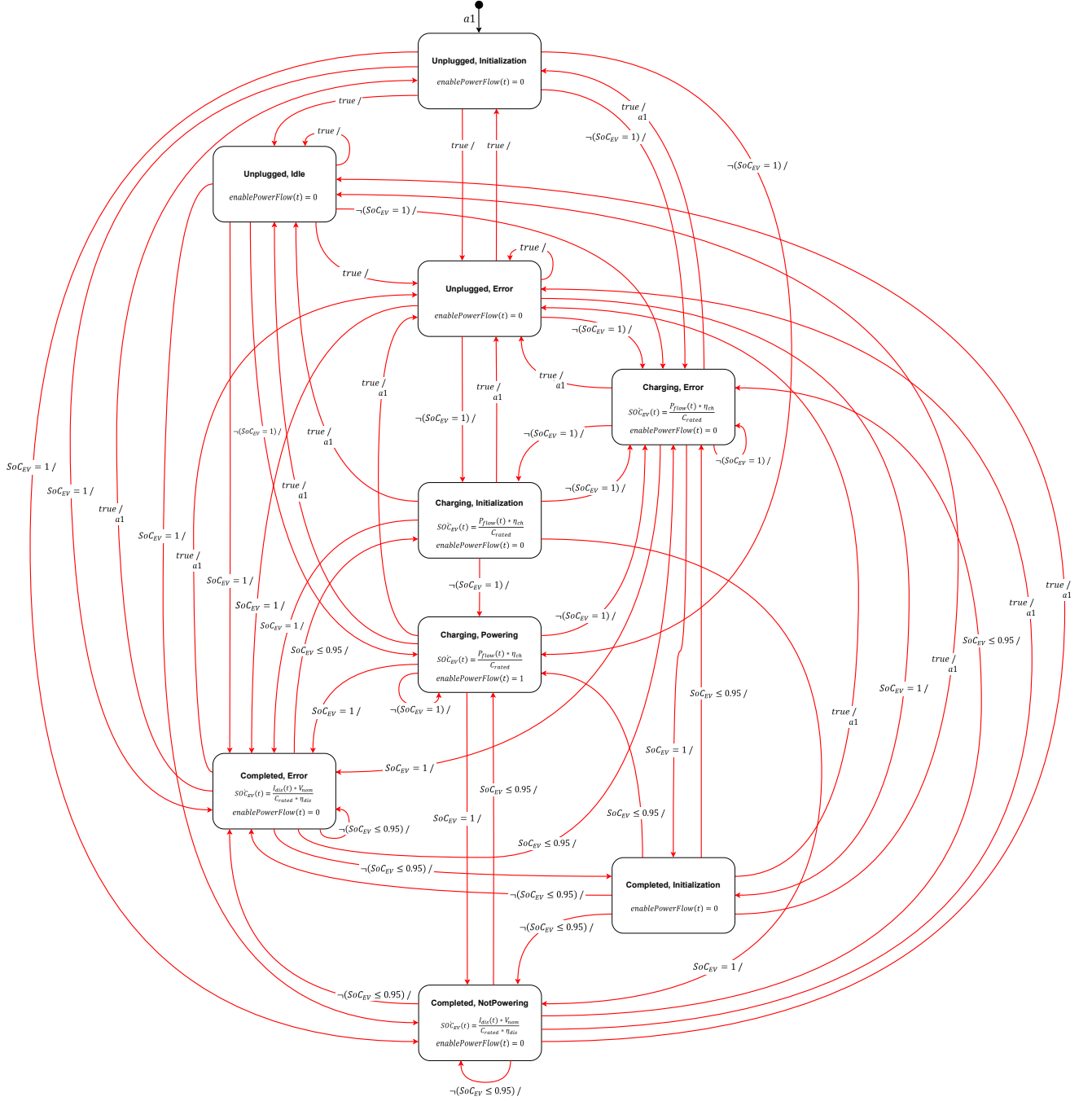


Fig. 14: Macchina a stati equivalente della composizione in cascata sincrona.

Relativamente alle caratteristiche dei segnali di ingresso e di uscita delle SM si faccia riferimento a fig. 13a.

- (a) Considerata la complessità della rappresentazione grafica, per ottimizzare la disposizione degli stati e la rappresentazione delle relative transizioni, è stato utilizzato il tool open-source Graphviz.

States = $\{(Unplugged, Initialization), (Unplugged, Idle), (Unplugged, Error), (Charging, Initialization), (Charging, Powering), (Charging, Error), (Completed, Initialization), (Completed, NotPowering), (Completed, Error)\}$

Inputs = $\{pflow \in [0, 10'000] \subset \mathbb{R}\}$

Outputs = $\{enablePowerFlow \in \mathbb{B}\}$

InitialState = $\{(Unplugged, Initialization) \in States\}$

$$\begin{aligned}
 \text{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. 15: Definizione formale della macchina equivalente relativa alla composizione in cascata sincrona, realizzata considerando solo un sottoinsieme rilevante degli stati. A tal proposito, si faccia riferimento a quanto riportato nelle sezioni IV-C.5 e IV-D.2.

E. *Definizione formale delle specifiche: utilizzo della Signal Temporal Logic*

Sulla base dei requisiti individuati nella sezione II-B, facendo ricorso ai formalismi della *Signal Temporal Logic* (STL), sono state definite le seguenti specifiche:

$$\varphi_1 = G \{ [plug(t) = 0] \implies [pReq(t) = 0] \}$$

$$\varphi_2 = G \{ [SoC(t) = 1] \implies [pReq(t) = 0] \}$$

$$\varphi_3 = G \{ [(0.95 \leq SoC(t) \leq 1) \mathbf{S}(SoC(t^*) = 1)] \implies (pReq(t) = 0) \} \mid t^* < t$$

$$\varphi_4 = G \{ [enablePowerFlow(t) = 1] \implies [plug(t) = 1] \}$$

$$\varphi_5 = G \{ [errorState(t) = 1] \implies [enablePowerFlow(t) = 0] \}$$

$$\varphi_6 = G \{ [enablePowerFlow(t) = 1] \implies [SoC(t) \neq 1] \}$$

$$\varphi_7 = G \{ 0 \leq \left[\sum_{i=1}^n pFlow_i(t) \right] \leq generatorPower(t) \}, \quad n = \text{numero di CP}$$

$$\varphi_8 = G \left\{ \left[\sum_{i=1}^{n_{idle}} pFlow_i(t) \right] = 0 \right\}, \quad n_{idle} = \text{numero di CP nello stato Idle}$$

$$\varphi_9 = \bigvee_{i=1}^n \left[pReq_i^{actual}(t) \neq pReq_i^{previous}(t) \right] \implies F_{[0,4]} \left\{ \bigwedge_{i=1}^n \left[pFlow_i(t) = \frac{generatorPower(t) \cdot poweringState_{CP_i}(t)}{n_{CP_active}} \right] \right\}$$

Poiché le specifiche riguardano i valori di segnali continui, è stato utilizzato il formalismo della STL.

Inoltre, come si evince dalla funzione di aggiornamento riportata in fig. 15, il modello della composizione (EV&CP) risulta non deterministico. Infatti, la *update function* è una funzione multivalore (*set valued*).

Ad esempio, a partire dallo stato (*Unplugged, Initialization*), al verificarsi della condizione $SoC = 1$ si ha la stessa possibilità di transitare in ognuno dei seguenti stati:

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

Per cui, la traccia di esecuzione del sistema non risulta essere lineare, ma ha una struttura ad albero (*computation tree*). Questo, costituisce un ulteriore motivo per cui, in ogni caso, non sarebbe stato possibile utilizzare la *Linear Temporal Logic* (LTL).

F. *Comunicazione o coordinazione tra le parti del sistema: utilizzo del Dataflow Model of Computation*

In questa sezione viene descritto come i punti di ricarica comunichino attraverso lo scambio di messaggi (o *token*) e coordinino le loro azioni al fine di soddisfare i requisiti di funzionamento. In particolare, nel sistema considerato, la comunicazione tra i CP si è resa necessaria per i seguenti motivi:

- Individuare un leader;
- Raggiungere il consenso.

Il flusso dati tra i CP può essere modellato mediante l'uso del *Dataflow Model of Computation* (MoC). Infatti, ogni punto di ricarica può essere visto come un attore che elabora i *token* ricevuti e reagisce sulla base della logica definita.

A tal proposito, si evidenzia che la parte del sistema dedicata al calcolo delle aliquote di potenza, che sfrutta l'algoritmo *Push-Sum*, è stata modellata come riportato

in fig. 16. Successivamente, il suddetto modello è stato implementato in Simulink, sfruttando gli strumenti messi a disposizione dalla libreria SimEvents.

1) *Libreria Simulink SimEvents*: Inizialmente, nel processo di implementazione del modello, è stata presa in considerazione la possibilità di utilizzare il *Dataflow Domain* di Simulink. Tuttavia, a causa della scarsa flessibilità dello strumento, si è optato per l'utilizzo della libreria SimEvents.

Quest'ultima, infatti, mette a disposizione un insieme di blocchi, che ben si prestano all'implementazione di processi che prevedono lo scambio di messaggi. Tra questi rientrano:

- Il blocco *Entity*, che consente di generare eventi contrassegnati da un valore numerico. Affinché un evento possa essere prodotto, tale blocco deve essere opportunamente sollecitato. Nello scenario di interesse il blocco opera la generazione di un messaggio;
- Il blocco *Entity Multicast*, che permette di notificare un gruppo di destinatari circa il verificarsi di un evento. Questo componente è responsabile dell'invio del *token*;
- Il blocco *Multicast Receive Queue*, che permette di ricevere messaggi da sorgenti multiple. Inoltre, tale blocco emula il comportamento di una coda FIFO, in cui i *token* sono accodati. In fase di implementazione a ogni CP ne è stato associato uno;
- Il blocco *Entity Terminator*, attraverso il quale si è in grado di prelevare messaggi dalla coda ed elaborarli.

2) *I punti di ricarica comunicano per raggiungere il consenso, utilizzo del dataflow sincro:* La logica di interazione tra i diversi CP, finalizzata al raggiungimento del consenso circa la percentuale di potenza da assorbire dal generatore globale, può essere compresa facendo riferimento al modello riportato in fig. 16. Lo stesso rappresenta un modello *dataflow* di tipo sincro.

Con *synchronous dataflow* (SDF) si fa riferimento a una forma vincolata di *dataflow*, in cui, ciascun attore, a ogni reazione, consuma un numero fissato di *token* su ciascuna porta di ingresso e produce un numero fissato di *token* su ciascuna porta di uscita.

Infatti, come si evince da fig. 16, ciascun attore, a ogni reazione, produce e consuma rispettivamente 2 *token*, che sono:

- Due *token* di ingresso, ossia i due valori v e w dell'attore mittente, che rappresentano rispettivamente il valore numerico e il peso associato, utilizzati dall'algoritmo *Push-Sum* per il calcolo della media;
- Due *token* di uscita, ossia i due valori v e w opportunamente elaborati dall'attore in questione.

