ALMA MATER STUDIORUM – UNIVERSITÀ DI BOLOGNA
CAMPUS DI CESENA

DIPARTIMENTO DI INFORMATICA – SCIENZA E INGEGNERIA
Corso di Laurea in Ingegneria e Scienze Informatiche

PROGETTAZIONE E SVILUPPO DI UN'ECU SOFTWARE-DEFINED CON DIGITAL TWIN SERVER-SIDE PER MOTORI A COMBUSTIONE INTERNA MONOCILINDRICI

Elaborato in
EMBEDDED SYSTEMS
AND INTERNET OF THINGS

Relatore
Prof. ALESSANDRO RICCI

Presentata da
ALESSANDRO PORCHEDDU

Corelatore
GABRIELE GNANI

Sessione di Laurea – Luglio 2026
Anno Accademico 2025 – 2026

"What I cannot create, I do not understand."
— Richard P. Feynman

Indice

Introduzione

Nei moderni sistemi automotive la centralina di controllo motore, comunemente indicata come ECU, non è più soltanto un dispositivo che genera i segnali di accensione, ma una piattaforma software attraverso la quale il comportamento del veicolo viene osservato, controllato e protetto. Telemetria, diagnostica, registrazione dei dati e aggiornamento del software sono ormai parte integrante del controllo motore, tanto nelle applicazioni di serie quanto nelle competizioni. Questa evoluzione non ha però raggiunto in modo uniforme tutti i settori: nei motori monocilindrici a due e quattro tempi alimentati a carburatore, l'elettronica di controllo è rimasta in gran parte quella di decenni fa, chiusa, non configurabile e priva di strumenti diagnostici.

Le conseguenze di questa problematica ricadono direttamente sulle attività di messa a punto e di manutenzione. In assenza di telemetria e diagnostica, la calibrazione continua a dipendere da prove empiriche e dall'esperienza dell'operatore, mentre le condizioni operative potenzialmente dannose per il propulsore possono essere individuate soltanto dopo che ne hanno compromesso il funzionamento. Le centraline aftermarket più evolute colmano una parte di questo divario, ma rimangono focalizzate sull'accensione e sulla calibrazione delle prestazioni, laddove l'integrazione tra controllo real-time, osservabilità e strumenti di validazione resta limitata o affidata a dispositivi separati.

La tesi nasce da questa disparità e si propone di verificare se, e a quali condizioni, i principi ormai consolidati nelle ECU automotive e motorsport possano essere trasferiti in questo dominio, senza compromettere i vincoli temporali imposti dal controllo del motore. L'obiettivo del lavoro è la progettazione e lo sviluppo di un'ECU *software-defined* per motori monocilindrici, affiancata da un Digital Twin server-side che ne conserva la configurazione, la cronologia operativa e le sessioni di utilizzo. L'ipotesi progettuale che attraversa l'intero lavoro è che tale trasferimento sia possibile a condizione di trattare la separazione tra il percorso critico di controllo e i servizi di supporto come un vincolo di progetto e non come un'ottimizzazione da introdurre a posteriori.

Il lavoro copre l'intero arco dello sviluppo, dall'analisi del dominio e dei requisiti alla definizione di un'architettura organizzata in sottosistemi con responsabilità e proprietà delle informazioni esplicite, supervisionata da una macchina a stati asincrona che stabilisce la modalità operativa della centralina. L'architettura è stata quindi implementata in un firmware C++ basato su ESP-IDF e FreeRTOS, eseguito sui due core di un ESP32-S3 e integrato in una PCB progettata appositamente per acquisire i segnali del motore e comandarne gli attuatori. Il prototipo risultante è stato infine convalidato su un banco prova costruito ad hoc, che emula a tutti gli effetti un motore generico e rende ripetibili le prove.

La validazione segue una metodologia articolata su più livelli, dai test deterministici eseguibili su host fino agli scenari completi sul banco, e mantiene esplicito il perimetro di ciò che ogni prova vuole validare. Il prototipo dimostra la fattibilità dell'approccio proposto, mentre l'idoneità all'installazione su veicolo richiede il completamento delle verifiche discusse nell'ultimo capitolo.

La tesi è organizzata come segue. Il primo capitolo descrive il dominio applicativo e lo stato dell'arte, individuando lo spazio progettuale nel quale si colloca l'idea di progetto e il ruolo assegnato al Digital Twin. Il secondo capitolo presenta l'analisi dei requisiti, definendo gli attori, il confine del sistema e i requisiti funzionali e non funzionali. Il terzo capitolo definisce l'architettura del sistema: la decomposizione dell'ECU in sottosistemi, la macchina a stati che ne supervisiona la modalità operativa, il client utente e il Digital Twin. Il quarto capitolo descrive lo sviluppo del prototipo, dal firmware alla progettazione della scheda. Il quinto capitolo presenta la metodologia e i risultati della validazione, insieme ai limiti delle prove svolte. Le conclusioni discutono infine i risultati raggiunti e gli sviluppi futuri del progetto.

Capitolo 1

Contesto Applicativo e Obiettivi del Progetto

La prima sezione di questo capitolo descrive il ruolo delle ECU nei sistemi automotive e le limitazioni che caratterizzano oggi il settore dei motori endotermici monocilindrici; la seconda esamina lo stato dell'arte attuale, dalle architetture elettroniche dei veicoli moderni alle centraline aftermarket disponibili per questo ambito. Dal confronto emerge lo spazio progettuale nel quale si inserisce l'idea di progetto, presentata nella terza sezione e completata, nella quarta, dalla descrizione del ruolo assegnato al Digital Twin.

1.1 Descrizione del dominio

Le centraline elettroniche, denominate con il termine ECU, acronimo di Electronic Control Unit, rappresentano uno degli elementi fondamentali dei moderni sistemi automotive. Un'ECU è un sistema embedded dedicato al controllo di specifiche funzioni del veicolo e, nel caso del motore, ha il compito di acquisire informazioni dai sensori, elaborarle secondo opportune strategie di controllo e generare i comandi necessari agli attuatori.

Nel corso degli anni, le ECU si sono evolute da semplici dispositivi di controllo a piattaforme software sempre più sofisticate. Nelle applicazioni automotive moderne e nelle competizioni motociclistiche di alto livello, le centraline non si limitano a gestire l'accensione o l'iniezione, ma integrano funzionalità avanzate quali telemetria, diagnostica, registrazione dei dati, configurazione dei parametri operativi e sistemi di protezione del motore. Questi strumenti consentono di osservare il comportamento del sistema, adattarne il funzionamento alle esi-

genze dell'applicazione e intervenire tempestivamente in presenza di condizioni anomale.

Tale evoluzione non ha però interessato in modo uniforme tutti i settori. Nel contesto dei motori endotermici monocilindrici, molte delle soluzioni attualmente disponibili si basano ancora su tecnologie e architetture sviluppate diversi decenni fa. Le centraline impiegate in questo ambito offrono generalmente funzionalità limitate: la telemetria è spesso assente, gli strumenti diagnostici sono ridotti o inesistenti e le possibilità di configurazione risultano molto ristrette. In molti casi non è possibile modificare facilmente la curva di anticipo, adattare la risposta all'acceleratore o personalizzare il comportamento della centralina in funzione delle caratteristiche del motore e delle esigenze dell'utilizzatore.

Anche gli strumenti dedicati alla sicurezza e alla salvaguardia del propulsore risultano spesso limitati. L'assenza di meccanismi di monitoraggio e protezione rende più difficile individuare condizioni operative potenzialmente dannose e intervenire prima che possano causare guasti o degrado delle prestazioni. Di conseguenza, molte attività di calibrazione e verifica continuano a dipendere da prove empiriche e dall'esperienza dell'operatore.

1.2 Stato dell'arte

Nel settore automotive moderno, lo sviluppo delle ECU si inserisce in una trasformazione più ampia dell'architettura elettronica del veicolo. Le architetture tradizionali, basate su numerose centraline dedicate a singole funzioni, si stanno progressivamente evolvendo verso strutture più centralizzate o a zone, nelle quali unità di calcolo ad alte prestazioni e zone controller aggregano sensori, attuatori e funzioni software distribuite nel veicolo [3, 10]. Questo cambiamento è legato al concetto di *software-defined vehicle*, in cui una parte crescente del valore e del comportamento del veicolo dipende dal software, dalla possibilità di aggiornamento, dalla gestione dei dati e dall'integrazione con servizi esterni [7].

In questo contesto, l'ECU non viene più considerata soltanto come un controllore locale, ma come un nodo di un sistema elettronico complesso. Le piattaforme software automotive moderne, come AUTOSAR Classic e AUTOSAR Adaptive, riflettono questa evoluzione: la prima è orientata a sistemi embedded con requisiti di prevedibilità, sicurezza e risposta real-time, mentre la seconda è pensata per ECU ad alte prestazioni, aggiornabili e riconfigurabili dinamicamente [2]. Accanto al controllo real-time assumono quindi un ruolo centrale anche la diagnostica, la comunicazione, la gestione della con-

figurazione, la cybersecurity, gli aggiornamenti software e la validazione del comportamento del sistema.

Anche nel motorsport e nelle applicazioni motociclistiche avanzate si osserva una progressiva integrazione tra controllo motore, acquisizione dati e strumenti di calibrazione. Le ECU da competizione sono spesso accompagnate da sistemi di data logging, dashboard configurabili, strumenti di analisi e software di calibrazione, con l'obiettivo di rendere il veicolo più osservabile e di supportare decisioni tecniche basate sui dati [6, 1]. In ambito diagnostico, anche il settore delle due ruote ormai utilizza interfacce in grado di leggere parametri dalle centraline, eseguire configurazioni, registrare sessioni diagnostiche e comunicare tramite protocolli automotive moderni come CAN FD [9].

Il caso dei motori monocilindrici a combustione alimentati a carburatore presenta tuttavia una situazione meno uniforme. In molti modelli di serie, come ad esempio le motociclette da motocross di media cilindrata, l'elettronica di controllo è rimasta sostanzialmente invariata rispetto a soluzioni introdotte diversi decenni fa: la centralina si occupa principalmente della gestione dell'accensione attraverso mappe predefinite, senza offrire alcuna capacità di configurazione, diagnostica o acquisizione dati. Nel mercato aftermarket sono disponibili alcune centraline programmabili e unità di accensione evolute, capaci di gestire mappe di anticipo, limitatori di regime, controllo della valvola di scarico (*power valve*), acquisizione dati e configurazione tramite PC o smartphone [11, 8]. Tali soluzioni sono particolarmente diffuse nel settore motociclistico sportivo e nelle competizioni, dove la possibilità di adattare il comportamento del motore alle diverse condizioni operative rappresenta un requisito fondamentale.

Nonostante queste evoluzioni, molte delle soluzioni disponibili risultano ancora focalizzate principalmente sulle funzioni di accensione e calibrazione delle prestazioni. Rispetto alle ECU automotive moderne, l'integrazione tra controllo real-time, telemetria, diagnostica, gestione della configurazione, protezione del motore e strumenti di validazione risulta spesso limitata o affidata a dispositivi separati. Di conseguenza, l'osservabilità del sistema e la disponibilità di informazioni utili per l'analisi del comportamento del motore rimangono generalmente inferiori rispetto a quanto avviene in altri settori applicativi.

Da questo quadro emerge uno spazio progettuale intermedio: trasferire in un'applicazione oramai quasi priva di nuovi sviluppi alcuni principi già consolidati nelle ECU automotive e motorsport più evolute, mantenendo però un'architettura semplice, controllabile e compatibile con i vincoli di un sistema embedded real-time.

1.3 Idea di progetto

L'idea alla base del progetto nasce dallo spazio progettuale appena individuato: realizzare un'ECU per motori endotermici monocilindrici che non si limiti alla sola generazione dei segnali di controllo, ma integri funzionalità quali telemetria, diagnostica, configurazione dei parametri operativi e meccanismi di protezione del motore.

La centralina viene quindi considerata come una piattaforma embedded completa, capace di acquisire informazioni dal motore, elaborarle in tempo reale e renderle disponibili per l'osservazione e la validazione del comportamento del sistema. Questa impostazione permette di superare il modello tradizionale dell'ECU come dispositivo chiuso e difficilmente ispezionabile, introducendo invece un'architettura più trasparente, configurabile ed estendibile.

Dal punto di vista funzionale, il progetto prevede che l'ECU sia in grado di gestire i segnali provenienti dai sensori, determinare lo stato operativo del motore e generare i comandi necessari agli attuatori rispettando i vincoli temporali imposti dal funzionamento del propulsore. Parallelamente, il sistema deve poter raccogliere e trasmettere dati di funzionamento, rilevare condizioni anomale, conservare configurazioni persistenti e fornire strumenti utili alla verifica delle strategie implementate.

Un aspetto centrale dell'idea progettuale è la separazione tra controllo del motore e funzionalità di supporto. Il controllo real-time deve avere la priorità assoluta, mentre telemetria, diagnostica, configurazione e registrazione dei dati devono essere progettate in modo da non compromettere l'affidabilità della centralina. Questa distinzione consente di introdurre funzionalità avanzate senza trasformarle in un rischio per il corretto funzionamento del motore.

L'ECU proposta non viene quindi intesa esclusivamente come un componente elettronico dedicato all'accensione o alla gestione di singoli attuatori, ma come il nucleo di un sistema più ampio. Attraverso l'integrazione con strumenti software esterni, essa può rendere il motore osservabile durante il funzionamento, permettere la modifica controllata dei parametri e supportare attività di analisi, calibrazione e validazione. In questo modo il progetto mira a costruire una base tecnica su cui sviluppare progressivamente funzioni più evolute, mantenendo una struttura coerente e adatta a un'applicazione embedded real-time.

Un aspetto aggiuntivo dell'idea di progetto è il Digital Twin, il cui scopo è fornire una rappresentazione persistente server-side dell'ECU fisica, della sua configurazione e della sua storia operativa. Il Digital Twin non sostituisce la centralina e non partecipa al controllo in tempo reale del motore, ma raccoglie

e organizza le informazioni necessarie a descrivere ciò che l'ECU ha osservato, deciso e comandato durante il funzionamento.

Il Digital Twin consente di associare ogni sessione registrata all'ECU che l'ha prodotta, alle revisioni hardware e software in uso, alle mappe di accensione, alle calibrazioni e ai parametri attivi al momento dell'acquisizione. In questo modo, l'ultimo stato noto della centralina e le sessioni precedenti non sono considerati dati isolati, ma parti di una stessa cronologia tecnica consultabile, confrontabile e verificabile.

Attraverso questa rappresentazione, il tecnico può ricostruire il comportamento dell'ECU a partire dai dati registrati, analizzare il contesto nel quale si sono verificati eventi od errori, riprodurre una sessione mediante *playback* o *replay* deterministico e confrontare il comportamento registrato con quello ottenibile usando mappe, parametri o configurazioni alternative.

Il Digital Twin ha quindi una funzione descrittiva, diagnostica e comparativa. Descrive lo stato e la storia dell'ECU, supporta l'analisi delle decisioni di controllo e permette di valutare modifiche software o di calibrazione senza alterare i dati originali. I risultati prodotti dal server rimangono distinti dalle osservazioni acquisite dall'ECU fisica, così da mantenere sempre chiara la differenza tra ciò che è avvenuto durante il funzionamento e ciò che è stato calcolato successivamente.

Capitolo 2

Analisi dei requisiti

2.1 Attori e confine del sistema

Il sistema principale è costituito dall'ECU, responsabile dell'acquisizione dei segnali, dell'elaborazione delle strategie di controllo e dell'attuazione dei comandi verso il motore. Gli attori esterni sono quattro: il motore, il pilota, il tecnico e il server remoto.

Il motore rappresenta il processo fisico da controllare: da esso vengono acquisiti i segnali necessari alla ricostruzione del suo stato. Il sensore fondamentale per il funzionamento del motore è il *pick-up*, poiché fornisce il regime di rotazione e la posizione angolare dell'albero motore. Inoltre, vengono impiegati altri sensori analogici per la misura della temperatura dei gas di scarico, indicata come *EGT*, e per l'acquisizione di eventuali segnali associati alla detonazione attraverso il *knock sensor*. Al tempo stesso, il motore costituisce il destinatario finale delle azioni elaborate dall'ECU, tra cui il comando di accensione della scintilla, la correzione della carburazione tramite il *power jet* e il controllo della valvola di scarico.

Il pilota interagisce con il sistema attraverso comandi diretti, legati alla guida del veicolo. Alcuni di questi segnali richiedono una risposta tempestiva e devono essere gestiti con priorità elevata, come l'intenzione di cambiare marcia proveniente dal *quickshifter* o le variazioni dell'apertura dell'acceleratore, acquisite tramite il *TPS*. Altri comandi, come la selezione della mappatura di anticipo o l'attivazione di strategie specifiche, non richiedono necessariamente la stessa latenza, ma devono comunque essere acquisiti e applicati in modo coerente con lo stato corrente della centralina.

Il tecnico rappresenta invece l'utente incaricato della configurazione, della calibrazione e della diagnosi. La sua interazione avviene attraverso il client utente eseguito su un computer o uno smartphone, che può fornire le proprie funzionalità tramite un'applicazione web senza vincolare a questa scelta la struttura logica del sistema. Attraverso il client il tecnico osserva la telemetria, modifica parametri e mappe di anticipo, verifica lo stato degli ingressi e degli attuatori e consulta le sessioni e i risultati resi disponibili dal Digital Twin.

Il server remoto ha un ruolo distinto rispetto al ciclo di controllo. In esso risiede il Digital Twin di ogni ECU, che è utilizzato per conservare dati, versioni delle configurazioni, informazioni diagnostiche, statistiche sulla telemetria ed eventualmente per distribuire aggiornamenti software. La disponibilità della rete o del server non costituisce una condizione necessaria per il funzionamento dell'ECU.

2.2 Requisiti

Nell'analisi del sistema sono stati individuati i seguenti requisiti:

2.2.1 Requisiti funzionali

RF-001 Il sistema deve acquisire un riferimento di posizione e velocità del motore per consentire il controllo sincronizzato delle funzioni ad esso associate.

RF-002 Il sistema deve determinare lo stato dinamico del motore, inclusi regime di rotazione, variazioni di velocità e stato di sincronizzazione.

RF-003 Il sistema deve validare la coerenza dei segnali utilizzati per il controllo motore e rilevare condizioni non plausibili.

RF-004 Il sistema deve consentire l'esecuzione delle strategie motore solo quando le informazioni necessarie risultano affidabili.

RF-005 Il sistema deve calcolare il comando di accensione sulla base dello stato del propulsore e delle strategie configurate.

RF-006 Il sistema deve supportare la configurazione dei parametri necessari alla sincronizzazione e al controllo del motore.

RF-007 Il sistema deve acquisire i principali parametri operativi del motore attraverso sensori dedicati.

RF-008 Il sistema deve normalizzare, validare e rendere disponibili i dati provenienti dai sensori alle funzioni di controllo.

RF-009 Il sistema deve supportare procedure di calibrazione dei sensori configurabili dall'utente.

RF-010 Il sistema deve applicare strategie di fallback in caso di mancanza o invalidità dei dati sensoriali.