Le equazioni di bilancio del modello, senza considerare il ramo di retroazione, che va dall'attore CP3 all'attore CP1, sono riportate di seguito:

$$2q_{CP1} = 2q_{CP2} \quad (20)$$

$$2q_{CP2} = 2q_{CP3} \quad (21)$$

Dalle sopraelencate equazioni, si ottiene il seguente sistema di equazioni lineari omogeneo:

$$\begin{cases} 2q_{CP1} - 2q_{CP2} = 0 \\ 2q_{CP2} - 2q_{CP3} = 0 \end{cases} \quad (22)$$

Quindi, è possibile ricavare la matrice M dei coefficienti, che moltiplica il vettore delle incognite $[q_{CP1}, q_{CP2}, q_{CP3}]^T$.

$$M = \begin{bmatrix} 2 & -2 & 0 \\ 0 & 2 & -2 \end{bmatrix} \quad (23)$$

Tale sistema, può essere rappresentato in forma matriciale come segue:

$$Mq = 0 \quad (24)$$

Bisogna, quindi, individuare una soluzione in " q vettore", che soddisfi il sistema.

Essendo il sistema sottodeterminato, cioè con un numero di incognite superiore al numero di equazioni, si hanno infinite soluzioni. Ciò è confermato anche dal fatto che la matrice dei coefficienti (eq. 23) non è di rango per colonne pieno. Di conseguenza, esistono infinite soluzioni oltre a quella banale. Quest'ultima, infatti, permetterebbe di risolvere il sistema ma non risulterebbe utile ai fini dell'analisi in quanto rappresenterebbe l'assenza di uno *scheduling*.

Considerato che le incognite q_{CPi} devono essere intere, tra le infinite soluzioni, quella che rende minima la dimensione dei *buffer* è la seguente:

$$q_{CP3} = q_{CP2} = q_{CP1} = 1 \quad (25)$$

Come dimostrato, quindi, le equazioni di bilancio considerate sono verificate e, di conseguenza, è possibile prevedere la presenza di *buffer* di dimensione limitata. Si dice che il modello è consistente.

Inoltre, è anche possibile affermare che esista uno *scheduling* (sequenza di reazioni delle macchine che compongono il sistema) che permetta al tutto di funzionare. A tal proposito, uno *scheduling* ammissibile risulta essere il seguente:

$$CP1 \ CP2 \ CP3 \quad (26)$$

Tale analisi, permette di capire che si tratta, senza considerare il *feedback*, di un modello consistente.

Cioè, esiste la possibilità di avere *buffer* limitati che garantiscano un'esecuzione infinita del sistema.

Il fatto di poter effettuare queste analisi rigorose è garantito dall'attributo "sincrono" di *dataflow*. Si ha la certezza, infatti, che ogni volta che reagisca un attore, il numero di *token* prodotto/consumato sia sempre lo stesso.

Tuttavia, la consistenza, in questo caso, non assicura l'esistenza di uno *scheduling* poiché c'è da considerare il ramo di retroazione.

Le equazioni di bilancio sono verificate e il modello è consistente, ma con riferimento a fig. 16, non prevedendo un'inizializzazione sul canale in retroazione si verificherebbe una situazione di stallo (*deadlock*). Per cui, lo *scheduling* non esisterebbe.

Per risolvere il *deadlock*, quindi, è sufficiente prevedere la presenza di 2 *token* di inizializzazione sul canale di retroazione $v = 0$ e $w = 0$.

3) *I punti di ricarica comunicano per eleggere un leader, utilizzo del dataflow strutturato*: Relativamente al processo di coordinazione tra i CP, con fine l'elezione di un leader, si sottolinea che si è realizzata direttamente l'implementazione della logica in Simulink. L'approccio adottato, in questo caso, è quello basato sul *dataflow* strutturato. Infatti, il comportamento di ciascun CP dipende dal tipo di messaggio ricevuto, in particolare dal suo valore. Il messaggio viene elaborato nell'*entity terminator* e a seconda del suo valore viene eseguita un'azione piuttosto che un'altra. Si tenga presente, che l'ultimo blocco citato permette di eseguire del codice come in una funzione MATLAB. Per cui, se necessario, consente di innescare l'invio di nuovi messaggi verso altri CP. Queste azioni sono organizzate in modo analogo alla programmazione strutturata, dove il comportamento del sistema è determinato da una serie di condizioni.

Di conseguenza, l'intero funzionamento del sistema si fonda su queste condizioni, che determinano in modo univoco le azioni che ciascun punto di ricarica deve compiere. Ogni CP agisce esclusivamente in risposta ai *token* ricevuti, interpretando il loro valore.

G. Algoritmo del Bullo: elezione del leader

L'algoritmo del Bullo, è una soluzione tipicamente utilizzata nei sistemi distribuiti per selezionare un processo coordinatore tra un insieme di processi. Immaginando un sistema distribuito come una rete di macchine interconnesse, l'algoritmo permette di eleggere a leader un nodo della rete, facendo in modo che tutti gli altri sappiano quando il leader è stato eletto e qual è il nodo scelto come tale. Inoltre, se il coordinatore fallisce, l'algoritmo riparte per avviare una nuova elezione.

Affinché l'algoritmo possa essere eseguito correttamente, ogni nodo della rete dev'essere contrassegnato da un *unique identifier* (UID) statico, che non può cambiare nel tempo. A ogni elezione, si è certi che il nodo attivo con UID più alto verrà eletto come leader.

La parola "bullo", presente nel nome dell'algoritmo, deriva dal fatto che quando un nodo si accorge che il leader non funziona correttamente (e.g., non riceve alcuna risposta da quest'ultimo), surclassa i nodi inferiori e "sfida" i

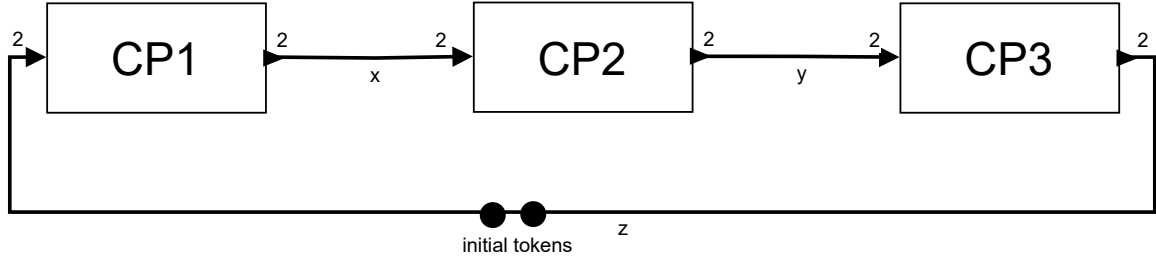


Fig. 16: Modello dataflow di tipo sincrono del processo di interazione tra i punti di ricarica, finalizzato al raggiungimento del consenso circa la percentuale di potenza destinata a ciascun CP attivo.

nodì con autorità superiore, cioè con UID più alti, tentando di prendere il controllo e inviando un messaggio a tutti i nodi con un UID maggiore del proprio.

Se un nodo con un UID più alto riceve il messaggio e può diventare leader, invia una risposta al nodo che ha tentato di prendere il controllo, segnalando che da questo momento in poi sarà lui stesso a candidarsi, inviando, a sua volta, un messaggio a tutti i nodi con un UID maggiore del proprio.

L'algoritmo prosegue in questo modo fino a quando il nodo che comunica ai propri superiori la disponibilità a diventare nuovo leader, non riceve alcun messaggio di risposta, oppure se il nodo che tenta di prendere il controllo è quello con UID più alto, cioè non c'è nessun altro nodo da contattare.

Quando un nodo viene eletto, si dichiara il nuovo leader, inviando un messaggio a tutti gli altri nodi e comunicando loro l'avvenuta elezione.

In questo lavoro, si è fatto uso di tale algoritmo per soddisfare la condizione iniziale necessaria a calcolare le aliquote di potenza medie che i punti di ricarica potessero erogare. Si aveva bisogno, infatti, che soltanto uno dei punti di ricarica memorizzasse il valore 1 e, che a tutti gli altri CP fosse associato un valore nullo.

A tal fine, in fase di simulazione, per replicare il funzionamento di tale algoritmo, sono stati utilizzati il software Simulink e la libreria SimEvents descritta precedentemente. Più in dettaglio, per ogni CP, è stata realizzata una funzione MATLAB che tenesse traccia di eventuali variazioni nel valore del segnale $pReq$, in uscita da EV. In caso, infatti, di un cambiamento nella richiesta di energia da parte di un qualsiasi veicolo connesso, attraverso questa funzione MATLAB si comunica a tutti gli altri CP della necessità di ridistribuire le risorse in modo intelligente. Gli stessi, quindi, azzerano i propri valori numerici utilizzati per il calcolo delle aliquote di potenza, con una conseguente sospensione dell'erogazione del flusso.

Ogni CP, è infatti provvisto di un modulo "dead" (dead_CP2 in fig. 17), attraverso il quale si invia un messaggio agli altri punti di ricarica, comunicando che il leader dev'essere rieletto, cioè che dev'essere riorganizzata la distribuzione dei flussi di potenza. Quando un CP riceve nella propria coda tale messaggio e lo elabora, azzerava il proprio valore, sospendendo l'erogazione di potenza.

A questo punto, ogni CP che è attivo e che ha necessità di ricaricare un veicolo, si candida a nuovo leader, comunicando tramite messaggio il proprio identificativo ai CP con UID superiore, i quali elaborano il messaggio e, se sono attivi, rispondono attraverso un opportuno modulo di risposta (response_2_to_1 in fig. 17), tentando a loro volta di assumere il controllo e sfidando i CP con UID superiore.

Il punto di ricarica che, inviato un messaggio contenente il proprio UID come candidatura, non riceve alcuna risposta entro il limite temporale di 1s, viene eletto a nuovo leader e provvede a comunicarlo agli altri punti di ricarica attraverso un opportuno modulo (elected_CP2 in fig. 17). Successivamente, imposta il proprio valore numerico utile al calcolo delle aliquote $v = 1$.

H. Algoritmo Push-Sum: consenso distribuito

L'algoritmo *Push-Sum* è un protocollo distribuito ampiamente noto in letteratura, che viene tipicamente utilizzato per calcolare funzioni di aggregati [6], tra cui la media dei valori su una rete di nodi. A differenza dell'algoritmo del bullo precedentemente descritto, in questo sistema il *Push-Sum* è caratterizzato da un processo di calcolo continuamente in esecuzione che, quindi, aggiorna costantemente la stima della media nei vari nodi della rete.

Si considera una rete costituita da N nodi disposti in una topologia ad anello, in cui ogni nodo comunica con il proprio vicino immediato in senso orario. Ogni nodo i possiede un valore numerico v_i che rappresenta una proprietà locale e che, in tal caso, indica l'aliquota di potenza erogata dal punto di ricarica CP_i . L'obiettivo dell'algoritmo è fornire a tutti i nodi una stima accurata del valore medio $\bar{v} = \frac{1}{N} \sum_{i=1}^N v_i$ attraverso un processo di distribuzione e aggiornamento continuo.

1) *Struttura dell'algoritmo*: L'algoritmo *Push-Sum* può essere suddiviso nelle seguenti fasi:

a) *Fase di inizializzazione*: Ogni nodo inizializza due variabili:

- v , che rappresenta il valore attuale del nodo;
- w , che rappresenta il peso associato a tale valore, inizialmente impostato a 1.

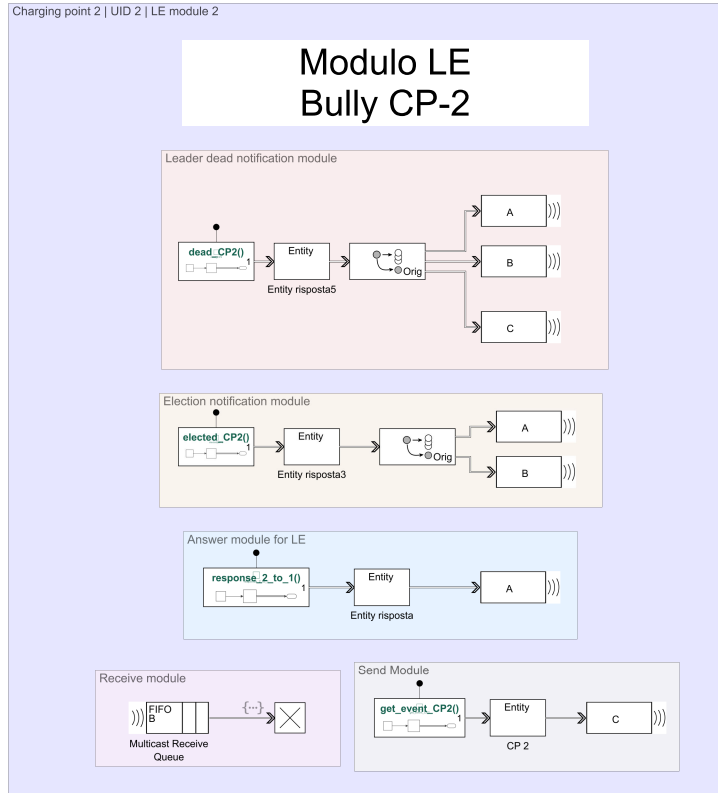


Fig. 17: Modulo dell'algoritmo del bullo per il CP2

b) Fase di esecuzione: Per i nodi attivi, cioè quelli con un veicolo collegato che richiede potenza, l'algoritmo esegue continuamente un ciclo, che viene descritto nei seguenti passaggi:

Algorithm 1 Push-Sum Averaging

- 1: **loop**
 - 2: `wait(Δ)` {Aspetta un tempo casuale}
 - 3: $p \leftarrow \text{next peer}$ {Seleziona il nodo successivo}
 - 4: `sendPush($p, (x/2, w/2)$)` {Invia metà valore e peso}

 - 5: $x \leftarrow x/2$ {Aggiorna il valore locale}
 - 6: $w \leftarrow w/2$ {Aggiorna il peso locale}
 - 7: **end loop**
-

Altrimenti, se il nodo rappresenta un CP inattivo, questi passaggi non vengono eseguiti.

c) Procedura di ricezione (onPush): Quando un nodo riceve un messaggio (m) ed è attivo, elabora il valore e il peso ricevuti ed esegue la seguente procedura:

Algorithm 2 onPush(m)

- 1: $x \leftarrow m.x + x$ {Somma il valore ricevuto a quello locale}
 - 2: $w \leftarrow m.w + w$ {Somma il peso ricevuto a quello locale}
-

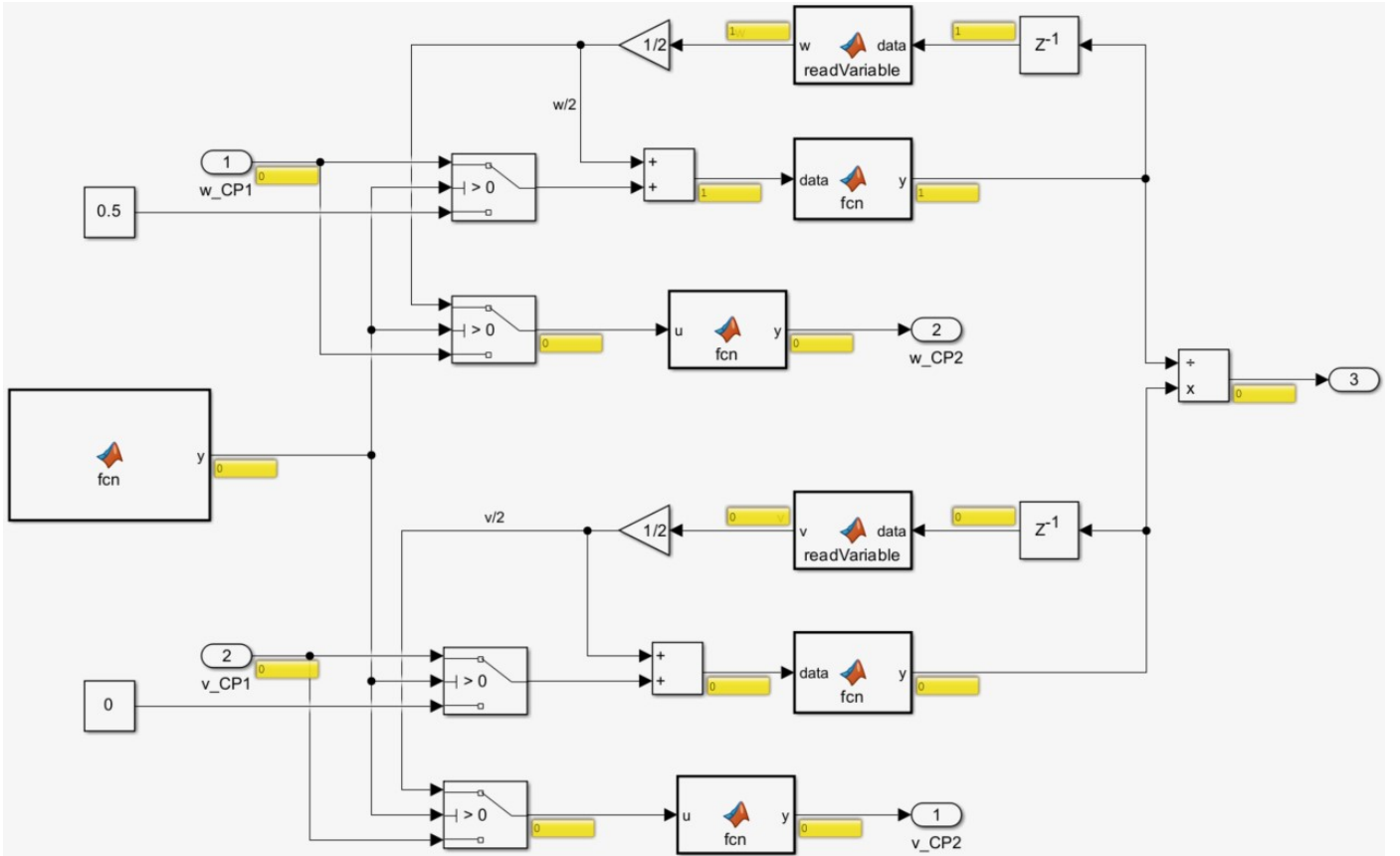
Al contrario, se il punto di ricarica che riceve il messaggio non è attivo, si limiterà a propagare il messaggio ricevuto al proprio vicino immediato in senso orario.

Quindi, il suo comportamento sarà quello di un semplice propagatore.

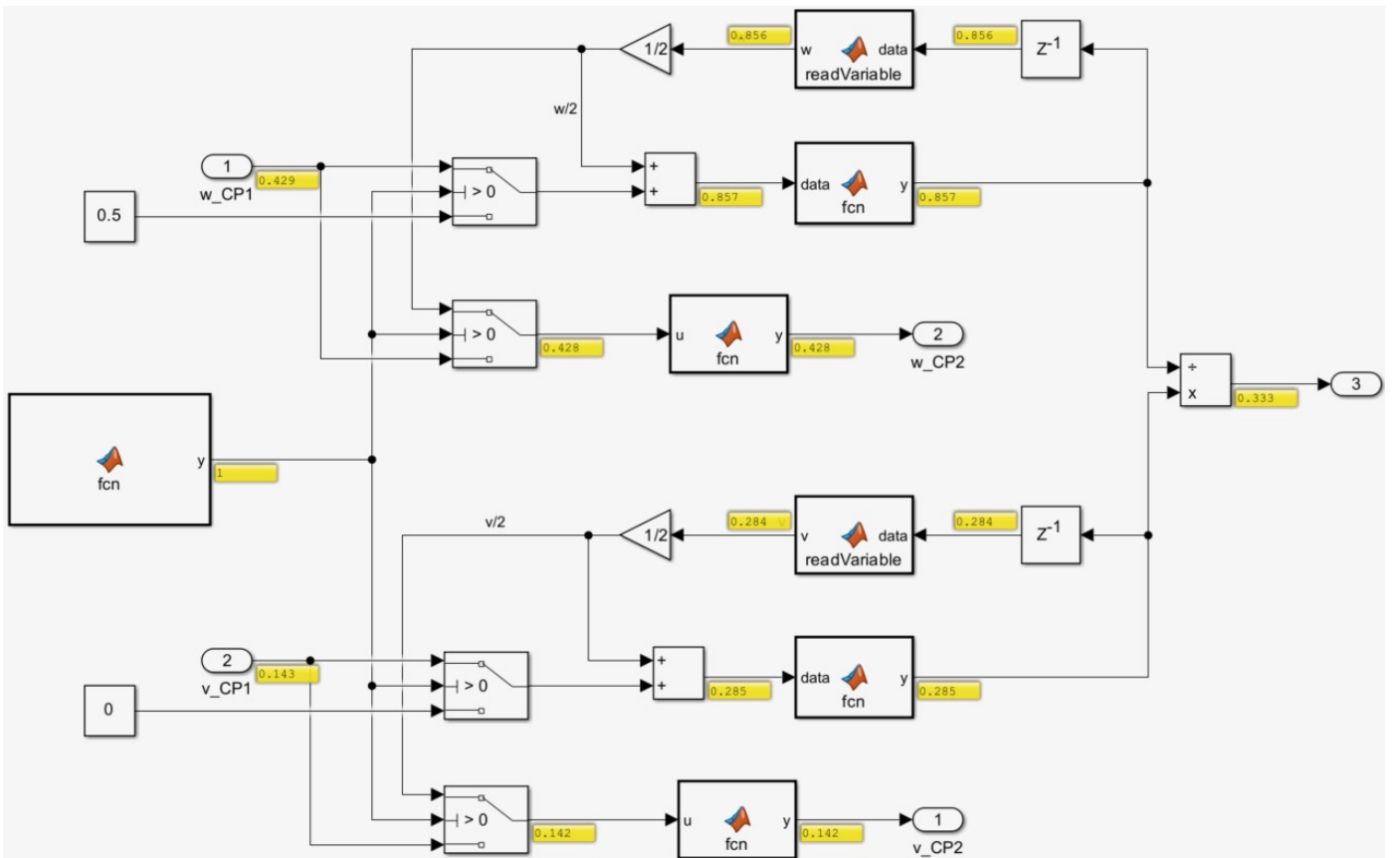
2) *Convergenza al valore medio:* Nel corso del tempo, ogni nodo diluisce il proprio valore e peso, distribuendoli tra i nodi vicini. Man mano che il processo di invio e ricezione continua, i valori e i pesi si diffondono attraverso la rete. Dopo un numero sufficiente di iterazioni, i valori e i pesi saranno distribuiti equamente tra i nodi, consentendo a ciascun nodo i di possedere una stima accurata del valore medio della rete, che viene calcolato come il rapporto tra v_i e w_i .

In altre parole, attraverso il processo di diluizione e redistribuzione continua, l'algoritmo *Push-Sum* garantisce che ogni nodo nella rete converga alla stessa stima del valore medio, a prescindere dalla topologia della rete. Infatti, il suddetto algoritmo è pensato per giungere a convergenza in sistemi distribuiti, ossia reti con una topologia che dipende dal contesto applicativo e che, quindi, non è quasi mai rappresentata da grafi completi.