RF-011 Il sistema deve gestire configurazioni memorizzate e ripristinarle automaticamente all'avvio.

RF-012 Il sistema deve validare configurazioni e strategie prima della loro applicazione operativa.

RF-013 Il sistema deve utilizzare configurazioni di sicurezza predefinite in caso di dati mancanti, corrotti o non validi.

RF-014 Il sistema deve consentire la configurazione e il tuning delle strategie motore tramite interfacce dedicate.

RF-015 Il sistema deve fornire strumenti di monitoraggio in tempo reale dello stato del motore e dell'ECU.

RF-016 Il sistema deve rendere disponibili informazioni operative e diagnostiche tramite telemetria.

RF-017 Il sistema deve monitorare continuamente lo stato di salute dei sensori e delle funzioni interne.

RF-018 Il sistema deve rilevare, classificare e registrare condizioni di errore e anomalie operative.

RF-019 Il sistema deve acquisire parametri termici rilevanti per il monitoraggio dello stato del motore.

RF-020 Il sistema deve fornire informazioni sullo stato termico del motore e sulle relative tendenze evolutive.

RF-021 Il sistema deve rilevare fenomeni potenzialmente dannosi per il motore e renderli disponibili alle strategie di controllo.

RF-022 Il sistema deve supportare ingressi dedicati all'interazione del pilota con le strategie motore.

RF-023 Il sistema deve consentire la selezione e il cambio delle strategie operative durante il funzionamento.

RF-024 Il sistema deve supportare la gestione di più mappe o profili di funzionamento selezionabili.

RF-025 Il sistema deve fornire informazioni sullo stato corrente delle strategie attive e delle modalità operative.

RF-026 Il sistema deve supportare funzionalità di diagnostica e manutenzione del sistema.

RF-027 Il sistema deve consentire la simulazione delle principali condizioni operative per attività di verifica e validazione.

RF-028 Il sistema deve supportare la raccolta e la consultazione dei dati necessari all'analisi delle prestazioni e degli errori.

RF-029 Il sistema deve garantire la disponibilità delle informazioni necessarie al controllo, alla diagnostica e alla supervisione del motore.

RF-030 Il sistema deve mantenere sul server un Digital Twin persistente per ogni ECU registrata, associandovi l'identità della centralina, la revisione hardware e le versioni di firmware, mappe, configurazione e calibrazione.

RF-031 Il sistema deve associare ogni sessione registrata ai riferimenti temporali e alle versioni effettivamente utilizzate dalla ECU durante l'acquisizione.

RF-032 Il sistema deve consentire la ricostruzione di una sessione registrata come sequenza ordinata di osservazioni, decisioni di controllo, comandi agli attuatori, eventi ed errori.

RF-033 Il sistema deve supportare come operazioni distinte il *playback* dei dati registrati, il *replay* deterministico della logica di controllo e il confronto con mappe o parametri alternativi.

RF-034 Il sistema deve preservare i dati originali ricevuti dalla ECU e conservare separatamente i risultati prodotti da *replay*, simulazioni e modelli server-side, mantenendone identificabile la provenienza.

RF-035 Il client utente deve inoltrare al server le informazioni dell'ECU destinate al Digital Twin e rendere disponibili al tecnico i risultati restituiti dai servizi server-side.

2.2.2 Requisiti non funzionali

RNF-001 Le funzioni critiche di controllo motore devono operare con comportamento deterministico e prevedibile.

RNF-002 Le funzioni critiche devono essere isolate dalle funzionalità non essenziali quali interfacce utente, comunicazioni e servizi ausiliari.

RNF-003 Il sistema deve garantire la precisione temporale necessaria al corretto controllo del motore.

RNF-004 Il sistema deve minimizzare la latenza tra acquisizione degli eventi critici e relativa elaborazione.

RNF-005 Le attività non critiche non devono compromettere le prestazioni delle funzioni di controllo motore.

RNF-006 I meccanismi di comunicazione e telemetria non devono essere bloccanti rispetto alle funzioni critiche.

RNF-007 Il sistema deve avviarsi e operare in condizioni sicure fino alla verifica della propria integrità operativa.

RNF-008 Il sistema deve essere in grado di rilevare e ripristinarsi da condizioni di blocco o malfunzionamento software.

RNF-009 Le configurazioni memorizzate devono essere protette da corruzione e perdita di dati.

RNF-010 Le strategie di controllo devono mantenere un comportamento coerente indipendentemente dal carico delle altre funzioni del sistema.

RNF-011 L'architettura software deve essere modulare e separare chiaramente responsabilità e domini funzionali.

RNF-012 Il sistema deve essere progettato per favorire manutenibilità, evoluzione e riutilizzo dei componenti software.

RNF-013 Le logiche di controllo devono essere verificabili e testabili indipendentemente dall'hardware finale.

RNF-014 Il sistema deve supportare attività di simulazione e validazione ripetibili e automatizzabili.

RNF-015 Le interfacce utente devono fornire una rappresentazione chiara dello stato operativo del sistema.

RNF-016 Ogni informazione resa disponibile dal sistema deve essere accompagnata, ove applicabile, da elementi che ne attestino la validità e l'affidabilità.

RNF-017 Il sistema deve essere robusto rispetto a rumore, disturbi e

condizioni operative non ideali.

RNF-018 La telemetria deve garantire coerenza, tracciabilità e contestualizzazione dei dati trasmessi.

RNF-019 In condizioni di risorse limitate, il sistema deve preservare le informazioni critiche per sicurezza e diagnostica.

RNF-020 Gli eventi e i dati registrati devono essere mantenuti in corretto ordine temporale.

RNF-021 Le funzionalità diagnostiche avanzate devono essere configurabili e non interferire con il funzionamento operativo.

RNF-022 L'accesso alle funzioni critiche deve essere protetto da adeguati meccanismi di autorizzazione e conferma.

RNF-023 Il sistema deve adottare strategie di degradazione controllata in presenza di errori o perdita di informazioni.

RNF-024 L'architettura deve essere estendibile e consentire l'introduzione di nuove funzionalità senza modifiche sostanziali ai componenti esistenti.

RNF-025 L'ECU deve continuare a svolgere le funzioni di controllo anche quando il client, la rete, il server o il Digital Twin non sono disponibili.

RNF-026 Lo stato operativo più recente conservato dal Digital Twin deve includere l'istante di osservazione, affinché un'informazione non aggiornata non venga presentata come stato fisico attuale.

RNF-027 Ogni record rilevante e ogni risultato generato sul server devono essere riconducibili all'ECU fisica, alla sessione registrata e alle versioni di riferimento.

RNF-028 Le stime e le simulazioni prodotte sul server devono rimanere chiaramente distinte dalle informazioni acquisite dall'ECU e non devono assumere autorità sulle decisioni di controllo real-time.

Capitolo 3

Architettura del sistema

3.1 Panoramica architetturale

L'architettura della centralina è stata definita in modo da astrarre la struttura software dalle specifiche scelte di implementazione e dall'hardware sul quale viene adottata.

Il processo fisico in analisi comprende la rotazione del motore, la combustione e il comportamento degli organi che ne influenzano l'erogazione. La ECU osserva tale processo acquisendo la posizione angolare dell'albero motore fornita dal pick-up, la posizione dell'acceleratore, la temperatura dei gas di scarico, le vibrazioni associate a possibili fenomeni di detonazione e i comandi provenienti dal pilota. A partire da queste informazioni ricostruisce lo stato operativo del motore e determina le azioni da applicare. Le principali azioni riguardano il comando di accensione della scintilla, la regolazione del power jet e il controllo della valvola di scarico.

Le interazioni del mondo esterno con l'ECU sono divise in due categorie: i segnali del motore e i comandi del pilota alimentano il flusso di controllo, mentre la web application, gli strumenti di calibrazione e gli eventuali comandi locali appartengono al flusso di configurazione e supervisione. La separazione tra questi flussi permette di attribuire ad ognuno di essi i requisiti di temporizzazione, validità e affidabilità coerenti con il suo impiego.

L'architettura distingue quindi un dominio di controllo da un dominio di supervisione e configurazione. Il primo comprende l'osservazione dell'ambiente, l'interpretazione dello stato, le decisioni di controllo e il coordinamento delle azioni che devono reagire agli eventi del motore con latenza limitata e comportamento prevedibile. Il secondo comprende la gestione dei parametri, la

comunicazione con le interfacce utente, la raccolta della telemetria e l'accesso al server remoto. Queste attività non partecipano direttamente alla temporizzazione degli attuatori e non devono bloccare né ritardare il percorso di controllo.

La diagnostica e la gestione degli errori costituiscono una responsabilità trasversale ai due domini. Ogni informazione destinata al controllo è accompagnata, quando necessario, da uno stato di validità; le condizioni anomale vengono rilevate e propagate ai sottosistemi interessati, che possono adottare una strategia di degradazione controllata. La telemetria rende osservabili lo stato del motore, le decisioni applicate e lo stato di salute dell'ECU senza diventare una dipendenza necessaria per il funzionamento del controllo.

Il percorso di supervisione viene esteso al di fuori della centralina tramite il client utente, che comunica in locale con l'ECU e attraverso internet con il server remoto. Il client mostra al tecnico lo stato attuale del sistema e inoltra al server i dati da conservare. In questo modo svolge la funzione di *gateway*, senza assumere il ruolo di componente responsabile della conservazione dello storico. Sul server risiede il Digital Twin associato a ciascuna ECU registrata; la centralina e il server, invece, non comunicano direttamente. Di conseguenza, la mancanza del collegamento remoto interrompe l'aggiornamento e la consultazione dei servizi server-side, ma non introduce dipendenze nel percorso di controllo.

La decomposizione interna dell'ECU segue dei principi fondamentali: separazione delle responsabilità, proprietà univoca dello stato e dipendenze ben definite e limitate tra i sottosistemi. Questa organizzazione consente ai sottosistemi logici di cooperare attraverso flussi informativi definiti, limitando l'accoppiamento tra funzioni con priorità differenti. In particolare, le attività di comunicazione, visualizzazione e gestione remota possono essere aggiornate indipendentemente dalle strategie motore, mentre il percorso di controllo conserva vincoli temporali e modalità di gestione degli errori chiaramente identificabili. Su questa struttura si basa la successiva descrizione, che esamina la decomposizione interna dell'ECU e i relativi flussi. Definite le informazioni esposte all'esterno, il capitolo passa quindi all'organizzazione del client utente e alle regole con cui mantiene separati il monitoraggio locale e l'accesso ai servizi remoti, per concludere con lo stato e le elaborazioni affidate al Digital Twin.

3.2 Architettura interna dell'ECU

La decomposizione logica introdotta nella sezione precedente è organizzata attraverso due flussi coordinati. Il flusso di controllo comprende i seguenti sottosistemi: *Environment Observation*, *State Interpretation*, *Control and Decision* e *Action Coordination*. A esso si affianca il flusso di supervisione della modalità operativa, governato da una *asynchronous finite state machine*, di seguito AFSM. Il primo produce le azioni richieste al sistema fisico; il secondo stabilisce il comportamento dell'ECU in funzione dei dati forniti dai sottosistemi.

La Figura 3.1 sintetizza il percorso operativo della centralina attraverso i flussi informativi scambiati tra i sottosistemi e con l'ambiente. L'*Environment Observation* si occupa di rappresentare nell'ECU i segnali fisici: il sottosistema acquisisce gli ingressi provenienti dai sensori e dai comandi del pilota, li normalizza e li associa a delle informazioni temporali, necessarie successivamente per ricostruirne la sequenza. Alla misura viene affiancato uno stato che ne descrive validità e condizioni di acquisizione, in modo che ogni consumatore riceva sia il valore osservato sia gli elementi necessari per stabilirne l'affidabilità. Le osservazioni convalidate vengono quindi pubblicate al sottosistema di *State Interpretation*, mentre disponibilità dei servizi, condizioni anomale ed eventuali errori vengono resi disponibili anche al flusso di supervisione.

Lo *State Interpretation* riceve queste osservazioni e le combina in un'unica rappresentazione coerente dello stato del motore e del sistema. In questa fase vengono derivate grandezze come il regime di rotazione, il livello di sincronizzazione e lo stato del motore, mantenendo esplicite le caratteristiche non processate delle informazioni di origine. Il risultato è una cattura istantanea riferita a un preciso contesto temporale e a una generazione coerente della configurazione, che permette alle strategie successive di operare su una visione unitaria del sistema. La stessa elaborazione produce eventi come l'acquisizione o la perdita della sincronizzazione, i quali descrivono una variazione dello stato fisico e diventano segnali per la supervisione della modalità operativa.

Parallelamente alla costruzione dello stato interpretato, il *Configuration Management* rende disponibile la configurazione attualmente in uso nell'ECU, identificata dal relativo numero di generazione. Le richieste provenienti dalla web application, dagli strumenti di servizio o dagli altri attori esterni raggiungono questo sottosistema attraverso la *External Interaction*, che le indirizza verso l'interfaccia corrispondente. Le richieste di arresto o ripristino vengono invece inoltrate all'AFSM, mentre le interrogazioni diagnostiche sono affidate alla *Telemetry and Diagnostics*. Quest'ultima riceve inoltre osservazioni, stato interpretato, decisioni, transizioni e stato delle azioni, costruendo una

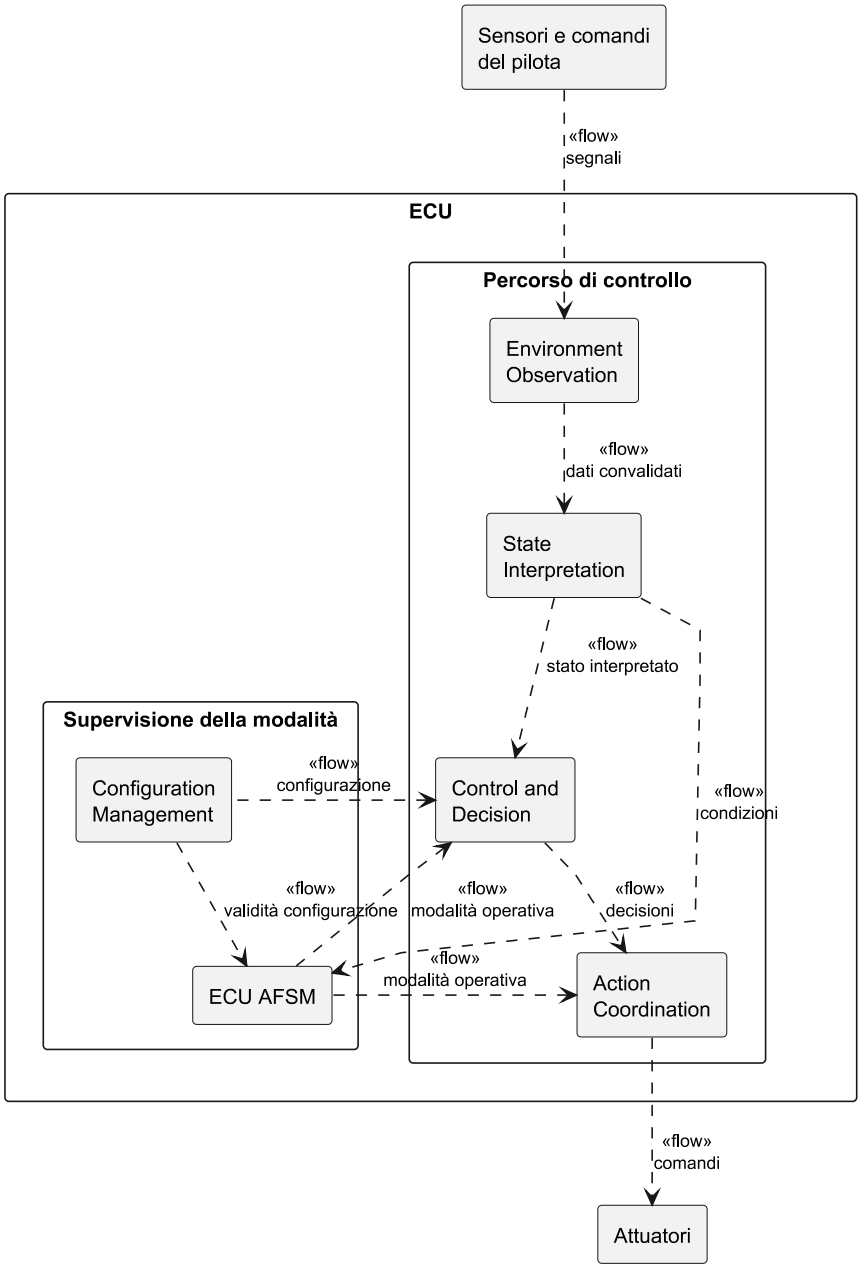

Figura 3.1: Flussi informativi tra il percorso di controllo e la supervisione della modalità operativa.

vista completa dello stato dell'ECU e una cronologia degli eventi attraverso un percorso parallelo a quello di controllo.

Quando lo stato interpretato e la configurazione risultano disponibili, l'AFSM dispone delle informazioni necessarie per stabilire la modalità operativa. A tali informazioni si aggiungono la disponibilità e la validità delle misurazioni pubblicate dall'*Environment Observation* e le condizioni derivate dallo *State Interpretation*. L'AFSM valuta questi elementi per valutare la transizione di stato ed è l'unica responsabile della modalità operativa. Al termine della valutazione pubblica un'istantanea della modalità, destinata sia alle decisioni di controllo sia alla telemetria.

Dopo la *State Interpretation*, la posizione dell'AFSM indica quindi una dipendenza informativa: sincronizzazione e condizioni del motore vengono ricostruite prima di essere valutate come condizioni per le transizioni. Gli stati in cui si può trovare l'AFSM sono elencati nella Tabella 3.1, mentre le relative transizioni sono rappresentate nella Figura 3.2.

Stato	**Descrizione**
`Uninitialized`	ECU in fase di inizializzazione.
`ConfigurationLoading`	Validazione della configurazione memorizzata, predefinita o fornita esternamente.
`Ready`	Configurazione validata: l'ECU è pronta.
`Synchronizing`	Le letture dei sensori sono attive mentre il riferimento del motore viene elaborato.
`Running`	Tutte le condizioni operative sono soddisfatte e le decisioni di controllo in modalità normale sono ammesse.
`DegradedRunning`	Il controllo è disponibile entro i limiti imposti da una condizione non critica.
`ControlInhibited`	Il controllo rimane inibito per mancanza di segnali necessari o per presenza di letture non valide.
`FaultLatched`	Viene richiesto il riconoscimento di un guasto o un intervento di manutenzione.
`Shutdown`	Il controllo dell'ECU è stato intenzionalmente arrestato o il sistema sta uscendo dalla modalità operativa.

Tabella 3.1: Fasi della modalità operativa dell'AFSM.

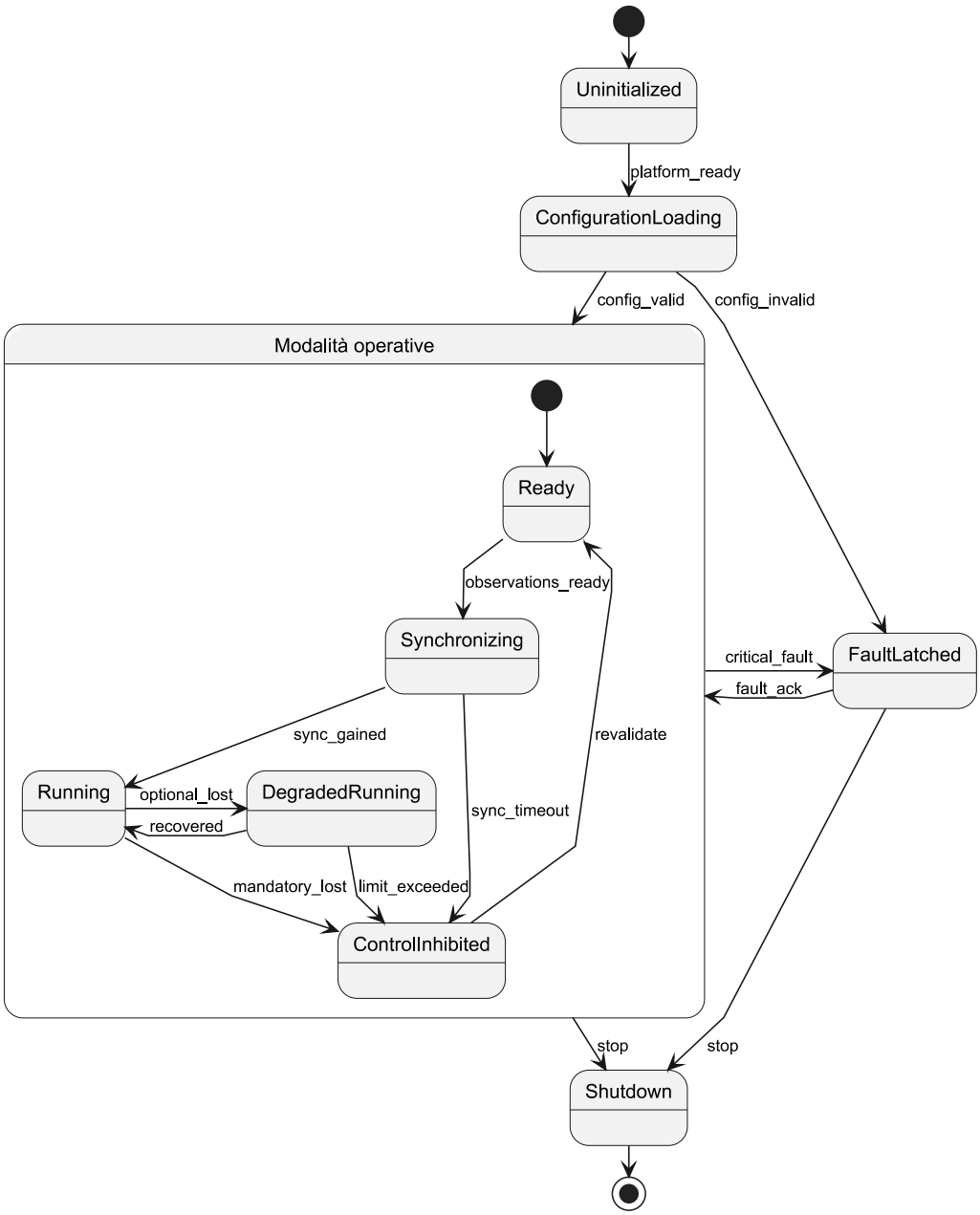

Figura 3.2: Macchina a stati delle modalità operative dell'ECU.

L'ingresso nello stato *Ready* richiede una configurazione valida, le letture dei sensori essenziali e un riferimento temporale; le condizioni opzionali determinano invece quali limiti vengono applicati se si entra nello stato di *DegradedRunning*. La perdita dei sensori vitali conduce all'inibizione del controllo, mentre il recupero del funzionamento avviene in presenza di nuove letture valide e aggiornate.

L'istantanea prodotta dall'AFSM completa il contesto utilizzato dal *Control and Decision*. Questo sottosistema considera lo stato interpretato, la configurazione attiva e la modalità operativa corrente e attua la strategia ammessa da tale combinazione. Da questa valutazione derivano le decisioni per l'accensione della scintilla, il *power jet*, la valvola di scarico e le funzioni accessorie. Ogni decisione conserva il contesto dal quale è stata generata ed esplicita il proprio stato, che può essere di funzionamento normale, limitato, inibito oppure associato a una condizione di errore. Il sottosistema mantiene così la proprietà della strategia di controllo e trasmette all'*Action Coordination* il comportamento da attuare in quel momento.

L'*Action Coordination* costituisce l'ultimo passaggio del percorso e traduce le decisioni ricevute in richieste fisiche ordinate e temporalmente coerenti. Prima dell'applicazione delle richieste confronta il contesto della decisione con la modalità AFSM corrente, la sincronizzazione disponibile, i vincoli temporali e lo stato del percorso di attuazione; in questo modo se la modalità operativa diventa più restrittiva o il riferimento temporale viene perso, le azioni in attesa vengono annullate. Il sottosistema risolve inoltre eventuali conflitti tra le richieste, assegna le priorità necessarie e mantiene lo stato delle azioni accettate, pianificate, applicate, completate o fallite. Gli esiti vengono pubblicati per la successiva valutazione dell'AFSM e come dati per la *Telemetry and Diagnostics*.

3.3 Responsabilità dei sottosistemi

La sequenza operativa descritta finora stabilisce come le informazioni attraversano l'ECU, ma non è sufficiente, da sola, a definire chi possa produrle, modificarle e validarle. Per evitare che una stessa decisione venga presa in punti differenti dell'architettura, ciascun sottosistema mantiene l'autorità su una categoria precisa di informazioni e la rende disponibile mediante interfacce esplicite. La decomposizione che ne deriva accompagna il dato dal contatto con il sistema fisico fino alla sua esposizione verso l'esterno, conservando in ogni passaggio il contesto temporale, la generazione della configurazione e le condizioni di validità necessarie al consumatore successivo.

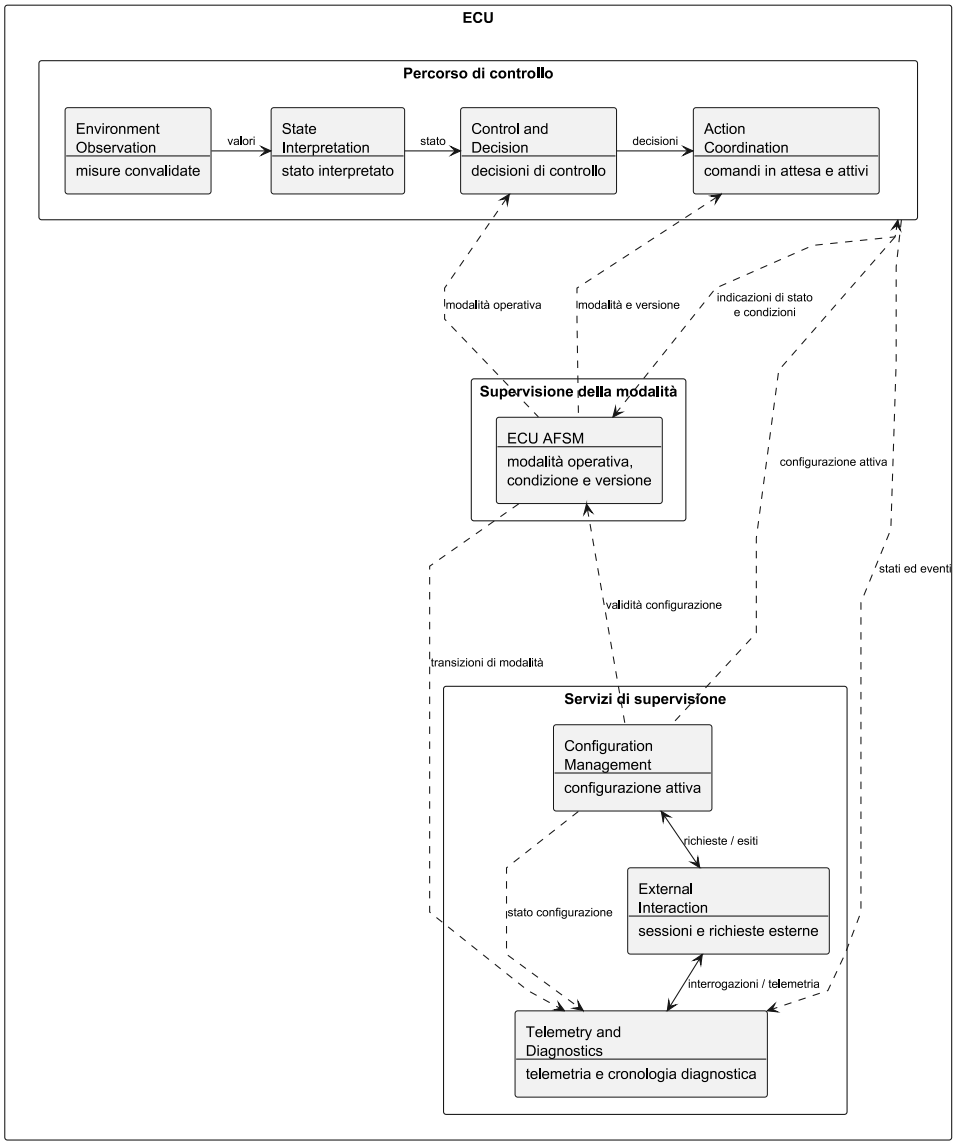

Figura 3.3: Principali dipendenze tra i sottosistemi dell'ECU.

La Figura 3.3 riassume questa assegnazione distinguendo il percorso critico, la supervisione della modalità e i servizi di supervisione. All'interno di ogni componente, il testo posto sotto il separatore identifica l'informazione di cui il sottosistema è proprietario, mentre le dipendenze tratteggiate condensano i flussi trasversali descritti nelle sezioni successive senza rappresentare i dettagli di implementazione.

3.3.1 Environment Observation

Il percorso ha origine nell'*Environment Observation*, che costituisce il confine tra i segnali elettrici provenienti dal veicolo e la loro rappresentazione nel dominio software dell'ECU. Il sottosistema acquisisce il riferimento temporale del pick-up, la posizione dell'acceleratore, le richieste del *quick shifter* e del selettore di mappa, oltre alle misure termiche e alle osservazioni legate alla combustione. I servizi di acquisizione specifici dell'hardware rimangono nascosti dietro questa interfaccia, mentre i valori pubblicati assumono un formato comune che consente al resto dell'architettura di trattarli senza conoscere il circuito o la periferica che li ha prodotti.

L'acquisizione non coincide con la semplice lettura di un valore. Ogni osservazione viene normalizzata, associata all'istante in cui è avvenuta e verificata rispetto ai limiti di plausibilità, aggiornamento e funzionamento definiti dalla configurazione attiva. Il sottosistema distingue quindi una misura valida da una misura degradata, obsoleta, errata o disabilitata e pubblica questa informazione insieme al dato. Le catture del pick-up e gli altri eventi che possono modificare il controllo seguono un percorso a bufferizzazione limitata, con un comportamento esplicito in caso di saturazione, mentre le grandezze a variazione più lenta possono essere rappresentate attraverso l'ultimo valore disponibile.

Le osservazioni convalidate alimentano lo *State Interpretation*; lo stato dei sensori e le condizioni anomale raggiungono invece l'AFSM e la *Telemetry and Diagnostics* attraverso interfacce dedicate. Se un ingresso non è disponibile o risulta implausibile, l'*Environment Observation* non sceglie autonomamente una nuova strategia motore, ma mette a disposizione i dettagli del problema e contrassegna l'eventuale valore sostitutivo come tale. In questo modo la modalità operativa resta una decisione dell'AFSM e la risposta di controllo viene determinata dai sottosistemi che ne possiedono l'autorità, senza nascondere il problema specifico avvenuto all'origine del percorso.

3.3.2 State Interpretation

Le singole misurazioni descrivono aspetti distinti del sistema fisico e possono essere prodotte con frequenze differenti; lo *State Interpretation* le riunisce per costruire una rappresentazione coerente del motore e delle richieste del pilota. A partire dalla sequenza degli eventi del pick-up ricava il regime di rotazione, la posizione angolare dell'albero motore e il livello di sincronizzazione, mentre dagli altri ingressi determina il carico, lo stato termico e le richieste che dovranno essere valutate dalle strategie. Il risultato non è una raccolta dei valori più recenti, ma un'istantanea riferita a un contesto temporale ed a una specifica versione della configurazione in uso nell'ECU.

Il sottosistema conserva sempre la validità, l'istante di acquisizione e l'incertezza delle informazioni di origine, così che un solo campo non affidabile non renda ambiguo l'intero stato. La perdita di un'osservazione opzionale può quindi invalidare soltanto la parte di controllo che ne dipende, lasciando disponibili le restanti grandezze, mentre l'assenza del riferimento dell'albero motore viene segnalata esplicitamente e impedisce di considerare affidabili le informazioni che richiedono la sincronizzazione. Quando necessario, il sottosistema considera anche lo stato recente delle azioni applicate, ma non incorpora la cronologia delle decisioni, che rimane responsabilità del *Control and Decision*.

A quest'ultimo e alla *Telemetry and Diagnostics* viene inviata l'istantanea interpretata; gli eventi di acquisizione o perdita della sincronizzazione e le altre condizioni rilevanti vengono sempre comunicate all'AFSM come dati da valutare per le transizioni. Se le osservazioni necessarie non sono disponibili, il sottosistema costruisce lo stato più completo consentito dai dati, qualificandolo come sconosciuto, non affidabile, degradato o in stallo. Esso descrive quindi ciò che può essere dedotto dal sistema fisico, ma lascia all'AFSM la scelta della modalità e al controllo la selezione del comportamento ammesso in quella modalità.

3.3.3 Control and Decision

Una volta disponibile una rappresentazione coerente del sistema, il *Control and Decision* stabilisce il comportamento da applicare per il ciclo termico corrente. La valutazione combina lo stato interpretato, la configurazione attiva, le richieste del pilota e la modalità operativa fornita dall'AFSM. Da questo contesto derivano le scelte relative all'accensione della scintilla, al *power jet*, alla valvola di scarico, al taglio richiesto dal *quick shifter* e alla selezione delle mappe, senza che il sottosistema debba conoscere i dettagli con cui tali scelte saranno attuate fisicamente.

Ogni ciclo di decisione utilizza un'unica istantanea dello stato, una sola versione della configurazione e un'unica istantanea della modalità operativa, evitando che un aggiornamento intermedio produca una combinazione incoerente. A parità di questi ingressi il risultato deve rimanere deterministico e deve essere ottenuto entro un tempo limitato, poiché il sottosistema fa parte del percorso critico. La decisione generata dal sottosistema indica sia il comportamento desiderato sia il suo stato, distinguendo una richiesta normale da una limitata, inibita o associata a un errore; conserva inoltre la motivazione e il contesto che l'hanno generata, in modo che possano essere verificati prima dell'applicazione e salvati per un'analisi futura delle decisioni.

In assenza anche solo di un dettaglio indispensabile dello stato o della configurazione, il *Control and Decision* non tenta di ricostruirli né trasferisce implicitamente l'errore al sottosistema successivo. Esso, invece, propone un'alternativa conservativa consentita dalla modalità corrente, applicando i limiti previsti, inibendo la funzione interessata oppure richiedendo un comportamento di sicurezza predefinito. Le decisioni e le relative motivazioni vengono inviate all'*Action Coordination* e rese osservabili dalla *Telemetry and Diagnostics*, mentre le condizioni che possono richiedere un cambio di modalità sono inviate direttamente all'AFSM.

3.3.4 Action Coordination

L'*Action Coordination* riceve le decisioni di alto livello e le porta al confine con l'hardware controllato. In questo passaggio una richiesta di accensione, di arricchimento della miscela o di posizionamento della valvola di scarico viene trasformata in un'azione ordinata rispetto alle altre e compatibile con il riferimento temporale disponibile. Il sottosistema separa così le decisioni, che stabiliscono cosa il motore dovrebbe fare, dalla coordinazione fisica, che determina se e quando la richiesta possa essere applicata.

Prima di accettare un'azione vengono confrontati la modalità e la relativa versione, lo stato di sincronizzazione, i vincoli temporali, i parametri attivi dell'attuatore e il contesto trasportato dalla decisione. Le azioni sincrone con l'albero motore sono pianificate soltanto in presenza di un riferimento affidabile, mentre richieste incompatibili vengono risolte secondo regole esplicite di priorità o rifiuto. La temporizzazione dell'accensione ha precedenza sugli aggiornamenti meno critici e un cambiamento verso una modalità più restrittiva provoca l'annullamento delle azioni in attesa che non risultano più ammissibili.

Il sottosistema segue l'azione durante tutto il suo ciclo di vita, distinguendo l'accettazione, la pianificazione, il completamento, il rifiuto e il fallimento quando tali stati sono significativi per l'attuatore. Un rifiuto segnala una

richiesta scartata prima dell'esecuzione, mentre un fallimento descrive un'azione accettata che non è stata applicata o completata correttamente. Questi esiti vengono comunicati all'AFSM e alla *Telemetry and Diagnostics*, ma la conseguente modalità operativa e la decisione del ciclo successivo rimangono rispettivamente responsabilità dell'AFSM e del *Control and Decision*.

3.3.5 Configuration Management

Il percorso di controllo utilizza parametri condivisi da più sottosistemi e richiede che essi descrivano sempre una configurazione completa. Il *Configuration Management* mantiene quindi l'unica copia autorevole della configurazione attiva e la identifica mediante un numero di versione. Mappe, soglie, calibrazioni, profili operativi e abilitazioni delle funzioni raggiungono questo confine come richieste provenienti dall'*External Interaction*, senza poter modificare direttamente i dati usati dal sottosistema di controllo.

Ogni richiesta viene verificata prima dell'accettazione e, quando il cambiamento dipende dalle condizioni del motore, lo *State Interpretation* fornisce il contesto necessario a individuare un punto di attivazione sicuro. Una configurazione accettata diventa visibile in modo atomico all'*Environment Observation*, allo *State Interpretation*, al *Control and Decision* e all'*Action Coordination*; nessuno di essi osserva quindi una combinazione parziale di valori vecchi e nuovi. Il caricamento iniziale e il salvataggio seguono lo stesso principio, mantenendo separata la configurazione valida in uso dalle operazioni di memoria o comunicazione che possono essere più lente.

Se una richiesta non supera la validazione, viene rifiutata e la versione precedente rimane invariata. In assenza di una configurazione persistente valida durante l'avvio, il sottosistema rende disponibile una configurazione predefinita esplicitamente sicura oppure pubblica uno stato non valido che consente all'AFSM di entrare in *FaultLatched*, secondo il profilo operativo scelto. Anche un errore di salvataggio viene esposto alla diagnostica senza rendere incoerente la copia già attiva in memoria, così che un guasto del percorso di supervisione non modifichi implicitamente il comportamento del percorso critico.