In fase di simulazione, è stato previsto un meccanismo secondo il quale, nel momento in cui, tra un passo di campionamento e il successivo, il valore medio di un nodo rimane costante a meno di un fattore $p = 0.0001$, si assume che la convergenza al valore medio sia stata raggiunta. Di conseguenza, solo se tale condizione è verificata, si può utilizzare il suddetto valore per determinare la percentuale di potenza del generatore "globale" che è possibile erogare dal CP in questione.



(a) Valori generati dal modulo *Push-Sum* precedentemente all'elezione di un leader.



(b) Valori generati dal modulo *Push-Sum* successivamente all'elezione di un leader e con tre punti di ricarica attivi.

Fig. 18: Schema Simulink del modulo *Push-Sum* del CP2.
Le etichette indicano i valori prodotti e inviati sul relativo canale.

V. IMPLEMENTAZIONE DEL MODELLO IN MATLAB SIMULINK & STATEFLOW

Dopo aver analizzato e modellato i vari aspetti caratterizzanti il sistema, si è proceduto alla completa implementazione dello stesso, attraverso gli strumenti Simulink e Stateflow.

A. Differenze tra il modello e la sua implementazione

Com'è possibile osservare dalle figure relative agli schemi Stateflow delle macchine a stati e della loro composizione (fig. 4 e 10), sono presenti delle differenze tra il modello e la sua implementazione. Tali divergenze sono principalmente dovute a:

- L'utilizzo di blocchi volti a ridurre la complessità del modello, messi a disposizione dallo strumento. Un esempio è rappresentato dall'utilizzo del blocco *connective junction*;
- L'utilizzo di guardie verificabili su base probabilistica, per le transizioni che rendono le macchine non deterministiche. A tal proposito, si faccia riferimento a quanto spiegato nel capitolo IV-B.3.

B. Schema Simulink & Stateflow dell'algoritmo Push-Sum

In fig. 18 è riportato lo schema Simulink del modulo *Push-Sum* del punto di ricarica CP2. Le etichette presenti sulle connessioni tra i blocchi, mostrano il valore inviato attraverso il relativo canale di comunicazione in un preciso passo della simulazione. Si tenga presente che la rete in questione è caratterizzata da 3 nodi disposti secondo una topologia ad anello, come quella rappresentata in fig. 16, e che ognuno di questi tre nodi è caratterizzato da un modulo *Push-Sum* identico.

In fig. 18a sono mostrati i valori calcolati in un istante di tempo in cui il leader non è stato ancora eletto. Dalla funzione MATLAB a sinistra, infatti, fuoriesce un valore numerico nullo che pilota i quattro blocchi *switch*, i quali permettono di propagare valori nulli di v e w al punto di ricarica successivo CP3.

La suddetta funzione MATLAB, quindi, produce un segnale di abilitazione degli *switch*, che è pari a 1 quando è presente un leader nella rete e pari a 0 altrimenti.

Dal blocco di ritardo Z^{-1} in alto a destra, fuoriesce un valore iniziale pari a 1, il quale viene continuamente dimezzato per il funzionamento dell'algoritmo illustrato nello schema 1.

Ci sono, però, due necessità che devono essere soddisfatte:

- 1) Immediatamente dopo l'elezione di un leader, come detto in IV-H.1.a, c'è necessità che si parta da una fase iniziale in cui i nodi posseggano tutti un valore iniziale del peso $w = 1$;
- 2) La stima dei valori medi, che ogni nodo calcola come il seguente rapporto:

$$\frac{v_i}{w_i} \quad (27)$$

Deve essere nulla finché non viene eletto un leader.

A tal proposito, il valore del peso 1 in ingresso al blocco *Gain*, dopo essere stato dimezzato viene sommato al valore uscente dal blocco costante 0.5.

Grazie a questa soluzione, infatti, il valore $w = 1$

rimane continuamente in circolazione fino all'elezione di un leader. Dal blocco di ritardo in basso, invece, fuoriesce un valore iniziale pari a 0.

Questo fa sì che, finché non venga eletto un leader, la stima del valore medio calcolata attraverso il blocco divisore di ogni CP_i sia costantemente nulla:

$$\frac{v_i}{w_i} = \frac{0}{1} = 0 \quad (28)$$

In fig. 18b sono mostrati i valori calcolati in un istante di tempo in cui il leader è già stato eletto, i CP attivi sono 3 e, inoltre, si è già raggiunta la corretta stima del valore medio, pari a 0.333. Quando il leader viene eletto, infatti, la funzione MATLAB a sinistra produce un valore costante pari a 1, il quale consente ai blocchi *switch* di far passare i valori di v e w provenienti dal CP1 e utilizzare quegli stessi valori sia per aggiornare quelli propri del punto di ricarica CP2, sia per calcolare una corretta stima del valore medio.

Le ulteriori sei funzioni MATLAB, vengono utilizzate:

- Per ristabilire la condizione iniziale dell'algoritmo *Push-Sum*, in seguito alla morte del leader. Cioè azzerare tutte le variabili v_i e porre a 1 tutti i w_i ;
- Compensare il fatto che si verificano dei ritardi tra la morte del leader e la generazione di un conseguente evento, che permetta di reinizializzare le uscite prodotte dai blocchi appena descritti.

C. Schema Simulink & Stateflow del sistema

In fig. 19 è riportato lo schema Simulink e Stateflow dell'intero sistema considerato. E' possibile notare come, per la simulazione, si sia optato per una rete di 3 punti di ricarica organizzati secondo una topologia ad anello.

La logica di funzionamento dello schema prevede, essenzialmente, il susseguirsi delle seguenti fasi:

- All'avvio della simulazione viene simulato il comportamento di un gruppo di veicoli che si connettono alla rete di CP considerati. In particolare, per ogni CP, su base probabilistica, viene stabilito se allo stesso è connesso o meno un EV. Inoltre, viene determinato casualmente il valore iniziale del *SoC* della sua batteria. Tali valori sono visualizzabili graficamente, nell'ambiente di simulazione, attraverso un blocco display posto a sinistra di ogni EV, e da un indicatore (blocco *horizontal gauge*) posto al di sotto di ogni veicolo;
- Quando un veicolo si collega a un CP e, quindi, il relativo segnale *pReq* assume un valore alto, la lampada alla destra del veicolo si colora di verde, in caso contrario si illumina di colore rosso;
- Quando c'è una variazione del valore del segnale *pReq*, la differenza tra il valore precedente e quello attuale del suddetto segnale viene catturata da un'apposita funzione MATLAB, che avvia l'algoritmo del Bullo per l'elezione del leader, come spiegato nel capitolo IV-G. Quindi, è importante notare come ogni variazione dei segnali *pReq*, in ingresso a ciascun CP, determini l'avvio del processo di elezione, necessario per il raggiungimento del consenso sulle aliquote di potenza erogabili da ciascun punto di ricarica;

- Ogni CP misura il tempo trascorso dall'invio del messaggio di candidatura, attraverso il proprio orologio. Si evidenzia, che si è assunto di operare in un sistema distribuito sincrono, in cui il termine "sincrono" viene, in questo caso, usato con un'accezione diversa rispetto a quella attribuitagli nel contesto dei controllori. Infatti, si vuole considerare la possibilità di individuare un *upper bound* sul ritardo di trasmissione. Per cui, si assume che il ritardo massimo osservabile, entro il quale un CP deve ricevere risposta dopo l'invio del messaggio di candidatura, sia pari a 1s. Quindi, se il CP non riceve risposte al messaggio di candidatura, facendo riferimento alla logica dell'algoritmo del Bullo, può proclamarsi leader. Questo permette di aggiornare le variabili sfruttate per raggiungere il consenso;
- Infine, l'algoritmo *Push-Sum*, che è continuamente in esecuzione, determina il raggiungimento del consenso. In particolare, il "Modulo algoritmo *Push-Sum CP_i*" di fig. 19 produce in uscita un valore compreso tra 0 e 1, che rappresenta la percentuale di potenza del generatore, erogabile dal singolo *CP_i*. Tale valore viene moltiplicato per il valore di potenza fornito dal generatore, che si è assunto costante nel tempo. Infine, il risultato della moltiplicazione viene fornito in ingresso a uno *switch*, pilotato dal segnale *enablePowerFlow*.

D. Limiti del modello considerato

Il modello sviluppato si basa su alcune assunzioni e semplificazioni, le quali sono discusse nei capitoli precedenti e qui riassunte per chiarezza:

- La rete da modellare sia rappresentata da un grafo orientato di ordine tre con una topologia ad anello. A tal proposito, si sottolinea che l'implementazione del modello proposta non esprime flessibilità circa l'aggiunta di nodi alla rete o circa la modifica della topologia stessa;
- La rete di comunicazione tra i CP non presenti *failure* e sia affidabile. Quindi, che la comunicazione tra i CP non venga mai interrotta o alterata;
- La rete di punti di ricarica venga considerata come un sistema distribuito sincrono, come descritto nella sezione V-C del documento. Dove, con sistema distribuito si intende una rete di nodi, che cooperano e comunicano, solo ed esclusivamente tramite lo scambio di messaggi;
- La composizione tra le macchine a stati EV e CP sia sincrona. A tal proposito, si ribadisce che il sincronismo nella realtà non esiste, come descritto nella sezione IV-D.1 di questo documento;
- Siano sempre verificati i presupposti necessari affinché l'algoritmo del Bullo funzioni. Ad esempio, ciascun nodo della rete deve essere etichettato con un identificativo univoco.

Quelle elencate sono le principali assunzioni che permettono al modello di funzionare. Tuttavia, ne sono presenti anche altre, meno significative ed evidenti. Con questo, si vuole porre enfasi sul fatto che si ha consapevolezza dei limiti del modello. Ciò nonostante, la cosa si sposa bene con la principale esigenza di contesto, ossia utilizzare con consapevolezza gli strumenti e i concetti appresi durante il corso di "Analisi e Controllo di Sistemi Cyberfisici".

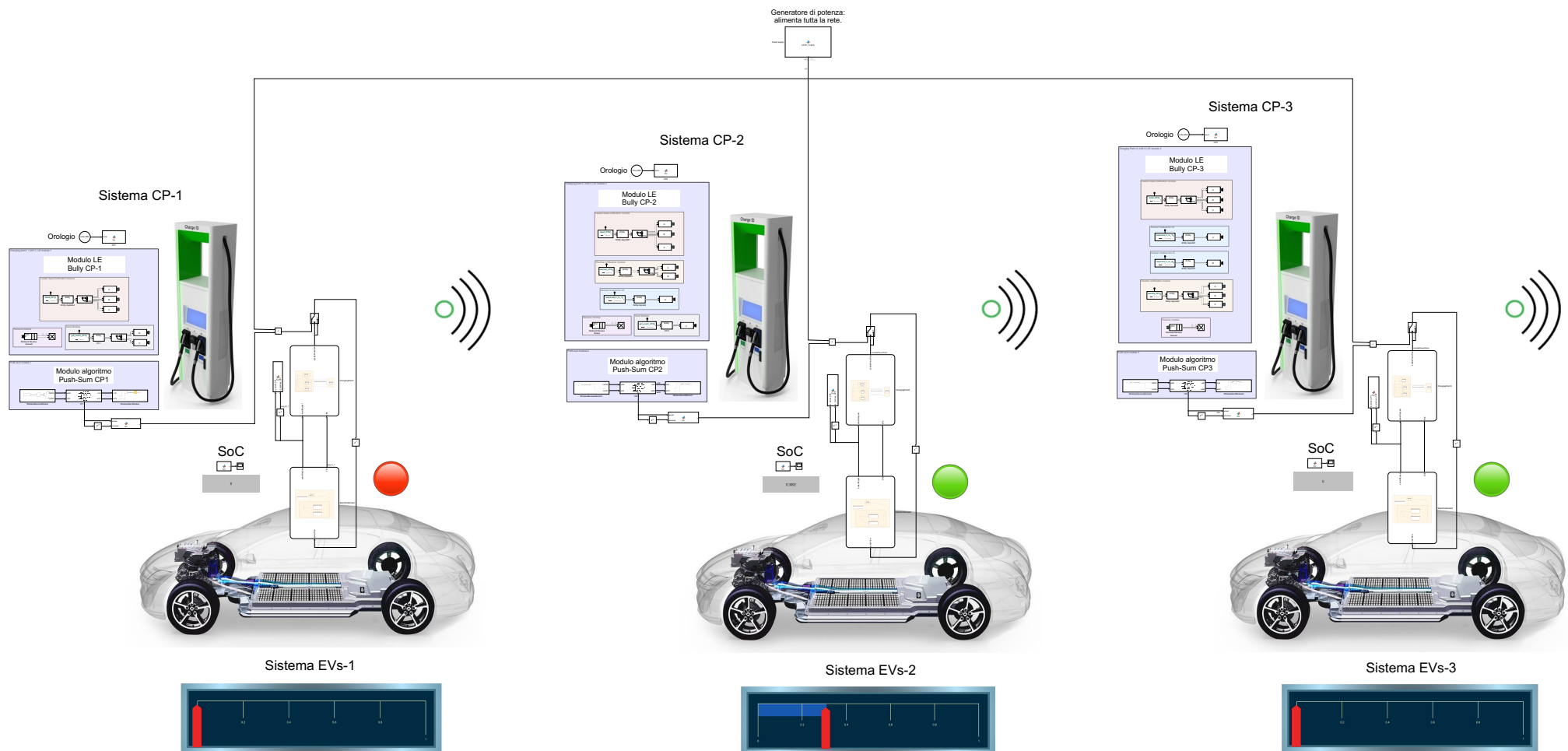


Fig. 19: Schema Simulink e Stateflow del sistema completo.

VI. VERIFICA FORMALE DELLE SPECIFICHE - STL

Per la verifica delle specifiche, definite in maniera formale tramite la STL, è stato utilizzato lo strumento Simulink Test. Con Simulink Test è possibile creare test su sistemi o su componenti isolate. È possibile definire valutazioni basate sui requisiti, a partire da una logica temporale come l'STL e specificare input, output attesi e tolleranze del test. Si tenga presente che Simulink Test non supporta l'utilizzo dell'operatore di implicazione e degli operatori temporali propri della STL. Quindi, al fine di verificare le specifiche descritte, queste ultime sono state adattate, facendo riferimento agli operatori supportati e messi a disposizione dallo strumento stesso. Ad esempio, per l'operazione di implicazione si è fatto riferimento alla seguente operazione equivalente:

$$A \implies B = \neg(A) \vee B \quad (29)$$

Come dimostrato dalla seguente tabella di verità:

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

Gli altri strumenti utilizzati, messi a disposizione da Simulink Test, sono:

- *Bounds check*, che si riferisce a una tipologia di test che verifica se i segnali o le variabili in un modello Simulink rispettano determinati limiti (*bounds*) durante l'esecuzione della simulazione. Questo tipo di controllo è fondamentale per identificare eventuali valori che oltrepassano i *range* attesi, segnalando potenziali errori o comportamenti anomali nel sistema. Ad esempio, tale test è stato usato per descrivere la specifica 7, riportata in tab. VIIa;
- *Trigger Response*, che si riferisce a un meccanismo per attivare (*trigger*) una risposta o un comportamento specifico in un sistema, in base a determinate condizioni che si verificano durante la simulazione. Tale test è stato usato per descrivere la specifica 9 (tab. IXa).

Si sottolinea, che in fase di definizione e verifica delle specifiche, sono stati concepiti segnali non previsti durante la fase di realizzazione del modello. Tali segnali sono:

- $SoC \in \mathbb{A}^{\mathbb{R}^+}$ con $\mathbb{A} = \{x \in \mathbb{R} \mid 0 \leq x \leq 1\}$ rappresenta il valore del livello di carica della batteria di un generico EV. Tale segnale è stato utilizzato nelle specifiche 2 (tab. IIa) e 6 (tab. VI);
- $completedState \in \mathbb{B}^{\mathbb{R}^+}$ assume valore 1 quando la macchina a stati del generico CP si trova nello stato *Completed*. Assume valore nullo in caso contrario. Tale segnale è stato utilizzato nella specifica 3 (tab. IIIb);
- $errorState \in \mathbb{B}^{\mathbb{R}^+}$ assume valore 1 quando la macchina a stati del generico CP si trova nello stato *Error*. Assume valore nullo in caso contrario. Tale segnale è stato utilizzato nella specifica 5 (tab. Va);
- $generatorPower = 10000$ rappresenta la potenza messa a disposizione dal generatore della rete, che

alimenta tutti i CP. Si è assunto che tale valore sia costante nel tempo. Tale segnale è stato utilizzato nelle specifiche 7 (tab. VIIa) e (tab. IXa) 9;

- $pS_i_idle \in \mathbb{R}^{\mathbb{R}^+}$ è il risultato della moltiplicazione tra il segnale $idleState \in \mathbb{B}^{\mathbb{R}^+}$, che indica se il CP_i è nello stato *Idle* o meno, e il segnale $pFlow \in \mathbb{R}^{\mathbb{R}^+}$. Quando il punto di ricarica si trova nello stato *Idle*, il relativo segnale $idleState$ è pari a 1, quindi l'uscita del moltiplicatore sarà nulla solo se sarà nulla la potenza erogata dal CP ($pFlow$). Nel caso in cui, invece, il punto di ricarica i abbia un veicolo connesso, allora il relativo segnale pS_i_idle (uscita del moltiplicatore) sarà comunque nullo, in quanto il relativo segnale $idleState$ sarà pari a 0. Tale segnale è stato utilizzato nella specifica 8 (tab. VIIIa);
- $pReq_i^{actual}$ rappresenta il valore del segnale $pReq$ relativo al CP_i , nell'istante di tempo corrente. Tale segnale è stato utilizzato nella specifica 9 (tab. IXa);
- $pReq_i^{previous}$ rappresenta il valore del segnale $pReq$ relativo al CP_i ritardato di un passo. Tale segnale è stato utilizzato nella specifica 9 (tab. IXa).

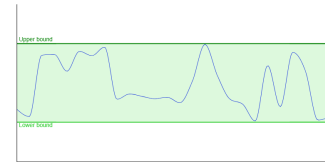
A causa dei limiti dello strumento Simulink Test, non è stato possibile verificare l'*assessment* relativo alla specifica $S_EV_#3$.

Per comodità, al fine di favorire la comprensione dei formalismi utilizzati nella definizione delle specifiche, i dettagli riportati nell'elenco soprastante sono stati inseriti anche nelle didascalie delle tabelle presenti nelle pagine successive.

1) Limiti del processo di verifica delle specifiche:

Simulink Test verifica le specifiche solo durante il tempo di simulazione, il che significa che se una determinata condizione o specifica non si verifica durante il tempo simulato, il test non riuscirà a rilevarlo. Questo può essere dovuto a vari motivi, tra cui:

- Durata della simulazione insufficiente: se la simulazione non dura abbastanza a lungo da far emergere la situazione in cui la specifica deve essere verificata, non verrà testata adeguatamente;
- Copertura incompleta del modello: se gli eventi o le condizioni che portano al fallimento della specifica non si verificano nel corso della simulazione, non sarà possibile verificare che il sistema reagisca correttamente a tali eventi.



(a) Rappresentazione visuale della specifica S.CPS.#1.



(b) Rappresentazione visuale della specifica S.CPS.#3.

Fig. 20: Rappresentazioni visuali di alcune specifiche definite tramite lo strumento Simulink Test.

Specifica 1	
Linguaggio Naturale	Un veicolo elettrico non può richiedere potenza se non è connesso a un punto di ricarica
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: Specifica 1 espressa in diversi formati

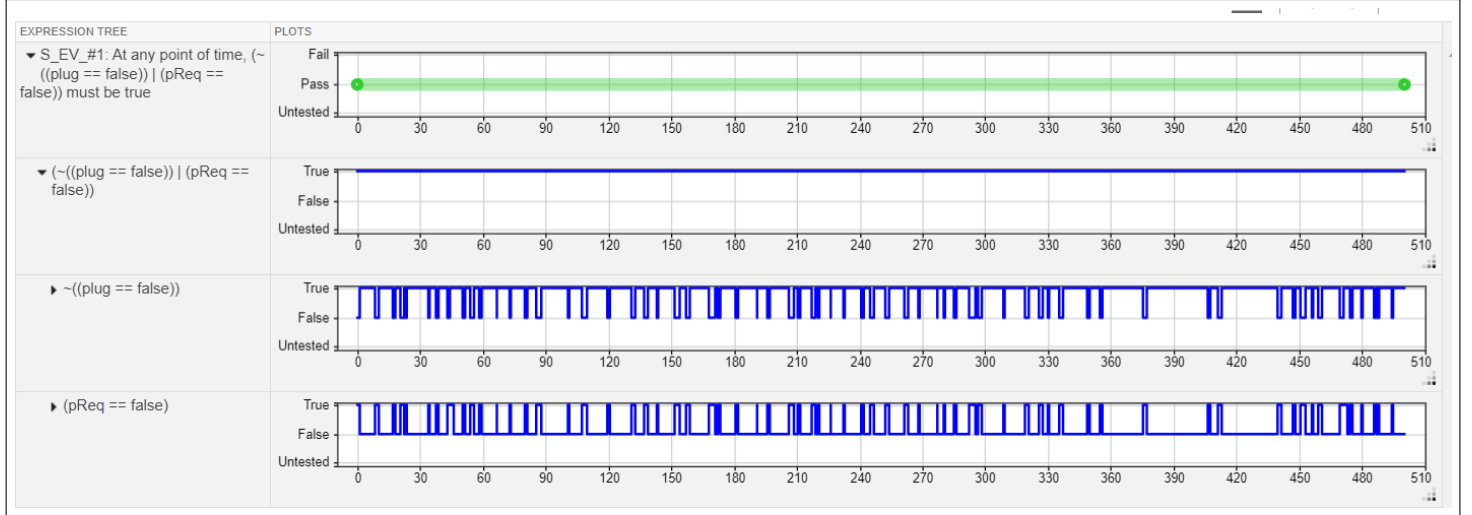


Fig. 21: Verifica della specifica S_EV_#1 in Simulink Test.

Specifica 2	
Linguaggio Naturale	Se la batteria è carica il veicolo non può richiedere potenza
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: Specifica 2 espressa in diversi formati.

(a) Il segnale $SoC \in \mathbb{A}_+^{\mathbb{R}}$ dove $\mathbb{A} = \{x \in \mathbb{R} \mid 0 \leq x \leq 1\}$ e rappresenta il valore del livello di carica della batteria del veicolo considerato.