3.3.6 Telemetry and Diagnostics

Mentre il controllo procede attraverso i propri sottosistemi, la *Telemetry and Diagnostics* osserva in parallelo ciò che accade nella ECU. Il sottosistema raccoglie le misure e lo stato dei sensori, le istantanee interpretate, le modalità e le transizioni dell'AFSM, le decisioni di controllo, gli esiti delle azioni e lo stato della configurazione. Questi elementi vengono aggiunti in una vista di salute

complessiva e in una cronologia diagnostica capace di spiegare non soltanto quale condizione si sia verificata, ma anche il contesto che ha condotto a un funzionamento limitato, inibito o guasto.

Le informazioni raccolte vengono preparate per l'utilizzo locale, la web application del tecnico, la registrazione e gli eventuali servizi remoti, mentre le interrogazioni esterne sono soddisfatte utilizzando istantanee pubblicate o record posseduti dal sottosistema. Questa funzione rimane tuttavia un osservatore del sistema: l'AFSM riceve le informazioni sullo stato rilevanti direttamente dai loro proprietari e non dipende dalla loro precedente aggregazione diagnostica. Allo stesso modo, i produttori del percorso critico forniscono le informazioni attraverso interfacce limitate e possono continuare a operare anche quando la telemetria non riesce a consumare tutti i dati.

In presenza di una comunicazione lenta o disconnessa, la telemetria può ridurre la frequenza, aggregare, scartare o bufferizzare i dati entro limiti stabiliti, senza bloccare l'osservazione, la supervisione della modalità, il controllo o l'attuazione. Il sottosistema distingue i dati persi a causa della comunicazione dai valori non validi prodotti da una condizione reale dell'ECU, evitando che un problema di trasmissione venga interpretato come un guasto del motore. Se la memorizzazione fallisce, lo stato di salute corrente resta disponibile in memoria e l'errore di registrazione viene a sua volta pubblicato come condizione diagnostica.

3.3.7 External Interaction

L'ultimo confine dell'architettura è rappresentato dall'*External Interaction*, che separa l'ECU dagli utenti, dalle applicazioni, dagli strumenti di servizio e dai servizi remoti. Il sottosistema riceve richieste espresse secondo protocolli esterni e le traduce nei formati interni previsti dalle interfacce proprietarie. Una modifica dei parametri viene quindi inoltrata al *Configuration Management*, mentre una richiesta diagnostica viene affidata alla *Telemetry and Diagnostics*; nessun client esterno ottiene accesso diretto allo stato dei sensori, alle strategie o ai driver degli attuatori.

Nella comunicazione verso l'esterno, l'*External Interaction* fornisce telemetria, stato di salute, risultati diagnostici e risposte di accettazione o rifiuto della configurazione. Le regole di sessione, trasporto e autorizzazione vengono applicate in questo confine, dove le richieste sono considerate più lente e meno affidabili dei dati interni di controllo. Ritardi della rete, aggiornamenti dell'interfaccia o indisponibilità dei servizi remoti non possono pertanto introdurre attese nel percorso critico, e l'eventuale perdita della comunicazione deve avere un effetto definito sulle sessioni e sulle modifiche temporanee ancora in attesa.

Se un canale esterno si interrompe, l'ECU continua a utilizzare l'ultima configurazione interna valida e le richieste incomplete vengono annullate, rifiutate o ripetute secondo la politica dell'interfaccia. Il sottosistema comunica un esito positivo soltanto quando il proprietario interno ha confermato l'accettazione della richiesta, impedendo che la semplice ricezione di un messaggio venga confusa con la sua applicazione. Questa separazione rende sostituibili la web application o il servizio remoto e conclude il percorso dell'informazione senza trasferire all'esterno l'autorità sul comportamento dell'ECU.

3.4 Flussi di comunicazione e sequenze operative

Le responsabilità definite nella sezione precedente stabiliscono quali informazioni produce e conserva ciascun sottosistema. Occorre ora descrivere come queste informazioni vengono comunicate al resto dell'ECU. L'architettura distingue più percorsi, ognuno caratterizzato da specifici vincoli temporali: il percorso di controllo segue il funzionamento del motore e deve completarsi entro tempi limitati e prevedibili, mentre configurazione, telemetria e diagnostica non determinano la temporizzazione degli attuatori.

Ogni informazione viene inviata dal sottosistema che ne è responsabile ai soli destinatari che devono utilizzarla. Questi ultimi non accedono quindi allo stato interno del produttore né ricostruiscono dati appartenenti a un altro sottosistema. Insieme al valore vengono comunicati il riferimento temporale, lo stato di validità e, quando necessario, lo stato di salute, la modalità operativa e la versione della configurazione.

I percorsi differiscono anche nel comportamento previsto in caso di ritardo o indisponibilità. Nel flusso critico, un'informazione obsoleta o non valida determina una limitazione o un'inibizione esplicita; i flussi di supervisione possono invece perdere un aggiornamento senza rallentare il controllo. Le interfacce descritte di seguito definiscono queste proprietà a livello logico, senza imporre uno specifico meccanismo di comunicazione tra task. I diagrammi associati ai percorsi principali ne rappresentano l'ordine temporale e rendono visibili i punti nei quali i sottosistemi scambiano informazioni.

3.4.1 Percorso di controllo

Il percorso di controllo ha origine nell'*Environment Observation*, che acquisisce i segnali del sistema fisico e i comandi immediati del pilota. Il fronte del segnale del *pick-up* e gli altri ingressi legati alla rotazione producono eventi

che devono essere trattati con latenza limitata. La posizione dell'acceleratore e la temperatura dei gas di scarico possono invece essere acquisite con la periodicità prevista per ciascun sensore. Il sottosistema associa a ogni misura l'istante di acquisizione, l'esito della validazione e le informazioni necessarie a stabilirne l'affidabilità. Allo *State Interpretation* viene quindi inviata la misura, mentre lo stato dell'ingresso viene comunicato all'AFSM e alla *Telemetry and Diagnostics*.

Lo *State Interpretation* riunisce le misurazioni in base al loro riferimento temporale e costruisce un'istantanea coerente dello stato del motore e dell'ECU. Il regime di rotazione, la posizione angolare dell'albero motore, il livello di sincronizzazione e le altre grandezze derivate vengono così raccolti in un'unica rappresentazione, associata a una specifica versione della configurazione. Se una misura arriva in ritardo, risulta errata o non è più aggiornata, il sottosistema ne segnala la condizione senza prolungarne implicitamente la validità.

Dallo *State Interpretation* partono due comunicazioni parallele. L'istantanea interpretata viene inviata al *Control and Decision* per il ciclo di controllo successivo, mentre le variazioni della sincronizzazione e le altre condizioni rilevanti vengono comunicate all'AFSM mediante un'interazione *event-driven*. Lo stesso principio vale per gli altri sottosistemi: il *Control and Decision* comunica le condizioni riscontrate durante la decisione e l'*Action Coordination* lo stato delle azioni. L'AFSM riceve quindi le informazioni direttamente dai rispettivi responsabili, senza dipendere dalla loro aggregazione diagnostica.

Quando sulla base delle informazioni ricevute è necessaria una transizione, l'AFSM aggiorna la modalità operativa e ne pubblica un'istantanea contenente la motivazione e il numero di versione. Il *Control and Decision* la combina con lo stato interpretato e la configurazione attiva per determinare il comportamento ammesso nel ciclo corrente. L'*Action Coordination* riceve la decisione insieme al contesto che l'ha generata e verifica nuovamente la modalità e la relativa versione prima di accettare, pianificare o annullare l'azione. La supervisione definisce così i limiti operativi senza sostituirsi alla sequenza di controllo.

L'acquisizione della posizione angolare dell'albero motore e la pianificazione dell'accensione della scintilla devono avvenire entro un tempo limitato e prevedibile. Lo stesso vincolo si applica alla comunicazione di una modalità più restrittiva, poiché le decisioni e le azioni ancora in attesa non possono utilizzare un contesto non più valido. In assenza di un'informazione indispensabile, il percorso non attende la telemetria né un intervento esterno: l'AFSM stabilisce la modalità consentita, il *Control and Decision* propone un comportamento compatibile e l'*Action Coordination* applica gli *interlock* immediati di pro-

pria competenza. Le informazioni meno critiche possono essere aggiornate con frequenze differenti senza introdurre attese nel controllo del motore.

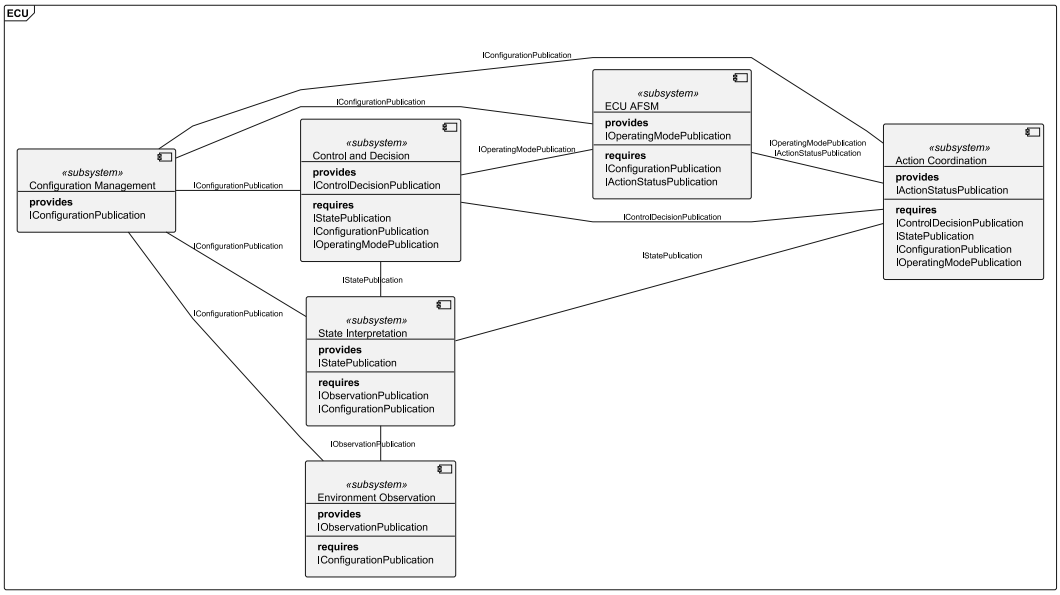

Figura 3.4: Contratti di interfaccia logici del percorso di controllo e relative dipendenze.

La Figura 3.5 riunisce il percorso di controllo e la supervisione dell'AFSM nella sequenza relativa a un ciclo ordinario. Lo stato interpretato, la configurazione e la modalità operativa raggiungono il *Control and Decision* attraverso interfacce distinte e vengono considerati nello stesso contesto. La *Telemetry and Diagnostics* compare soltanto a valle dell'azione coordinata: la registrazione dell'esito non introduce quindi una dipendenza nel percorso critico.

3.4.2 Percorso di configurazione

Il percorso di configurazione ha origine all'interfaccia esterna dell'ECU. Le modifiche provenienti dalla web application, dagli strumenti di servizio o da un'interfaccia locale vengono ricevute dall'*External Interaction*, che verifica la sessione e traduce la richiesta nel formato previsto dall'interfaccia interna. Il sottosistema avvia quindi lo scambio, ma non valida i nuovi parametri: tale responsabilità spetta al *Configuration Management*.

Il *Configuration Management* mantiene la richiesta separata dalla configurazione in uso e controlla la struttura dei dati, gli intervalli ammessi e la coerenza tra parametri, mappe e funzioni abilitate. Quando l'applicazione dei nuovi valori dipende dalle condizioni del motore, lo *State Interpretation* fornisce il contesto

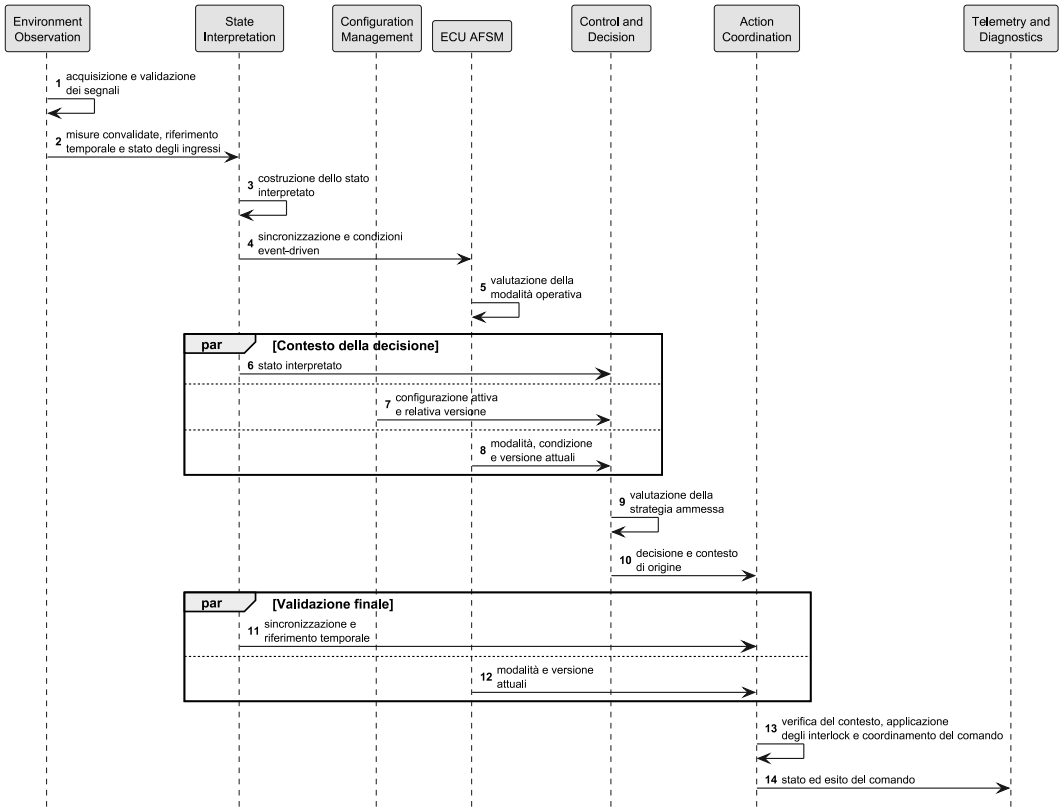

Figura 3.5: Sequenza del ciclo di controllo in condizioni operative normali.

necessario a individuare un punto di attivazione sicuro. Se la richiesta non supera questi controlli, il *Configuration Management* la rifiuta senza modificare la configurazione attiva. L'*External Interaction* comunica quindi l'accettazione al client soltanto dopo aver ricevuto la conferma dal sottosistema responsabile.

In seguito alla validazione, il *Configuration Management* rende attiva la configurazione candidata e le assegna un numero di versione. Il sottosistema ne distribuisce quindi un'istantanea completa all'*Environment Observation*, allo *State Interpretation*, al *Control and Decision* e all'*Action Coordination*. Ciascun destinatario passa dalla versione precedente a quella nuova senza osservare combinazioni intermedie; il ciclo di controllo utilizza così un insieme di valori coerente anche quando l'aggiornamento viene richiesto con il motore in funzione.

Anche all'AFSM viene comunicato lo stato di validità della configurazione, poiché le modalità che abilitano il controllo richiedono una versione attiva e valida. Il salvataggio può precedere o seguire l'attivazione, secondo la politica scelta, ma non deve rendere visibile una configurazione parziale. Un errore di scrittura viene comunicato alla diagnostica e al client senza alterare la copia coerente già presente in memoria.

L'*External Interaction* gestisce ritardi, duplicazioni e interruzioni del canale esterno senza propagarne gli effetti oltre la richiesta in corso. Fino all'attivazione di una nuova versione, i sottosistemi continuano a utilizzare l'ultima configurazione valida; una richiesta incompleta può essere annullata o ripetuta senza modificare il percorso critico. Lo scambio *request-response* consente alla web application e agli strumenti di calibrazione di evolvere indipendentemente dai meccanismi con cui la configurazione viene conservata e distribuita nell'ECU.

La Figura 3.6 evidenzia il confine tra la ricezione della richiesta esterna e la sua accettazione da parte del proprietario della configurazione. L'*acknowledgement* viene restituito soltanto dopo l'attivazione della nuova versione; un rifiuto conclude invece lo scambio senza modificare la configurazione già utilizzata dai sottosistemi.

3.4.3 Telemetria e monitoraggio

Durante il funzionamento, ogni sottosistema comunica alla *Telemetry and Diagnostics* le informazioni necessarie a osservare l'ECU. L'*Environment Observation* invia le misure, la loro validità e lo stato dei sensori; lo *State Interpretation* fornisce le istantanee dello stato ricostruito; il *Control and Decision* rende disponibili le decisioni e il relativo contesto; l'*Action Coordination* comunica lo

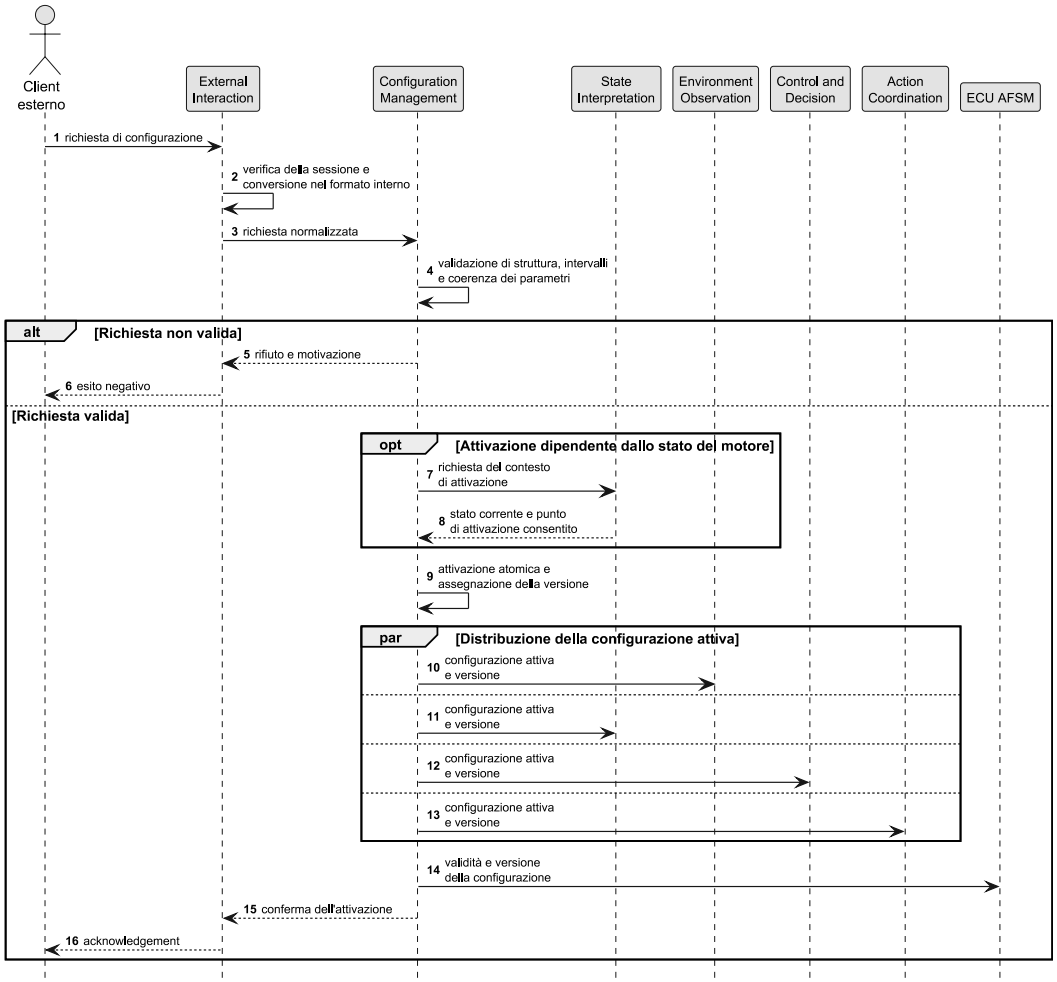

Figura 3.6: Sequenza di validazione e attivazione di una nuova configurazione.

stato e l'esito delle azioni. A queste informazioni si aggiungono la versione della configurazione e le transizioni dell'AFSM, necessarie per ricondurre ogni evento alla condizione operativa in cui si è verificato.