Specifica 3	
Linguaggio Naturale	Se la batteria ha un livello di carica compreso tra il 95% e il 100% e precedentemente questo era uguale al 100%, allora non potrà richiedere potenza finché il SoC non scende al di sotto del 95%
STL - φ_3	$G \{ [(0.95 \leq SoC(t) \leq 1) \mathbf{S}(SoC(t^*) = 1)] \implies [(pReq(t) = 0) \mathbf{U}^w (SoC(t) \geq 0.95)] \mid t^* < t \}$
Simulink Test - S_EV_#3 (assessment non verificato)	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: Specifica 3 espressa in diversi formati.

- (a) La formula STL è leggibile nel seguente modo: "Deve essere sempre verificato che il fatto che $0.95 \leq SoC(t) \leq 1$ dal momento in cui è stato uguale a 1, in un tempo passato, implichi che non ci sia richiesta di potenza finché il valore di *SoC* non scende al di sotto di 0.95.
- (b) Nella scrittura dell'*assessment* in Simulink Test, è stato considerato il fatto che, al raggiungimento di un valore di $SoC(t) == 1$, se il veicolo è collegato ($plug == true$), si entrerà nello stato *Completed* e si rimarrà in tale stato fin quando il valore di *SoC* dello stesso veicolo rimarrà superiore al 0.95. Infatti, il segnale $completedState(t) \in \mathbb{B}^{\mathbb{R}^+}$ assume valore 1 se ci si trova nello stato *Completed*, 0 altrimenti.

Specifica 4	
Linguaggio Naturale	Il punto di ricarica può erogare potenza solo se ha un veicolo collegato
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: Specifica 4 espressa in diversi formati

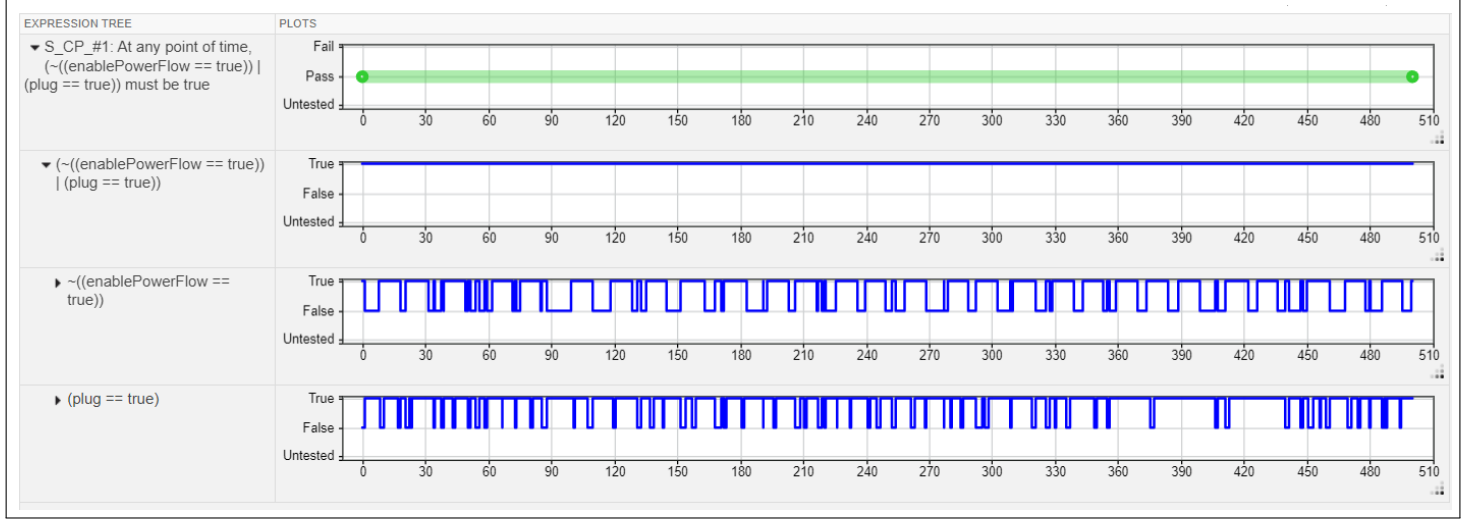


Fig. 22: Verifica della specifica S_CP_#1 in Simulink Test.

Specifica 5	
Linguaggio Naturale	Il punto di ricarica non può erogare potenza se è guasto
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: Specifica 5 espressa in diversi formati

(a) Il segnale $errorState(t) \in \mathbb{B}^{\mathbb{R}^+}$ è stato utilizzato esclusivamente in fase di validazione delle specifiche. Lo stesso assume valore 1 quando il punto di ricarica è in stato di errore, altrimenti è pari a 0.

Specifica 6	
Linguaggio Naturale	Il punto di ricarica eroga potenza al veicolo, solo se quest'ultimo non ha la batteria carica
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: Specifica 6 espressa in diversi formati.

Specifica 7	
Linguaggio Naturale	In qualsiasi istante di tempo, la somma delle potenze erogate dai punti di ricarica deve essere maggiore o uguale a zero e non deve mai superare il valore di potenza fornito dal generatore
STL - φ_7	$G\{0 \leq [\sum_{i=1}^n pFlow_i(t)] \leq generatorPower(t)\}, \quad n = \text{numero di CP}$
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: Specifica 7 espressa in diversi formati

(a) Il segnale $pFlow_i(t)$ rappresenta l'aliquota di potenza erogata da CP_i . Il segnale $generatorPower$ è stato assunto costante e pari a 10'000.

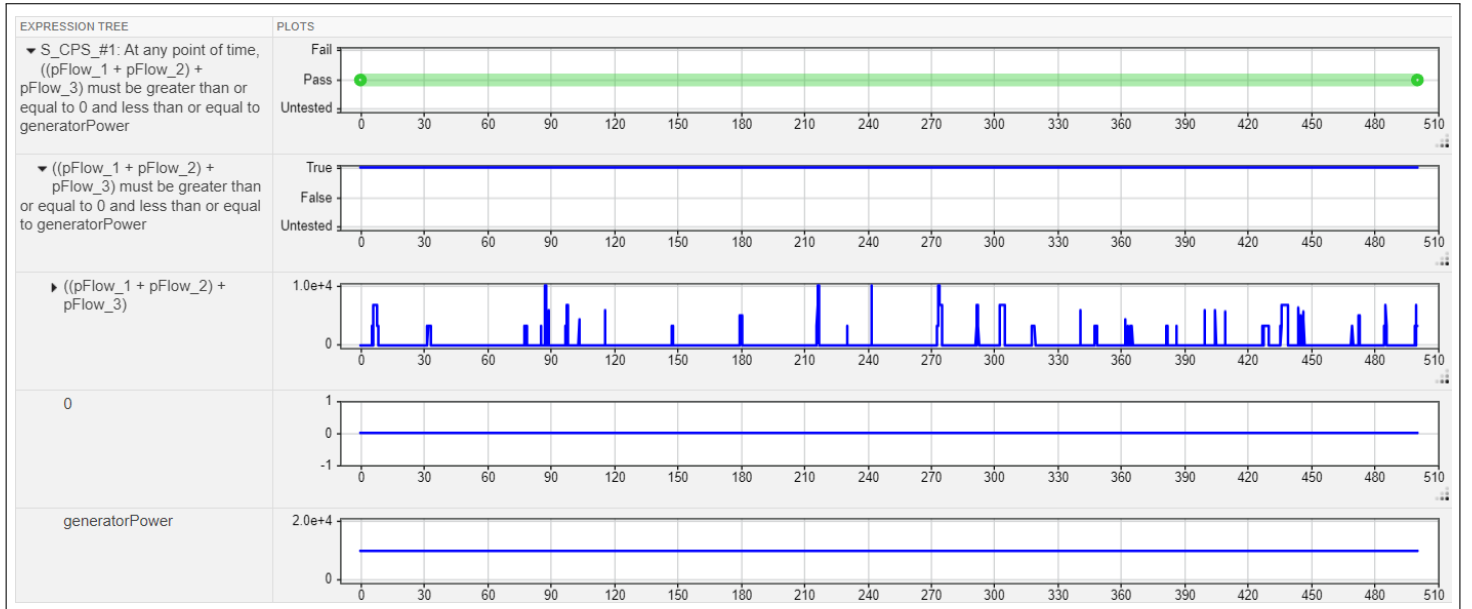


Fig. 23: Verifica della specifica S_CPS_#1 in Simulink Test.

Specifica 8	
Linguaggio Naturale	In qualsiasi istante di tempo, la somma delle potenze erogate dai punti di ricarica ai quali non sono connessi veicoli deve essere nulla
STL - φ_8	$G\{[\sum_{i=1}^{n_{idle}} pFlow_i(t)] = 0\}, \quad n_{idle} = \text{numero di CP nello stato Idle}$
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: Specifica 8 espressa in diversi formati

(a) Il segnale $pS_i_idle(t) \in \mathbb{R}^+$ è il risultato della moltiplicazione tra il segnale $idleState(t) \in \mathbb{B}^+$, che indica se il CP_i è nello stato *Idle* o meno, e il segnale $pFlow(t) \in \mathbb{R}^+$. Quando il punto di ricarica si trova nello stato *Idle*, il relativo segnale $idleState(t)$ sarà pari a 1, quindi l'uscita del moltiplicatore sarà nulla solo se sarà nulla la potenza erogata dal CP ($pFlow(t)$). Nel caso in cui, invece, il punto di ricarica i abbia un veicolo connesso, allora il relativo segnale $pS_i_idle(t)$ (uscita del moltiplicatore) sarà comunque nullo, in quanto il relativo segnale $idleState(t)$ sarà pari a 0. Si noti che il segnale $pS_i_idle(t)$ è stato modellato in questo modo al fine di individuare una strategia per individuare i CP nello stato *Idle*.

Specifica 9	
Linguaggio Naturale	Entro 4s dall'ultima variazione della richiesta di potenza di un veicolo, ogni CP attivo e non guasto deve erogare una potenza uguale a quella del generatore diviso il numero di punti di ricarica attivi
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: Specifica 9 espressa in diversi formati

(a) Il termine $\alpha(t) = \frac{generatorPower(t) \cdot poweringState_{CP1}(t)}{\text{numero di CP attivi all'istante } t}$ e rappresenta l'aliquota di potenza erogata dal punto di ricarica CP1 nell'istante di tempo t . Si evidenzia, che quando tale CP non ha un veicolo connesso richiedente potenza, il numeratore si annulla in quanto $poweringState_{CP1}(t) = 0$. Infatti, la macchina a stati del CP si trova nello stato *Powering* se e solo se risulta $plug(t) = 1$ e $pReq(t) = 1$. Inoltre, nel caso in cui il numeratore non sia nullo, lo stesso viene diviso per il numero di CP attivi, che sono quelli che partecipano al processo di consenso. Lo stesso ragionamento si applica per i segnali $\beta(t)$ e $\gamma(t)$.

VII. RISULTATI SPERIMENTALI

Quando si concepisce un modello di un sistema, come nel caso di un *cyber-physical system*, l'obiettivo principale è permettere analisi approfondite. Il modello non è creato solo per rappresentare il sistema, ma per fungere da strumento su cui eseguire valutazioni che sarebbero impraticabili o rischiose da condurre direttamente sul sistema reale.

A tal proposito, sono state eseguite varie simulazioni per valutare alcuni aspetti specifici del sistema. Queste simulazioni consentono di analizzare il comportamento del modello sotto diversi scenari e condizioni, verificando come il CPS reagisca in situazioni critiche o particolarmente complesse. Attraverso tali simulazioni, è possibile ottenere una comprensione più approfondita delle prestazioni, della sicurezza e dell'affidabilità del sistema, confermando la validità del modello prima della sua applicazione nel mondo reale.