La *Telemetry and Diagnostics* riceve questi dati in parallelo e li organizza in una vista dello stato corrente e in una cronologia diagnostica. Le misure periodiche possono essere campionate o aggregate, mentre le transizioni, i rifiuti e i *fault* vengono conservati come eventi. Il sottosistema mantiene l'ordine temporale e le relazioni tra stato, decisione e azione, evitando di presentare una misura senza la modalità AFSM e la configurazione che ne determinavano il significato.

Le richieste esterne di monitoraggio vengono inoltrate dall'*External Interaction* alla *Telemetry and Diagnostics*. Quest'ultima risponde utilizzando le istantanee e i record che conserva e può inviare alla web application un flusso di dati continuo in *streaming*. Le interrogazioni non accedono direttamente ai task di acquisizione o controllo, e gli aggiornamenti dell'interfaccia utente non condizionano la produzione dei dati successivi.

Le interfacce in ingresso alla telemetria hanno capacità e tempi di servizio limitati. Se i dati vengono prodotti più rapidamente di quanto possano essere elaborati, il sottosistema può ridurre il campionamento, aggregare i valori, sostituire i dati periodici meno recenti, assegnando una priorità maggiore agli eventi diagnostici. Un campione perso durante la trasmissione rimane distinto da una misura non valida all'origine: nel primo caso risulta incompleta l'osservazione esterna, nel secondo cambia la conoscenza che l'ECU possiede del sistema fisico. Anche in assenza della rete, della registrazione o del client, il percorso critico continua a operare senza attendere la telemetria.

3.4.4 Errori e diagnostica

Ogni errore viene rilevato dal sottosistema che dispone delle informazioni necessarie a riconoscerlo. L'*Environment Observation* segnala un ingresso assente, non plausibile o acquisito in condizioni non affidabili; lo *State Interpretation* individua le incoerenze tra le misurazioni e la perdita della sincronizzazione; il *Control and Decision* comunica l'impossibilità di produrre una decisione ammessa; l'*Action Coordination* rileva la violazione di un vincolo temporale o il mancato completamento di un'azione. Il *Configuration Management* rimane responsabile degli errori di validazione e memorizzazione, mentre l'*External Interaction* distingue i problemi di sessione o trasporto dalle condizioni interne della centralina.

Il sottosistema che rileva l'errore comunica la condizione insieme alla sua provenienza, al riferimento temporale e ai dettagli necessari a stabilirne la rilevanza. Se il problema può modificare il comportamento consentito all'ECU, la segnalazione viene inviata direttamente all'AFSM mediante una comunicazione *event-driven* e, in parallelo, alla *Telemetry and Diagnostics*. L'AFSM determina la modalità operativa, mentre la diagnostica conserva l'evento e il contesto in cui si è verificato.

La transizione stabilita dall'AFSM dipende dalla gravità del problema e dalla funzione coinvolta. La perdita di una misurazione opzionale può condurre a *DegradedRunning*, rendendo disponibili soltanto le strategie e i limiti compatibili con le informazioni rimaste. L'assenza di un segnale indispensabile porta a *ControlInhibited*, mentre una condizione critica o non recuperabile conduce a *FaultLatched*. Ogni transizione comprende una nuova versione della modalità operativa e una motivazione esplicita, utilizzate dal *Control and Decision* e dall'*Action Coordination* senza che i due sottosistemi interpretino autonomamente l'errore.

La nuova modalità viene applicata già alla prima decisione o azione successiva. Il *Control and Decision* limita, inibisce o sostituisce la strategia; l'*Action Coordination* verifica nuovamente le richieste pianificate e annulla quelle non più compatibili. Gli *interlock* che devono impedire immediatamente un'azione non sicura rimangono nell'*Action Coordination* e non attendono la registrazione diagnostica. Quest'ultima può quindi essere più lenta della reazione all'errore senza entrare nella catena di protezione.

La *Telemetry and Diagnostics* collega la segnalazione iniziale alla transizione dell'AFSM, alle decisioni limitate o inibite e all'esito delle azioni. L'*External Interaction* rende queste informazioni disponibili al tecnico e agli eventuali servizi remoti, senza posticipare la risposta interna in caso di ritardi nella comunicazione. Per consentire il recupero, l'AFSM richiede nuove misurazioni valide e aggiornate e, quando necessario, una nuova sincronizzazione o una conferma esplicita; la diagnostica registra poi il ritorno alla modalità consentita. In assenza di dati recenti, il sistema rimane nella condizione conservativa già raggiunta.

La Figura 3.7 separa la reazione richiesta dal percorso di controllo dalle successive attività di registrazione e comunicazione. L'AFSM e la *Telemetry and Diagnostics* ricevono la stessa evidenza dal sottosistema che ha rilevato il *fault*; soltanto l'AFSM stabilisce però la nuova modalità operativa, mentre la diagnostica conserva il contesto e lo rende disponibile all'esterno.

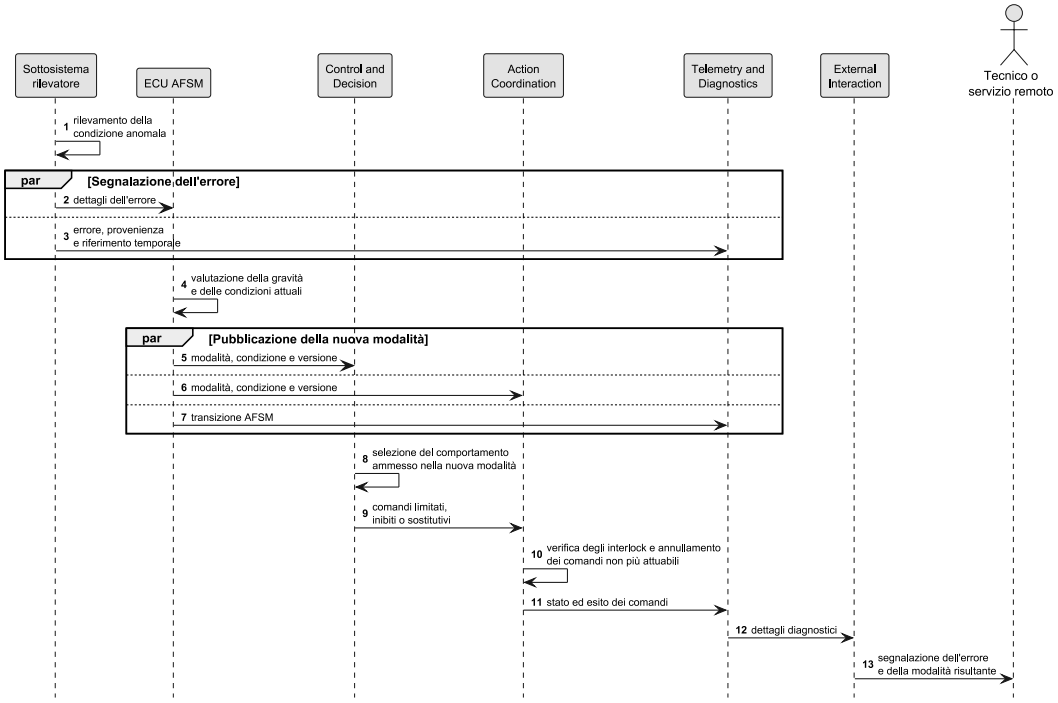

Figura 3.7: Sequenza di gestione di un errore.

3.4.5 Inizializzazione e abilitazione del controllo

L'inizializzazione coinvolge più percorsi e non può essere ricondotta a una singola interfaccia. L'AFSM mantiene il controllo disabilitato mentre il *Configuration Management* rende disponibile una configurazione valida, l'*Environment Observation* comunica la propria *readiness* e lo *State Interpretation* costruisce il primo stato coerente. La *Telemetry and Diagnostics* può essere avviata in parallelo, poiché la sua disponibilità non costituisce una condizione per abilitare le azioni. La sequenza descrive le dipendenze tra le operazioni, ma non assegna all'AFSM l'avvio dei task, che rimane una scelta della progettazione di dettaglio.

La Figura 3.8 rappresenta il percorso che conduce alla modalità *Running*. Il passaggio attraverso *Ready* e *Synchronizing* impedisce di utilizzare le prime misurazioni prima che la configurazione e il riferimento temporale risultino validi. Se manca una delle informazioni sullo stato obbligatorie, l'AFSM rimane nella modalità conservativa applicabile e il *Control and Decision* non genera la prima decisione di controllo.

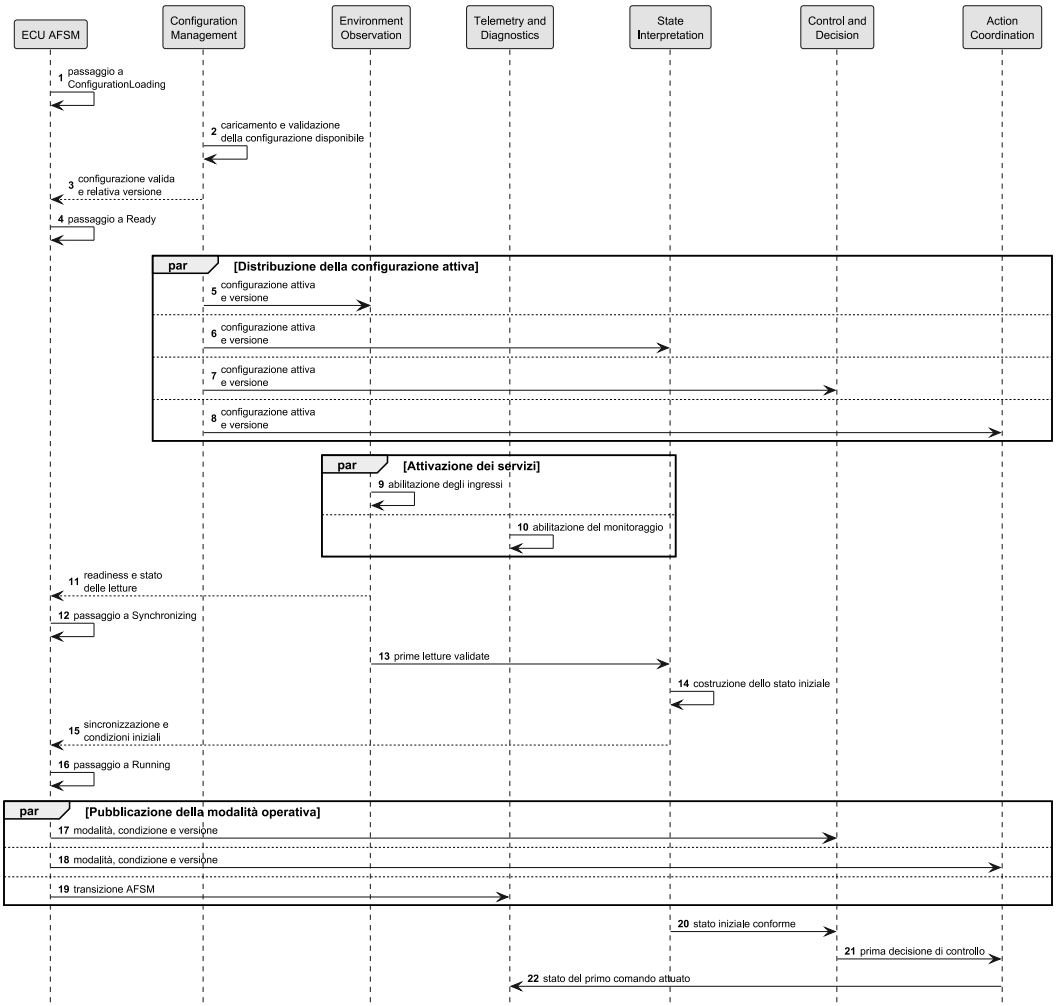

Figura 3.8: Sequenza di inizializzazione dei sottosistemi e abilitazione del controllo.

3.5 Architettura del client utente

Le sezioni precedenti hanno definito quali informazioni raggiungono il confine dell'ECU e quali sottosistemi rimangono responsabili della loro produzione e validazione. A partire da tali interfacce, il client organizza il monitoraggio, la formulazione delle richieste e l'accesso ai servizi remoti. La sua architettura separa lo scambio delle informazioni dalla loro rappresentazione e dalle modalità con cui vengono presentate al tecnico, così che una variazione del collegamento o dell'interfaccia utente non modifichi il significato dei dati ricevuti.

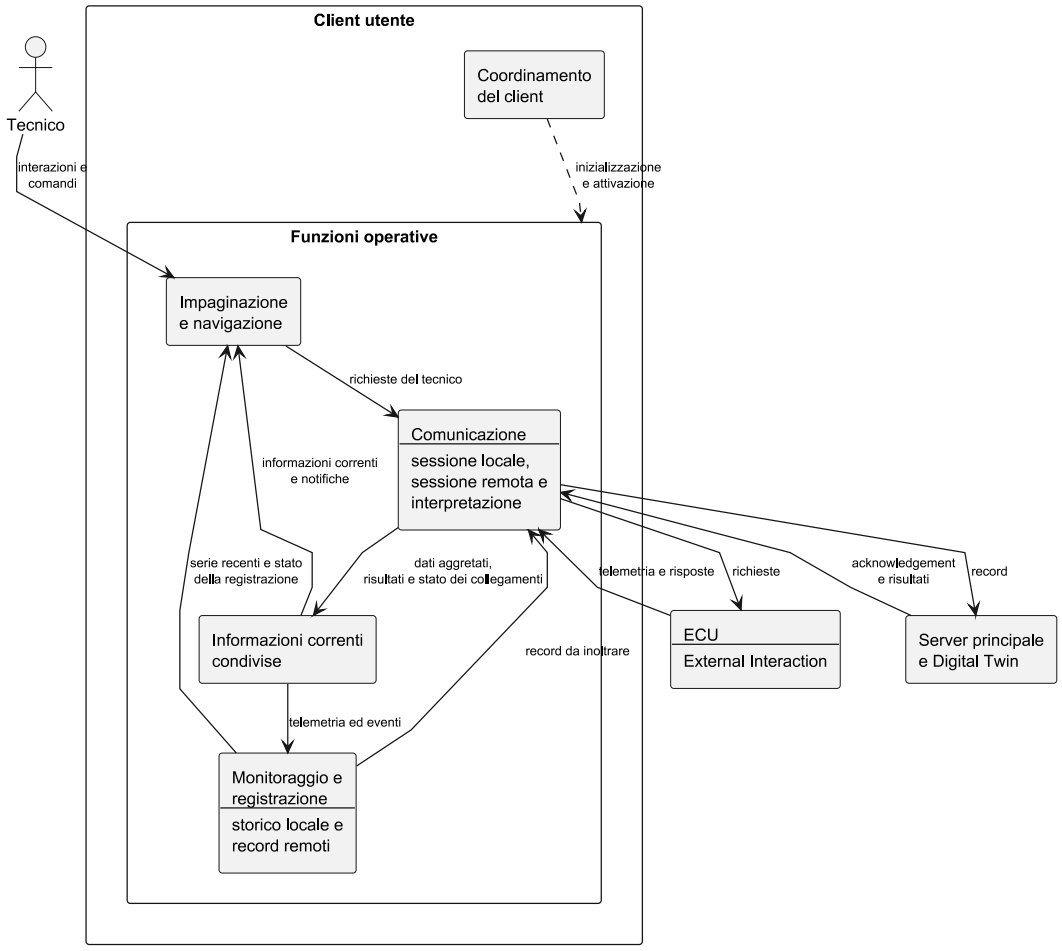

Figura 3.9: Decomposizione logica del client utente e separazione tra il collegamento locale con l'ECU e il percorso opzionale verso il server.

3.5.1 Organizzazione delle responsabilità

Il coordinamento del client predispone le funzioni necessarie al momento dell'avvio e ne regola l'attività durante l'utilizzo. Esso stabilisce quali viste devono essere disponibili, associa a ciascuna le informazioni di cui ha bisogno e avvia i servizi comuni, ma non accentra le decisioni relative alla comunicazione, alla navigazione o alla registrazione. Ciascuna di queste funzioni mantiene pertanto lo stato necessario al proprio funzionamento e lo rende visibile al resto del client soltanto quando deve essere condiviso.

La comunicazione mantiene distinti il collegamento locale con l'ECU e quello opzionale con il server. Il primo trasporta la telemetria, le informazioni che descrivono la centralina e le risposte alle richieste del tecnico; il secondo viene utilizzato per la registrazione persistente e per i risultati prodotti dal Digital Twin. La separazione impedisce che ritardi o errori del percorso remoto interferiscano con lo scambio locale e consente alle funzioni di presentazione di utilizzare le informazioni senza dipendere dal modo in cui sono state trasportate.

Una pagina dell'interfaccia riceve soltanto le informazioni necessarie alla funzione che espone e viene aggiornata mentre risulta attiva, evitando che le parti non visibili continuino a elaborare ogni variazione della telemetria. Lo stesso confine separa la visualizzazione dai comandi: la pagina raccoglie l'azione dell'utente e la consegna alla funzione responsabile della comunicazione, senza modificare direttamente lo stato ricevuto dall'ECU.

Al monitoraggio corrente si affianca uno storico locale di informazioni recenti, utilizzato per osservare l'andamento recente delle grandezze e collocare gli eventi nel relativo contesto temporale. I record destinati a una conservazione più lunga seguono invece il percorso verso il server e non vengono ricavati dai valori già preparati per la visualizzazione. In questo modo la registrazione mantiene l'unità e il contesto delle informazioni prodotte dall'ECU, mentre la presentazione può applicare formati e livelli di dettaglio adeguati alla vista senza alterare il dato di origine.

3.5.2 Gestione dei flussi informativi

Il collegamento del client all'ECU inizializza una sessione locale. Le informazioni che descrivono l'identità della centralina, le caratteristiche del flusso di dati disponibile e le impostazioni della registrazione vengono acquisite quando sono rese disponibili e completano progressivamente la rappresentazione condivisa. Le pagine della webapp possono così distinguere un'informazione non

ancora disponibile da una misura non valida, senza attribuire ai primi dati di telemetria un significato che richiede un contesto ancora assente.

Durante il funzionamento, ogni aggiornamento della telemetria viene trattato come un insieme coerente di valori, metadati ed eventi riferiti allo stesso istante e alla stessa versione. La funzione di comunicazione ne interpreta la struttura e aggiorna la rappresentazione condivisa, dalla quale le pagine attive ricavano il contenuto da mostrare. In parallelo, il monitoraggio aggiunge allo storico un solo campione per ciascun aggiornamento e conserva gli eventi associati, evitando di costruire una sequenza composta da valori appartenenti a istanti differenti.

Un comando del client segue il percorso opposto. L'azione formulata dal tecnico viene processata dalla webapp, trasformata in una richiesta compatibile con l'interfaccia dell'ECU e inviata attraverso la sessione locale. Il client distingue l'invio dall'applicazione effettiva: la semplice trasmissione non modifica la rappresentazione dello stato autorevole e l'esito viene mostrato soltanto sulla base delle informazioni restituite dalla centralina. Rimangono così separate la preparazione della richiesta, svolta dal client, e la validazione dei parametri o delle condizioni operative, che appartiene ai sottosistemi interni dell'ECU.

Quando viene richiesta una registrazione sul server remoto, i record completi della telemetria vengono affiancati al flusso locale e inoltrati al server secondo il loro ordine. Il client considera acquisito un record soltanto dopo la conferma del servizio remoto e conserva entro un limite definito quelli ancora in attesa. La registrazione utilizza quindi la stessa informazione ricevuta dalla ECU, ma procede attraverso un percorso indipendente da quello che aggiorna le viste e lo storico recente.

3.5.3 Continuità del servizio e condizioni di errore

La perdita del collegamento locale interrompe l'arrivo di nuove informazioni e impedisce l'invio di ulteriori richieste. I valori già presenti possono rimanere visibili come ultima osservazione disponibile. Una nuova connessione ricostruisce la sessione e acquisisce nuovamente le informazioni disponibili, senza ripetere in modo implicito i comandi formulati prima dell'interruzione.

L'indisponibilità del server ha un effetto più limitato, poiché riguarda soltanto la registrazione persistente e la consultazione dei risultati remoti. Il monitoraggio locale continua a utilizzare la telemetria proveniente dalla ECU e lo storico recente rimane disponibile finché il client resta attivo. I record non ancora confermati dal server possono essere trattenuti soltanto entro la capacità

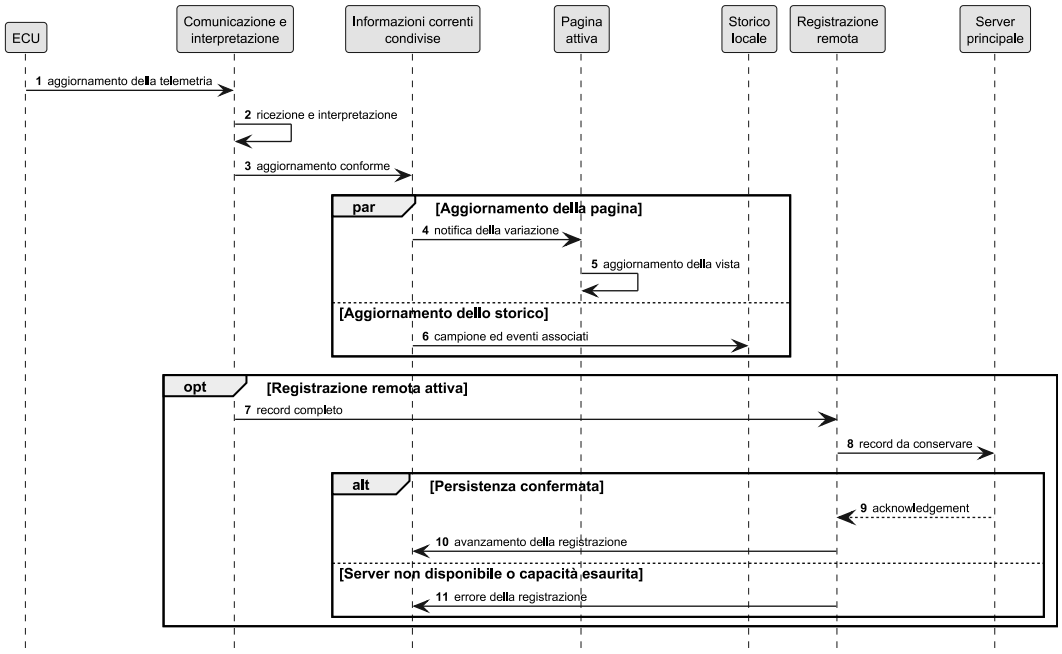

Figura 3.10: Aggiornamento della telemetria nel client e inoltro opzionale del record al server.

assegnata; quando tale limite viene raggiunto, la registrazione viene arrestata e il tecnico viene avvisato.

Un errore del trasporto rimane distinto da un valore dichiarato non valido dall'ECU. Nel primo caso il client non ha ricevuto o conservato una parte del flusso, mentre nel secondo presenta correttamente una condizione rilevata alla fonte. Conservare questa distinzione permette di attribuire il problema al confine nel quale si è verificato e impedisce che una perdita di comunicazione venga interpretata come un nuovo stato del motore. In entrambe le condizioni, il client può degradare il monitoraggio o interrompere una registrazione, ma non modifica il percorso di controllo né la configurazione valida mantenuta dalla centralina.

3.6 Architettura del Digital Twin

Il client inoltra al server remoto le informazioni destinate alla conservazione; il server le associa all'ECU che le ha prodotte e aggiorna il Digital Twin corrispondente soltanto quando comprendono il contesto necessario a interpretarle. Il client gestisce la comunicazione e la presentazione dei risultati mentre il

Digital Twin conserva la cronologia e i risultati delle elaborazioni server-side, senza partecipare al percorso di controllo né inviare comandi all'ECU.

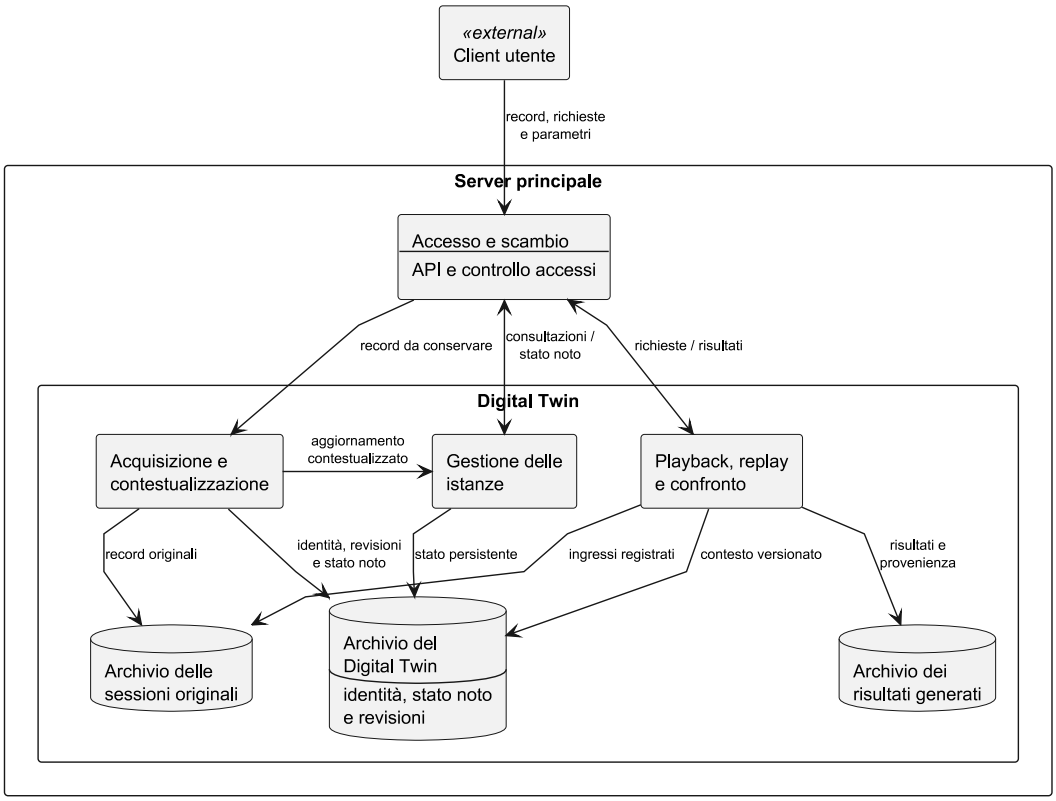

Figura 3.11: Decomposizione logica del server e separazione tra gestione del Digital Twin, sessioni originali e risultati generati.

La Figura 3.11 distingue il confine di accesso, la contestualizzazione dei record, la gestione delle istanze e le elaborazioni eseguite sui dati registrati. Gli archivi mantengono separati lo stato persistente del Digital Twin, le sessioni originali e i risultati generati sul server, senza introdurre un percorso di comando verso l'ECU.

3.6.1 Stato e provenienza delle informazioni

Il Digital Twin conserva l'origine delle singole informazioni. Le misurazioni acquisite dai sensori rimangono distinte dalle grandezze ricavate, dalle decisioni generate dalla logica di controllo e dai comandi inviati agli attuatori. Una stima ottenuta da un modello o un calcolo eseguito con parametri alternativi viene invece registrata come risultato server-side, non come valore osservato

sull'ECU fisica. Il tecnico può quindi distinguere ciò che è avvenuto durante il funzionamento dai risultati generati in seguito dal server.

Ogni sessione registrata identifica l'ECU di origine, gli istanti di osservazione e le revisioni del firmware, delle mappe, della configurazione e delle calibrazioni utilizzate durante l'acquisizione. I risultati generati dal server indicano inoltre la procedura di *replay* o di simulazione che li ha prodotti e, quando è stato utilizzato un modello, la relativa versione. Il Digital Twin non modifica i record originali. Ogni elaborazione successiva produce un nuovo risultato collegato alla sessione di origine. In questo modo, il tecnico può ripetere l'analisi senza perdere il riferimento ai dati acquisiti.

3.6.2 Ricostruzione, replay e confronto

Il tecnico può consultare una sessione registrata mediante il *playback*, eseguire un *replay* deterministico oppure confrontare il comportamento dell'ECU con quello ottenuto mediante parametri alternativi. Il *playback* presenta i dati ricevuti dall'ECU secondo il loro ordine temporale, senza ricalcolare il comportamento della centralina. Nel *replay* deterministico, il server riapplica agli ingressi registrati la stessa versione della logica di controllo e verifica se, a parità di ingressi e revisioni, ottiene le decisioni registrate. Per valutare una configurazione alternativa, il server elabora invece gli stessi ingressi con mappe o parametri diversi. Il risultato rimane separato dai dati originali e dal risultato del *replay*.

La Figura 3.12 separa i tre percorsi offerti dal server. Il *playback* restituisce i record originali, mentre il *replay* deterministico e il confronto con parametri alternativi eseguono una nuova elaborazione e ne conservano il risultato con la relativa provenienza. In tutti i casi, la sessione acquisita dall'ECU rimane invariata.

A partire dai dati registrati e dai risultati generati, il tecnico può ricostruire l'andamento di una sessione e analizzarne i dati per ciascuna rotazione dell'albero motore. Può inoltre verificare come una misurazione o una correzione abbia contribuito a una decisione, esaminare il contesto che precede un *fault* e individuare le regioni operative nelle quali una mappa alternativa avrebbe prodotto un comando differente.

3.6.3 Limiti del modello

Nella sua configurazione iniziale, il Digital Twin descrive l'ECU e ne ricostruisce il comportamento a partire dalle informazioni disponibili, ma non costituisce un modello completo del processo fisico. Un comando calcolato con

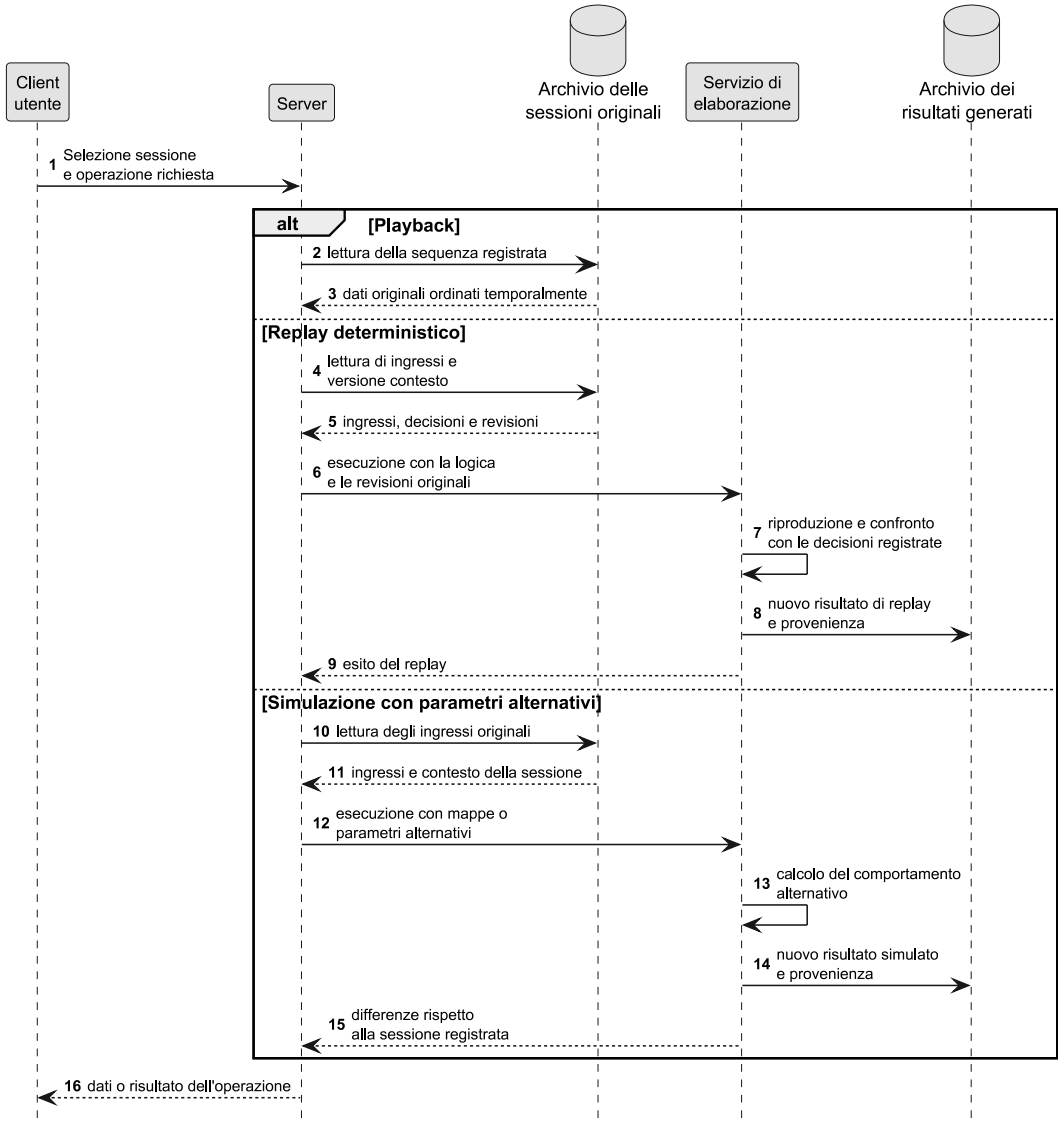

Figura 3.12: Sequenza di playback, replay deterministico e simulazione con parametri alternativi.

una mappa alternativa rimane pertanto un risultato server-side utile al confronto tra due strategie di controllo e non descrive un effetto fisico osservato sul motore. Il Digital Twin potrà includere modelli termici, stime del rischio di detonazione, analisi del degrado degli attuatori e funzioni di diagnostica predittiva solo dopo che la loro affidabilità sarà stata verificata su dati fisici adeguati. Anche in quel caso, manterrà i risultati distinti dalle osservazioni prodotte dall'ECU.

Capitolo 4

Sviluppo del prototipo

4.1 Contesto dell'implementazione

4.1.1 Sistema implementato

Il progetto è stato realizzato fisicamente sotto forma di prototipo dell'ECU, costruito attorno a un PCB progettato appositamente. La scheda riunisce i circuiti necessari ad acquisire i segnali del motore e a comandarne gli attuatori, mentre sul SoC ESP32-S3 è stato implementato il firmware che realizza i sottosistemi e l'AFSM descritti nel capitolo precedente. La descrizione presentata in questo capitolo si riferisce a una precisa configurazione del sistema, costituita dalla PCB `v1.0` e dal firmware `v1.1`. Il riferimento congiunto a queste due revisioni evita di attribuire al prototipo caratteristiche appartenenti a fasi differenti dello sviluppo e costituisce il punto di partenza per interpretare le prove discusse nelle sezioni successive.

Per questo prototipo si è scelto un modulo Wemos LOLIN ESP32-S3 Mini basato sul SoC ESP32-S3FH4R2 [12]. Il modulo è collegato alla scheda mediante pin header e può quindi essere sostituito facilmente. Il firmware è sviluppato con ESP-IDF `v5.5.4` [4] e utilizza l'implementazione dual-core di FreeRTOS fornita dal framework [5].

4.1.2 Mappatura dell'architettura logica

Il capitolo precedente ha suddiviso l'ECU in sette sottosistemi e in un'AFSM, assegnando a ciascuno una responsabilità precisa e le informazioni che può produrre o modificare, senza vincolare questa organizzazione a una specifica tecnologia. L'implementazione conserva tali confini, ma non li traduce in una

corrispondenza diretta tra sottosistemi e task FreeRTOS. Le unità di esecuzione sono state separate in base ai vincoli temporali, alla possibilità di eseguire operazioni bloccanti e alla necessità di gestire gli effetti di un eventuale errore.

Il Core 1 ospita le attività rivolte al motore, dall'acquisizione del pick-up alla pianificazione delle uscite, mentre il Core 0 esegue le funzioni di rete, memorizzazione, diagnostica e telemetria. Nel percorso principale, *State Interpretation*, *Control and Decision* e la parte temporalmente critica di *Action Coordination* formano una sequenza interna al task `engine_control`. Questa scelta consente alle tre fasi di utilizzare lo stesso stato interpretato, la stessa configurazione e la stessa modalità AFSM senza introdurre uno scambio attraverso una coda tra una fase e la successiva.

La Figura 4.1 riassume questa mappatura e rende visibile la separazione tra il percorso rivolto al motore e i servizi di supporto.