In particolare, nel caso considerato sono state prodotte diverse viste grafiche del comportamento del sistema, che sono riportate nelle figure 24, 25, 26 e 27.

Nel grafico di fig. 24 è rappresentato come i punti di ricarica giungano a consenso in relazione alle richieste di potenza dei veicoli. Graficamente, il processo di consenso è rappresentato dall'effetto imbuto, che le funzioni producono progressivamente, nell'avvicinarsi al medesimo valore finale. Si sottolinea, che nel contesto considerato, il consenso è rappresentato dalla condizione per cui tutti i punti di ricarica attivi posseggano lo stesso valore, calcolato attraverso l'algoritmo *Push-Sum*. Tale valore rappresenta l'aliquota di potenza del generatore globale a loro destinata. Analizzando il grafico risulta che:

- L'effetto imbuto sia tanto più prolungato quanto più rimane invariato lo stato delle richieste di potenza da parte dei veicoli, nei confronti dei rispettivi CP. Infatti, l'imbuto si interrompe quando c'è un'alterazione delle richieste di potenza;
- Dal punto di vista prestazionale, sarebbe auspicabile accorciare il più possibile l'arco temporale necessario per giungere a consenso. E' evidente, che su questo aspetto intervengano le latenze di comunicazione, la topologia e il numero di nodi della rete;
- Il sistema presenta una vulnerabilità. Infatti, qualora sia presente una continua modifica dello stato di una richiesta di potenza, nei confronti di un determinato CP, il consenso non verrebbe mai raggiunto. In termini tecnici, ciò si traduce in un continuo avvio dell'algoritmo del Bullo, che determina un continuo riavvio del processo di elezione del leader e, quindi, l'annullamento delle stime calcolate attraverso l'algoritmo *Push-Sum*.

Di seguito, è fornita un'interpretazione del comportamento del sistema nell'arco temporale della simulazione che va da 0s a 5s (fig. 24). Da tale finestra temporale si evince che:

- Il punto di ricarica 1 riceve una richiesta di potenza. Infatti, il segnale *pReq_CP1* "si alza";
- Il punto di ricarica 2 riceve una richiesta di potenza. Infatti, il segnale *pReq_CP2* "si alza";
- Il punto di ricarica 3 riceve una richiesta di potenza. Infatti, il segnale *pReq_CP3* "si alza";
- Avvenuta l'elezione del leader, le stime dei moduli *Push-Sum* iniziano a oscillare fino a quando non viene raggiunto il consenso (4.5s);
- Infine, il comportamento del sistema viene alterato dall'interruzione di richiesta di potenza da parte del veicolo nei confronti del punto di ricarica 1.

Nei grafici di fig. 25, 26 e 27, è rappresentato come il *SoC* del veicolo connesso a un determinato CP cresca in corrispondenza degli intervalli temporali in cui il consenso è stato raggiunto, cioè nei quali si verifica la convergenza delle stime calcolate attraverso il modulo *Push-Sum*. Analizzando fig. 27 risulta che:

- Avviata la simulazione il livello di carica della batteria è stazionario e pari al 40%;
- All'istante di tempo 4.5 si verifica il raggiungimento del consenso. Per cui, il *SoC* inizia a crescere (la batteria carica);
- Nei due successivi tentativi di convergenza non si riesce a raggiungere il consenso, motivo per il quale il *SoC* resta stazionario al 52%;
- Poco dopo l'istante di tempo 10 si verifica un cambio repentino del *SoC*, tale variazione corrisponde alla connessione di un nuovo veicolo, con un nuovo livello di carica;
- Raggiunto il consenso, all'istante di tempo 15.4, la batteria inizia a caricare e ciò si evince dall'evoluzione del *SoC*;
- Nell'istante di tempo 22.8 viene considerato un nuovo veicolo, non ancora connesso al CP. Il livello di carica della sua batteria è pari al 100%. Tale veicolo si connette al CP all'istante di tempo 25 ma, siccome il suo *SoC* è massimo, non gli viene erogata potenza e, di conseguenza, inizia a scaricarsi.

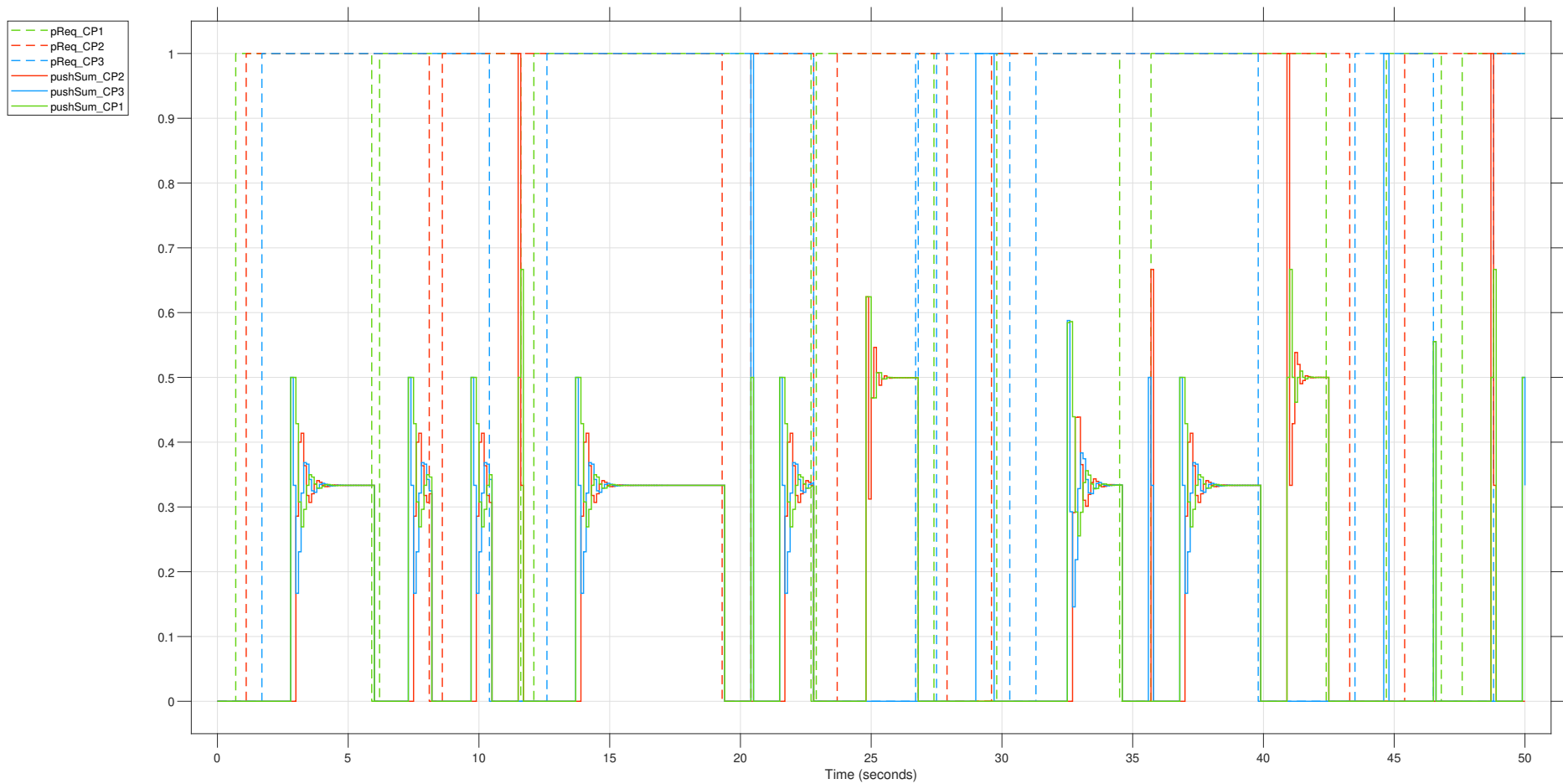


Fig. 24: Grafico che mostra come i punti di ricarica giungano a consenso in relazione alle richieste di potenza dei veicoli. Le funzioni rappresentate con tratto continuo rappresentano le stime del valore medio calcolate attraverso i moduli *Push-Sum*. Le linee tratteggiate rappresentano le richieste di potenza. I colori servono a distinguere i diversi nodi (punti di ricarica) della rete.

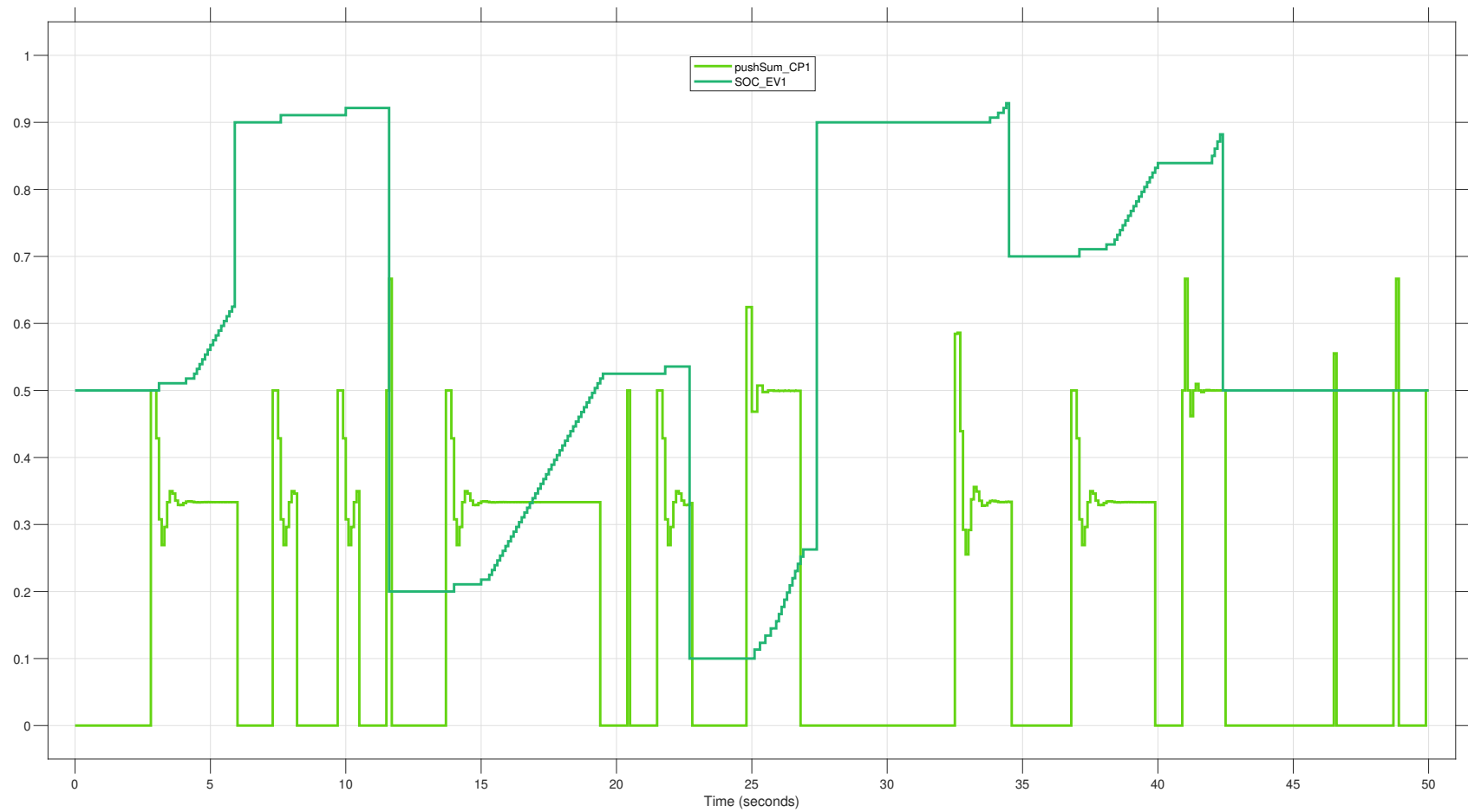


Fig. 25: Grafico che mostra il valore del *SoC* del veicolo connesso al CP1 (funzione rappresentata dal colore più scuro) in relazione alla stima del valore medio calcolata attraverso il modulo *Push-Sum* del punto di ricarica 1 (funzione dal colore più chiaro).