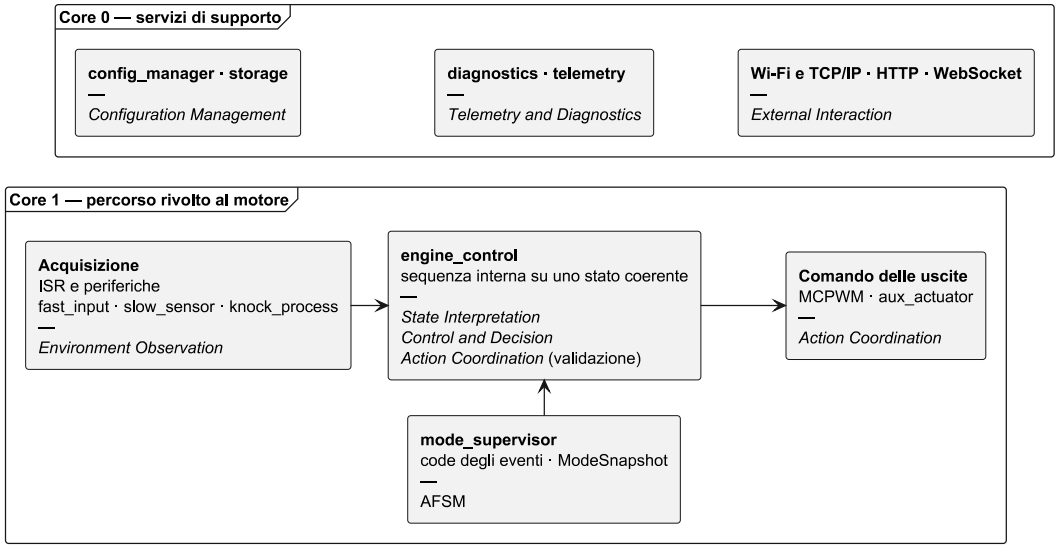

Figura 4.1: Mappatura delle responsabilità logiche sulle unità di esecuzione del prototipo.

4.1.3 Vincoli di implementazione

La prima revisione della scheda è stata guidata soprattutto dal contenimento dei costi e dalla rapidità di prototipazione. Per questo motivo la PCB comprende gli ingressi e le uscite necessari a validare il percorso di controllo principale, mentre rinvia a una revisione successiva i circuiti dedicati al knock sensor e alle misure della temperatura dei gas di scarico e dell'acqua. Ciascuna di queste funzioni richiede un circuito integrato specifico e avrebbe aumentato

il costo della scheda, il numero di componenti da approvvigionare e il tempo necessario per completare il primo prototipo.

La scelta del modulo Wemos non deriva dalla stessa semplificazione economica, ma dalla necessità di rendere sostituibile l'unità di elaborazione. La centralina, alimentata con una tensione nominale di 12 V, integra nella sezione della CDI tensioni continue comprese indicativamente tra 200 V e 400 V. Il collegamento mediante pin header permette di rimuovere l'ESP32-S3 qualora un guasto nella sezione ad alta tensione danneggi il modulo, senza dover sostituire l'intera PCB.

Ai vincoli hardware si affianca la necessità di mantenere prevedibile il tempo che separa l'acquisizione del riferimento angolare dal comando della CDI. Il runtime assegna quindi al Core 1 il percorso rivolto al motore e impedisce che questo attenda operazioni di rete, memorizzazione, diagnostica o telemetria. Le attività prive di una scadenza legata alla rotazione sono eseguite sul Core 0 e comunicano con il dominio del motore mediante strutture a capacità limitata e istantanee coerenti. I limiti temporali, le priorità e l'assegnazione delle periferiche saranno precisati nella sezione dedicata all'organizzazione del firmware.

4.2 Fondamenti dell'implementazione firmware

4.2.1 Ambiente di sviluppo e riproducibilità della build

Il firmware di riferimento è sviluppato in C++23 e ha come build target `esp32s3`. La frequenza della CPU è stata fissata a 240 MHz durante l'esecuzione dei servizi dell'ECU, mentre il tick di FreeRTOS è impostato a 1 kHz. Il dynamic frequency scaling e il light sleep automatico sono disabilitati, poiché entrambi introdurrebbero variazioni non necessarie nel tempo di risposta del percorso rivolto al motore.

La configurazione del progetto estende quella standard di ESP-IDF soltanto nei punti che modificano il comportamento dell'ECU. Essa definisce l'esecuzione dual-core e le affinità dei task, abilita i controlli di integrità di FreeRTOS e il Task Watchdog, rende cache-safe le operazioni MCPWM utilizzate dal percorso critico e colloca i servizi Wi-Fi e TCP/IP sul Core 0. Comprende inoltre il supporto alla configurazione Wi-Fi, all'aggiornamento OTA, alla persistenza NVS e al filesystem SPIFFS che contiene le risorse della WebUI. In questo modo `sdkconfig` non rappresenta soltanto un dettaglio della build, ma una parte

delle condizioni necessarie per riprodurre l'organizzazione temporale descritta nel seguito.

La partition table è organizzata attorno all'aggiornamento remoto e alla distribuzione dell'interfaccia utente, come riassunto nella Tabella 4.1. Le due immagini applicative `ota_0` e `ota_1` dispongono di spazio sufficiente a far sì che una nuova versione possa essere trasferita senza sovrascrivere quella in esecuzione. Lo spazio rimanente forma la partizione SPIFFS `www`, dalla quale vengono serviti i file della WebUI.

Partizione	**Dimensione**	**Descrizione**
`nvs`	`0x4000` byte	Memoria non volatile
`otadata`	`0x2000` byte	Dati per aggiornamenti OTA
`phy_init`	`0x1000` byte	Dati di inizializzazione PHY
`ota_0`	`0x177000` byte	Immagine applicativa principale
`ota_1`	`0x177000` byte	Immagine applicativa secondaria
`www`	`0xF9000` byte	Filesystem SPIFFS (WebUI)

Tabella 4.1: Suddivisione della memoria flash tramite partition table.

4.2.2 Organizzazione delle classi e delle interfacce

L'organizzazione del firmware conserva le responsabilità definite nel capitolo precedente senza trasformare i task in sostituti delle classi di dominio. Nel percorso principale, `StateInterpreter` costruisce uno stato coerente a partire dalle osservazioni disponibili, `ControlDecision` applica la strategia ammessa dalla modalità corrente e `ActionCoordinator` valida e prepara le azioni fisiche. Le tre classi vengono invocate in successione da `EngineController`, che mantiene coerenti la generazione dello stato, della configurazione e della modalità per l'intera elaborazione. `EngineControlTask` adatta questa sequenza al risveglio e alle primitive di FreeRTOS, ma non contiene le regole che determinano la decisione.

Le osservazioni raggiungono il controllo attraverso interfacce di lettura che espongono snapshot immutabili. Le classi che acquisiscono gli ingressi rapidi, i sensori lenti e la misura di knock mantengono il possesso dei rispettivi dati e rendono disponibile una nuova versione soltanto dopo aver completato validazione e aggiornamento. `EngineController` non accede direttamente ai driver, così la provenienza del dato e il suo contesto di acquisizione non modificano la logica che lo utilizza. Sul lato delle uscite, `ActionCoordinator` usa interfacce distinte per la CDI e per gli attuatori ausiliari: la prima resta sincronizzata

con il pick-up nel task `engine_control`, mentre power jet e valvola di scarico vengono applicati dal task `aux_actuator`.

La modalità operativa è dettata da `AfsmStateMachine`, che valuta eventi e condizioni senza dipendere da driver, rete o storage. `ModeSupervisor` stabilisce la precedenza tra gli eventi, applica l'ordinamento necessario durante le transizioni e rende disponibile la nuova `ModeSnapshot`; `ModeSupervisorTask` fornisce soltanto l'adattamento al runtime. Le interfacce `IAfsmEventSink` e `IAfsmEventSource` separano producer e consumer degli eventi, `ISystemEvidence` consente di leggere le informazioni sullo stato possedute dagli altri sottosistemi e `IModeSnapshotPublisher` rende disponibile il risultato della transizione. L'interfaccia `IControlInterlock`, invece, permette all'AFSM di inibire immediatamente il controllo senza attribuirle il comando diretto degli attuatori.

La stessa separazione viene mantenuta per le funzioni sul Core 0. La classe che gestisce la configurazione è l'unica responsabile della configurazione attiva e usa un'interfaccia di salvataggio invece di accedere direttamente alla NVS. Le classi di diagnostica e telemetria leggono snapshot o ricevono eventi senza diventare proprietarie dello stato osservato. Gli handler HTTP e WebSocket, infine, traducono le richieste esterne in operazioni tipizzate e le inoltrano all'interfaccia appropriata; non leggono i sensori né invocano direttamente il controllo o i driver degli attuatori.

4.2.3 Criteri di implementazione orientata agli oggetti

Le classi appena introdotte sono implementate in C++, mentre il C rimane limitato ai punti di ingresso richiesti da ESP-IDF, come `app_main`, gli interrupt e le callback delle periferiche. Queste funzioni costituiscono adapter sottili: acquisiscono il dato fornito dalla piattaforma e lo trasferiscono a un'interfaccia tipizzata, senza eseguire filtraggio, controllo o transizioni dell'AFSM. La logica di dominio può così essere testata su host senza avviare FreeRTOS o riprodurre le periferiche del microcontrollore.

Ogni classe mantiene privato il proprio stato modificabile e ogni risorsa ha un solo proprietario autorevole. Le dipendenze vengono fornite esplicitamente alla costruzione e attraversano interfacce strutturate, anziché essere recuperate da variabili globali modificabili. Gli eventi e le modalità utilizzano enumerazioni fortemente tipizzate, mentre le informazioni di dominio sono raccolte in *value object* specifici. Un evento dell'AFSM, per esempio, conserva tipo, producer, timestamp e generazione della sorgente, evitando payload generici il cui significato dipenderebbe da convenzioni esterne.

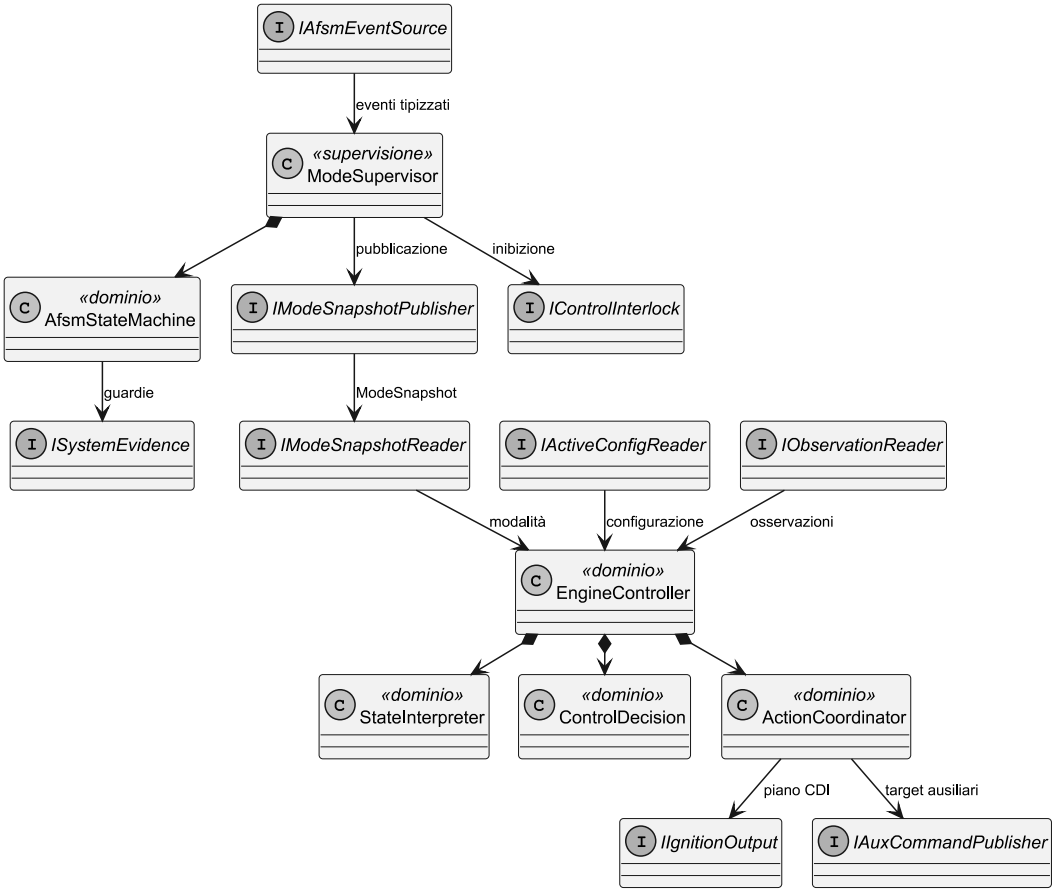

Figura 4.2: Classi e interfacce principali del percorso di controllo e della AFSM.

RTTI è disabilitato e il design non dipende dall'identificazione dinamica dei tipi. I template vengono impiegati quando permettono di esprimere a compile time una proprietà comune, come il tipo e la capacità di una coda o il payload di una snapshot, senza introdurre controlli a runtime. Il codice applicativo non esegue allocazioni dinamiche dopo lo startup. Stack, code, snapshot e buffer necessari al controllo vengono predisposti prima che le uscite possano essere abilitate, mentre i wrapper dichiarano proprietari, lifetime e contesti dai quali possono essere usati. I servizi Wi-Fi e HTTP interni a ESP-IDF possono continuare a gestire le proprie risorse sul Core 0, ma non condividono con il percorso rivolto al motore lock o strutture la cui disponibilità dipenda da un'allocazione.

4.2.4 Modello di esecuzione e concorrenza

I task applicativi sono creati in memoria statica e con affinità esplicita. Il Core 1 ospita il dominio rivolto al motore: `engine_control` ha priorità 22, `mode_supervisor` priorità 21 e `fast_input` priorità 18. A questi seguono `aux_actuator`, `knock_process` e `slow_sensor`, rispettivamente con priorità 16, 14 e 10. L'ordine rispecchia la gravità delle conseguenze di un eventuale ritardo: l'acquisizione del pick-up, la supervisione della modalità e la decisione di controllo devono precedere l'elaborazione di sensori opzionali o più lenti.

Il Core 0, invece, esegue i servizi che possono richiedere più memoria o incorrere in attese non compatibili con la rotazione del motore. `config_manager` usa priorità 10, `diagnostics` priorità 8, l'HTTP server priorità 6, `telemetry` priorità 5 e `storage` priorità 4. Anche il driver Wi-Fi e il task TCP/IP sono vincolati a questo core. L'affinità non viene però considerata una barriera sufficiente: una scrittura in flash può interferire con entrambi i core, perciò NVS e OTA sono sottoposti anche a vincoli sulla modalità AFSM e sullo stato del motore.

Gli interrupt acquisiscono soltanto eventi hardware. La callback MCPWM del pick-up salva il valore del capture timer in un record già allocato, prova a inserirlo nella coda e notifica `engine_control`; non calcola RPM, angolo o anticipo, né esegue logging. Lo stesso criterio è applicato agli ingressi digitali e alla conclusione della finestra di knock. Il lavoro che richiede validazione, filtraggio o accesso a un bus viene rinviato al task che possiede la relativa risorsa.

Nel percorso del motore non sono ammessi mutex, allocazioni dinamiche, attese per spazio in coda, comunicazioni via rete, salvataggio in NVS o logging. Le sezioni critiche necessarie a condividere registri con un ISR devono restare

limitate e non possono trasformarsi in un meccanismo generale di sincronizzazione. Per le periferiche, la mutua esclusione deriva, invece, dall'ownership: un solo task configura e utilizza ciascun driver, eliminando la contesa della risorsa dal normale percorso applicativo.

Questa distribuzione concretizza la separazione già sintetizzata nella Figura 4.1. Le priorità determinano l'ordine di preemption all'interno di ciascun core, mentre i meccanismi impiegati per lo scambio cross-core sono descritti nella sottosezione seguente.

4.2.5 Comunicazione e proprietà dei dati

Il meccanismo di comunicazione dipende dal significato del dato. Gli eventi che devono essere elaborati in ordine e i comandi che non possono essere sostituiti dall'ultimo valore disponibile attraversano code FreeRTOS a capacità limitata. Le direct task notification vengono usate soltanto per risvegliare un task e non trasportano informazioni di dominio: dopo il risveglio, il consumer legge sempre la propria coda o la snapshot resa disponibile dal producer. Questa regola evita che lo stesso dato assuma significati differenti in base al bit che ha causato l'attivazione.

Gli stati correnti vengono comunicati mediante snapshot immutabili e versionate, collocate nella memoria interna. Ogni snapshot ha un solo writer: `fast_input` invia gli ingressi rapidi, `slow_sensor` e `knock_process` rendono disponibili le rispettive osservazioni, `engine_control` lo stato interpretato e la decisione, `mode_supervisor` la modalità e `config_manager` la configurazione attiva. Durante l'aggiornamento un contatore atomico assume prima un valore dispari e poi il successivo valore pari; il reader accetta la copia soltanto se il contatore è pari e non cambia tra l'inizio e la fine della lettura. Dopo tre tentativi falliti il dato è considerato non disponibile per il ciclo termico corrente, senza prolungare indefinitamente il percorso critico.

Ogni informazione usata dal controllo porta con sé il timestamp, la propria validità e la versione del producer. Le decisioni e le azioni conservano anche la versione della configurazione e della modalità AFSM. Prima di preparare un nuovo piano, `engine_control` legge la modalità, acquisisce la configurazione e le osservazioni, quindi legge nuovamente la modalità. Se le due generazioni non coincidono, ripete una sola volta l'acquisizione; un secondo cambiamento rende il ciclo termico corrente inutilizzabile e il comando per la CDI viene omesso.

La perdita di un segnale del pick-up rende non affidabile la sequenza angolare, attiva l'interlock e richiede una nuova sincronizzazione; la saturazione del ca-

nale AFSM critico conduce a *FaultLatched*, perché potrebbe essere stato perso un evento che modifica i permessi del controllo. In caso di eccesso di dati di telemetria, invece, il sottosistema sostituisce il frame di stato non ancora inviato con quello più recente. In questo modo il sistema non tratta l'overload come un errore indistinto, ma lo gestisce in base alle conseguenze che la perdita ha sull'informazione trasportata.

4.2.6 Modello temporale

Tutte le unità di esecuzione utilizzano un tempo monotono espresso come contatore unsigned a 64 bit di microsecondi. I task lo ottengono tramite `esp_timer_get_time()`, mentre i valori catturati dalle periferiche vengono convertiti nello stesso dominio prima di essere pubblicati. Il tick di FreeRTOS serve esclusivamente a rilasciare attività con periodo nell'ordine dei millisecondi; non viene usato per determinare la posizione dell'albero motore o i timestamp dei sensori. I task periodici impiegano `vTaskDelayUntil()`, in modo che il tempo impiegato da una singola esecuzione non si accumuli trasformandosi in un ritardo di fase nei periodi successivi.

Il vincolo più stringente deriva dalla distanza tra il pick-up e il punto nel quale deve avvenire l'accensione. Con un impulso per giro, a 25 000 RPM due segnali consecutivi distano 2.4 ms; a 20 000 RPM, un pick-up collocato a 35 gradi prima del punto morto superiore lascia circa 292 µs prima del raggiungimento di tale posizione. Calcolare dopo il fronte l'intera decisione non offrirebbe quindi un margine sufficiente. Il firmware prepara durante il giro corrente un `IgnitionPlan` destinato al giro successivo, associandolo alle versioni dello stato, della configurazione e della modalità.