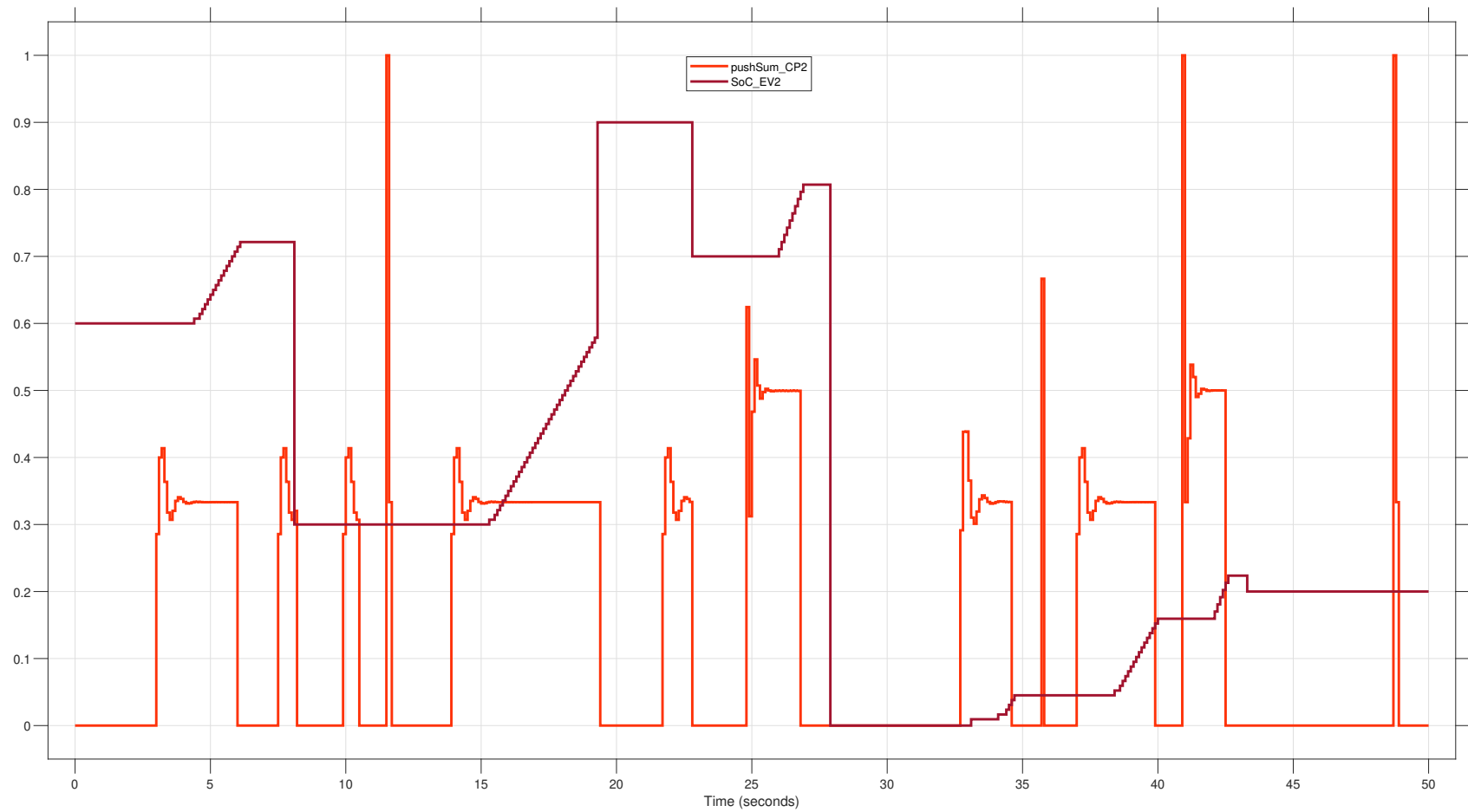


Fig. 26: Grafico che mostra il valore del *SoC* del veicolo connesso al CP2 (funzione rappresentata dal colore più scuro) in relazione alla stima del valore medio calcolata attraverso il modulo *Push-Sum* del punto di ricarica 2 (funzione dal colore più chiaro).

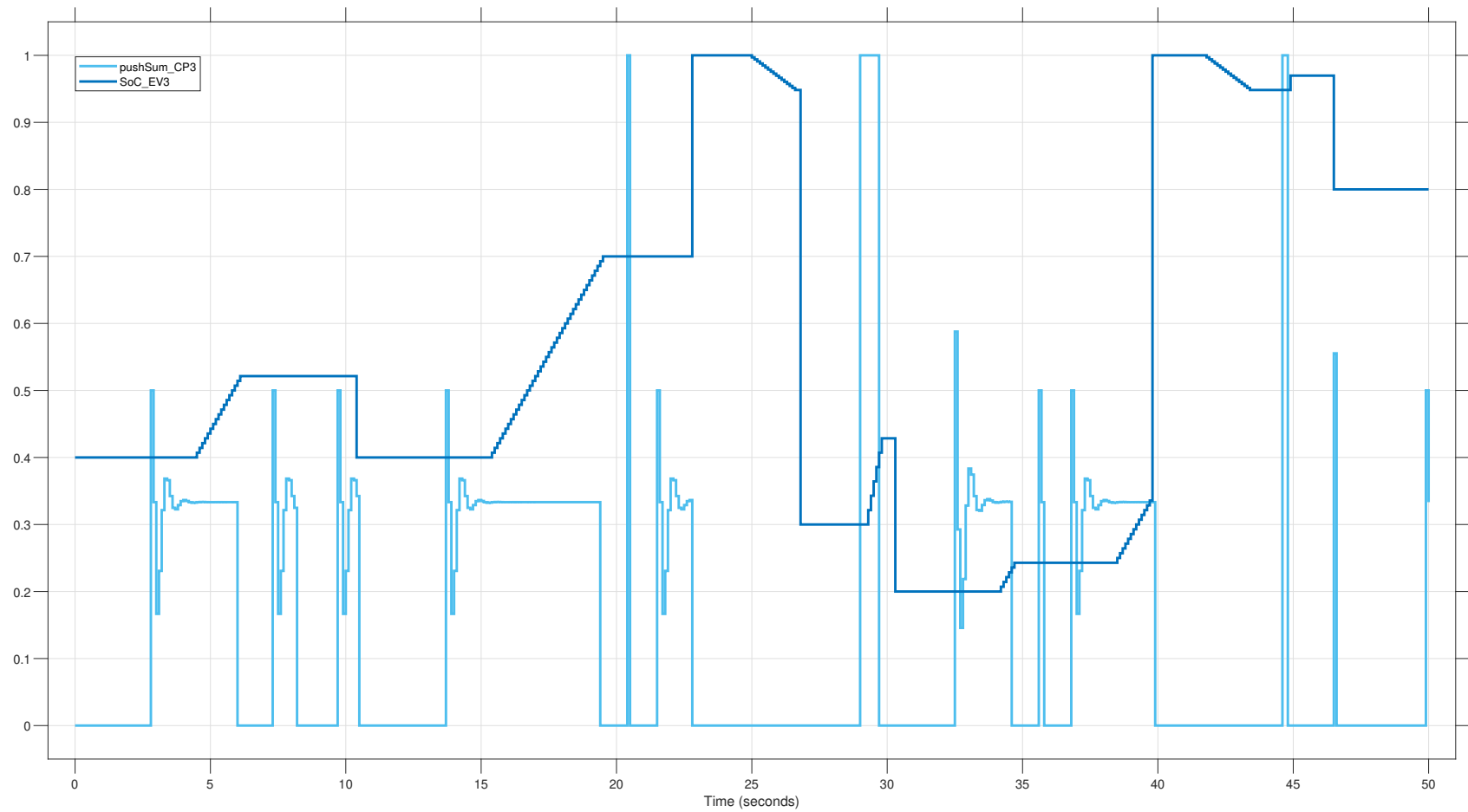


Fig. 27: Grafico che mostra il valore del *SoC* del veicolo connesso al CP3 (funzione rappresentata dal colore più scuro) in relazione alla stima del valore medio calcolata attraverso il modulo *Push-Sum* del punto di ricarica 3 (funzione dal colore più chiaro).

VIII. CONCLUSIONI E LAVORI FUTURI

Il lavoro discusso in questo documento ha consentito di valutare le potenzialità dei concetti, e delle conoscenze apprese durante il corso di "Analisi e Controllo di Sistemi Cyberfisici". Durante la fase di implementazione del modello, il gruppo di lavoro si è dovuto ingegnare al fine di risolvere diverse problematiche che si sono palesate nell'applicare i concetti teorici appresi. Quindi, il *team* è stato in grado di cercare le conoscenze necessarie, e di applicarle per superare i diversi ostacoli incontrati, in linea con quanto espresso dalla filosofia citata all'inizio del corso: "*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)*".

Sviluppi futuri della soluzione proposta potrebbero interessare diversi aspetti del progetto, tra cui:

- Estendere i modelli dei sistemi di base, ossia quelli del veicolo elettrico e del punto di ricarica. Quindi, dettagliarli secondo le esigenze di contesto;
- Modificare l'implementazione del modello, offrendo la possibilità di considerare una rete di punti di ricarica non eterogenei. Inoltre, sarebbe interessante valutare come diverse topologie di rete influiscano sui tempi di convergenza dell'algoritmo *Push-Sum*, utilizzato per giungere a consenso circa le aliquote di potenza che ciascun CP deve erogare;
- Implementare il modello *dataflow* ricorrendo a soluzioni diverse rispetto a quella usata in questo documento (libreria *Simulink SimEvents*). A tal proposito, si è notato che, in implementazioni di schemi complessi, la libreria non offra strumenti sufficienti per ridurre l'*effort* del processo di implementazione.

Concludendo, si sottolinea che si è acquisita consapevolezza sul saper gestire scenari complessi grazie alle conoscenze apprese. Infatti, seppur con dovute semplificazioni, è stato possibile strutturare una soluzione a un problema mai gestito prima d'ora dal gruppo di sviluppo.

L'intero progetto, compreso di file di test, è stato reso disponibile su GitHub in un repository consultabile [qui](#).

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]. Disponibile alla pagina: <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]. Disponibile alla pagina: <https://doi.org/10.1016/j.apenergy.2008.11.021>.
- [3] E-Station. Guida alle auto elettriche. Disponibile alla pagina: <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. Maggiori informazioni sulla libreria sono disponibili alla pagina <https://it.mathworks.com/products/simulink-test.html>.

- [6] Márk Jelasity, *Gossip-based Protocols for Large-scale Distributed Systems*, DSc Dissertation, Szeged, 2013.