Quando arriva un nuovo fronte, la callback MCPWM risveglia `engine_control`. Il task valida il piano già calcolato, controlla gli interlock e programma un impulso one-shot attraverso la periferica MCPWM. Questo percorso rapido dispone di un budget di 75 µs; la risoluzione del timer di uscita è 2 µs e un comando che risulti in ritardo di oltre 100 µs non viene eseguito. Soltanto dopo avere programmato o rifiutato l'azione corrente, il task completa la validazione del pick-up, aggiorna lo stato del motore, valuta il controllo e prepara il piano successivo. L'intera fase deve rimanere entro 500 µs, lasciando parte dell'intervallo minimo tra due fronti agli interrupt e alle attività ESP-IDF con priorità superiore.

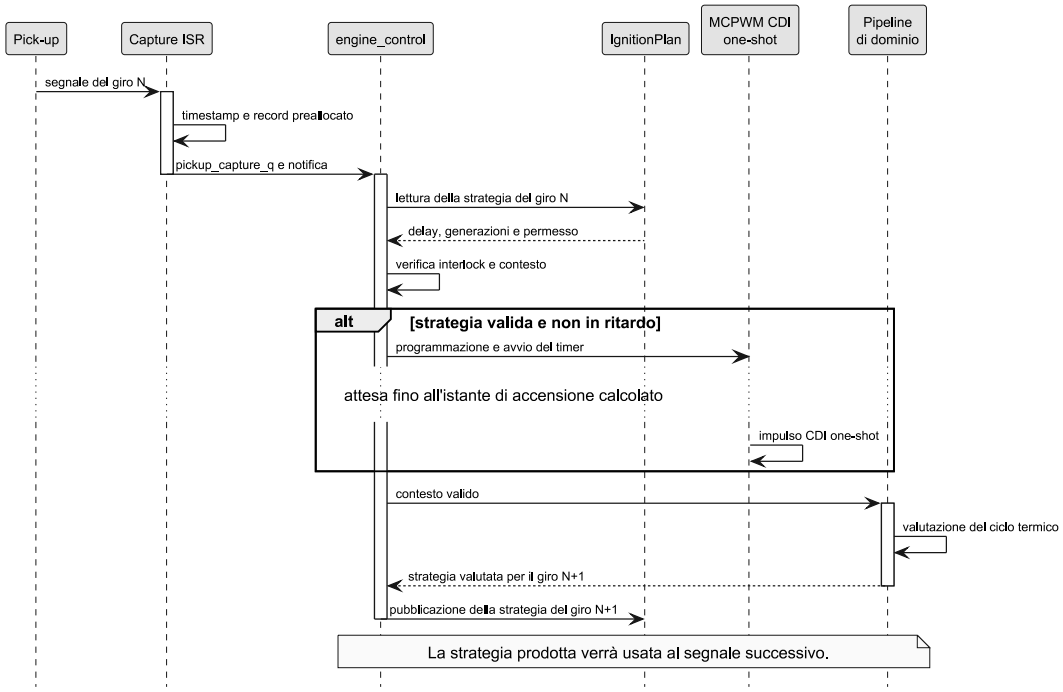

Figura 4.3: Sequenza temporale dalla cattura del pick-up alla preparazione della strategia di accensione.

4.2.7 Gestione della memoria e delle risorse

Le risorse necessarie al controllo vengono allocate prima che l'ECU possa abilitare le uscite e non cambiano dimensione durante il funzionamento. Rientrano in questa politica gli stack e i TCB dei dieci task applicativi, le code, i wrapper delle snapshot, i record utilizzati dagli ISR, i due slot per le configurazioni candidate, i piani di accensione e il ring diagnostico. Le risorse interessate dal percorso critico risiedono nella DRAM interna; il codice e le costanti usati dalle callback MCPWM vengono collocati in IRAM o DRAM quando richiesto dalla configurazione cache-safe.

Dopo il superamento della barriera di startup, nessun task applicativo può eseguire allocazioni dinamiche. I servizi di rete interni a ESP-IDF rimangono confinati al Core 0 e non devono introdurre un lock richiesto dal dominio del motore. La configurazione ricevuta dall'esterno segue lo stesso principio di limitazione: sono disponibili due slot statici da 32 KiB e una richiesta che supera tale dimensione viene rifiutata prima del parsing, evitando che un payload remoto determini un consumo non controllato di memoria.

La dimensione degli stack è stabilita per ciascun task in base al carico previsto

e viene verificata sotto stress tramite lo stack high-water mark. Una configurazione è accettabile soltanto quando rimane libero almeno il 25% dello stack e, in ogni caso, non meno di 1024 byte. Analogamente, ogni task registra il massimo tempo di esecuzione, i deadline miss e il massimo riempimento delle proprie code. Il sottosistema di diagnostica raccoglie queste misure sul Core 0 e non le stampa direttamente dal percorso del motore, dove il logging altererebbe proprio le grandezze che si intendono osservare.

Il salvataggio di dati persistenti è effettuato esclusivamente da `storage`. Una scrittura in flash è consentita soltanto quando la modalità AFSM corrente non autorizza il controllo e il motore è fermo; OTA e sostituzione del filesystem richiedono lo stato *Shutdown*. Il vincolo tiene conto del fatto che le operazioni su SPI flash possono influire su entrambi i core: collocare il task sul Core 0, da solo, non garantirebbe l'isolamento temporale del controllo.

4.2.8 Regole di errore e degradazione

Ogni errore viene rilevato dalla classe che possiede i dati necessari a riconoscerlo. La stessa classe invia direttamente all'AFSM gli eventi che possono modificare la modalità operativa e mette a disposizione della diagnostica i dettagli utili alla successiva analisi. L'invio delle informazioni diagnostiche è non bloccante e avviene sempre dopo la strategia di sicurezza: se viene persa la sincronizzazione, per esempio, il percorso del pick-up asserisce subito il proprio interlock e soltanto dopo rende osservabile l'evento agli altri servizi.

Gli interlock della CDI sono raccolti in una maschera atomica i cui bit hanno proprietari distinti. La perdita della posizione angolare dell'albero motore, una modalità che non autorizza il controllo, una configurazione non valida, un errore del percorso di uscita o una richiesta di shutdown possono quindi inibire l'accensione senza attendere il completamento di una transizione distribuita. Prima di ogni avvio del timer, `engine_control` controlla nuovamente la maschera. Se nel frattempo la modalità operativa è diventata più restrittiva, l'azione in sospeso viene annullata e l'uscita è forzata al livello basso; un impulso della CDI già iniziato viene invece registrato come interrotto, poiché il software non può annullare una scarica fisica già avviata.

Un dato scaduto, non valido o incoerente non viene mai riutilizzato senza essere segnalato. La mancanza di dati obbligatori porta almeno a *ControlInhibited*, mentre la perdita di una funzione opzionale può condurre a *DegradedRunning* e disabilitare soltanto la strategia che dipende da essa. Anche il recupero è basato su nuove misurazioni valide e stabili: la conferma dell'operatore può autorizzare il tentativo, ma non sostituisce l'evidenza tecnica né permette di passare direttamente da *FaultLatched* a una modalità che comanda gli attuatori.

Le regole appena descritte garantiscono che i dati impiegati dal controllo siano validi e aggiornati; il controllo delle scadenze completa questa risposta, assicurando che anche l'elaborazione e l'attuazione restino entro tempi compatibili con la sicurezza. Il superamento isolato del budget temporale di `engine_control` invalida soltanto il piano per il giro successivo; se però tre superamenti si verificano nello stesso secondo, l'AFSM viene portata in *ControlInhibited*. Su una scala temporale diversa interviene il Task Watchdog: configurato con un timeout di 2 s, individua starvation e deadlock dei task principali, senza tuttavia entrare nel merito delle deadline nell'ordine dei microsecondi.

Anche la sequenza di shutdown antepone l'azione di sicurezza all'osservabilità: gli interlock vengono verificati prima di rendere disponibile la nuova modalità, la CDI viene forzata allo stato spento e gli attuatori ausiliari vengono disabilitati; soltanto dopo possono essere completati l'ultimo record diagnostico e le operazioni di persistenza ammesse.

4.3 Implementazione dell'AFSM

L'AFSM determina la modalità operativa dell'ECU a partire dalle condizioni osservate dai sottosistemi. Nel capitolo precedente questa responsabilità è stata descritta attraverso gli stati e le transizioni; nell'implementazione, gli altri componenti inviano all'AFSM eventi tipizzati e ricevono un'istantanea coerente della modalità. Oltre a codificare la macchina a stati, occorre garantire che ogni cambiamento venga deciso da un'unica entità e sia comunicato ai sottosistemi in un ordine compatibile con la sicurezza del controllo.

La politica di transizione è quindi separata dalle primitive del runtime. Le condizioni e gli invarianti possono essere verificati su host, mentre il task dedicato serializza gli eventi, applica le regole di precedenza e collega l'AFSM alle code, agli interlock e al meccanismo di comunicazione descritto nella sezione precedente. Questa separazione permette di seguire l'intero percorso di una decisione senza attribuire a FreeRTOS responsabilità proprie dell'AFSM.

4.3.1 Responsabilità e interfacce di implementazione

La classe `AfsmStateMachine` conserva lo stato corrente, il motivo dell'ultima transizione, la generazione della modalità, le degradazioni attive e i guasti memorizzati. Valuta ogni evento rispetto ai dati correnti e produce un esito di transizione, senza dipendere da task, driver o servizi della piattaforma. `ModeSupervisor` contiene l'istanza della macchina a stati ed esegue le operazioni il cui ordine deve essere osservabile: arbitra gli eventi, modifica i bit

di interlock assegnati all'AFSM, rende disponibile la nuova `ModeSnapshot` e risveglia i sottosistemi che la utilizzano. `ModeSupervisorTask`, infine, adatta questa logica a FreeRTOS senza introdurre altre condizioni o transizioni.

Questa ripartizione delle responsabilità esclude esplicitamente l'acquisizione e la validazione dei sensori, la ricostruzione dello stato motore, l'attivazione della configurazione e il calcolo delle decisioni di controllo. L'AFSM non programma nemmeno gli attuatori né possiede lo storico diagnostico: può asserire i propri interlock, mentre la cancellazione delle azioni pendenti e l'applicazione delle uscite sicure rimangono responsabilità di *Action Coordination*. Anche i dettagli di un guasto rimangono presso il sottosistema che dispone dei dati necessari a riconoscerlo; l'AFSM ne conserva soltanto la categoria richiesta per stabilire la modalità e il percorso di recupero.

Questa separazione è rappresentata nella Figura 4.4 attraverso le classi e le interfacce pubbliche. Le porte per task e ISR forniscono le stesse garanzie mediante `IAfsmEventSink`, pur adattandosi a primitive di piattaforma differenti, mentre `IAfsmEventSource` presenta al supervisore una sequenza già serializzata. La macchina a stati dipende soltanto da `ISystemEvidence` e `IMonotonicClock`; l'invio degli aggiornamenti, gli interlock e le notifiche restano sotto il controllo del supervisore.

4.3.2 Rappresentazione dello stato

La modalità è rappresentata da un'enumerazione che riprende gli stati della Figura 3.2; non viene quindi ricostruita a partire da valori booleani indipendenti. Il valore iniziale è *Uninitialized*, accompagnato dal motivo `Startup`, così anche la prima istantanea ha un significato esplicito. I dati necessari a valutare una guardia, come la presenza di degradazioni o la conferma di un guasto, appartengono allo stato della macchina e non sono distribuiti tra gli adattatori e i componenti che producono gli eventi.

I dati di stato sono raccolti in `ModeSnapshot`. Il campo `mode` indica la modalità efficace, mentre `reason` conserva l'evento o il dato che ha determinato l'ultimo cambiamento. Il contatore monotono `mode_generation` aumenta esattamente una volta per ogni transizione e permette ai sottosistemi di riconoscere una decisione preparata in un contesto ormai superato. L'istante in cui la nuova modalità diventa efficace è registrato in `transition_timestamp_us` mediante il riferimento temporale monotono comune, anziché mediante l'istante dichiarato da una sorgente non ancora verificata.

L'istantanea contiene inoltre `latched_fault_mask`, che raccoglie le categorie di guasto ancora attive. Un nuovo guasto può estendere questa maschera an-

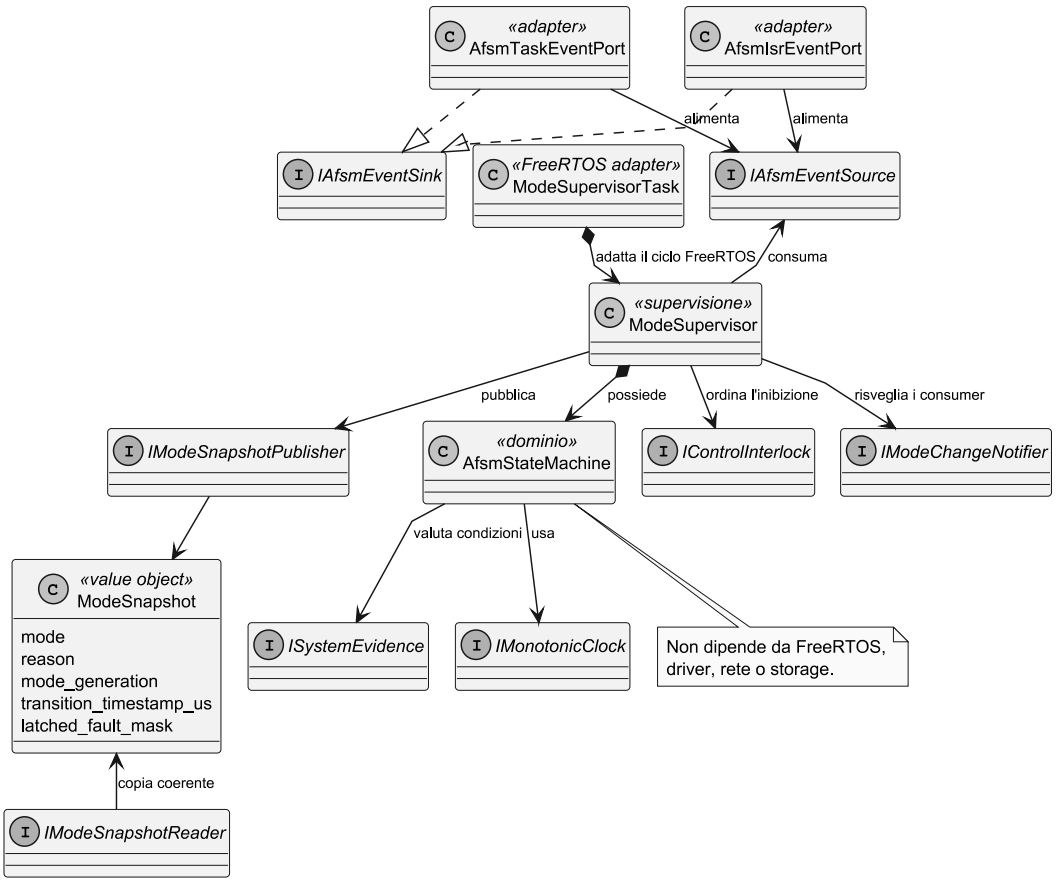

Figura 4.4: Classi e interfacce dell'AFSM.

che quando l'AFSM si trova già in *FaultLatched*; in tal caso la modalità non cambia, `mode_generation` non viene incrementata e il motivo della transizione originaria non viene perduto. L'aggiornamento rende comunque disponibili tutti i campi come una sola copia coerente, impedendo a chi la legge di combinare la modalità precedente con una maschera o un istante di transizione appartenenti a un aggiornamento successivo.

4.3.3 Eventi e specifica delle transizioni

Gli ingressi dell'AFSM sono oggetti immutabili, ciascuno specifico per un tipo di evento. Ogni istanza indica il tipo, il sottosistema che l'ha prodotta, l'istante di acquisizione sul riferimento monotono e la generazione della sorgente, insieme ai dati necessari a descrivere il fatto rappresentato. In questo modo, `SynchronizationLost` trasporta la posizione angolare dell'albero motore e la causa della perdita, mentre `ConfigurationInvalidated` identifica la generazione di configurazione interessata; non esiste un intero generico il cui significato debba essere dedotto a posteriori dal tipo dell'evento.

Gli eventi di disponibilità descrivono il completamento progressivo dell'avvio, dalla preparazione della piattaforma fino alla configurazione e alle osservazioni obbligatorie. Gli eventi legati alle condizioni fisiche comunicano, invece, la perdita o il recupero delle osservazioni, la sincronizzazione e i limiti degradati. A questi si affiancano le richieste esterne di conferma, arresto e reset e gli eventi interni con cui il supervisore rappresenta una scadenza superata o la saturazione di una coda. La provenienza resta tracciabile: una richiesta della WebUI non può dichiarare valida una misura e una scadenza non viene attribuita al sottosistema osservato.

La classificazione critica o normale determina l'ordine di elaborazione, non il significato dei dati trasportati. La perdita di sincronizzazione, l'invalidazione della configurazione, un guasto critico e una richiesta di arresto hanno precedenza sugli aggiornamenti relativi alla disponibilità o al recupero. Gli eventi con la stessa sorgente, categoria e generazione vengono accorpati; prima di ignorare un evento obsoleto, il supervisore lo confronta con l'istantanea del sottosistema responsabile. In particolare, un guasto critico non viene scartato soltanto perché il suo istante di acquisizione precede una transizione indipendente: se l'ordine degli eventi non basta a stabilire la condizione corrente, il supervisore rilegge i dati e sceglie l'esito più restrittivo.

Le transizioni rimangono quelle già presentate nella Figura 3.2; l'implementazione ne precisa le condizioni e gli effetti. La sola ricezione di un evento non soddisfa una guardia, poiché `AfsmStateMachine` verifica configurazione, istante di acquisizione delle osservazioni, sincronizzazione e generazioni attraverso

`ISystemEvidence`. Una coppia stato–evento non ammessa lascia inalterata la modalità e viene riportata alla diagnostica senza bloccare il supervisore. Quando, invece, la guardia è soddisfatta, l'esito comprende la modalità di destinazione, il motivo, l'aggiornamento dei guasti e l'ordine da rispettare per l'interlock e l'invio.

4.3.4 Elaborazione concorrente degli eventi

Tutti i cambiamenti di modalità sono serializzati nel task `mode_supervisor`. Gli eventi possono provenire da un task o da una ISR, ma chi li produce non invoca direttamente la macchina a stati né accede al suo stato interno. Ogni evento viene depositato nella porta appropriata, che risveglia il task destinatario. Il supervisore esamina prima il canale critico, senza attendere che quello normale si svuoti, e limita il numero di eventi elaborati consecutivamente. In questo modo rispetta le precedenze di sicurezza senza permettere a una sorgente continua di monopolizzare il Core 1 o impedire l'aggiornamento del Task Watchdog.

Le scadenze temporali sono assolute e vengono rivalutate anche quando le code rimangono occupate. Il superamento della scadenza di sincronizzazione genera quindi un evento interno senza dipendere da un periodo di inattività del sistema. La saturazione delle code segue una regola analoga: sul canale critico rimane memorizzata una segnalazione indipendente dall'evento perso e l'AFSM entra in *FaultLatched*; la saturazione del canale normale porta almeno a *ControlInhibited* e impone di rileggere tutti i dati prima del recupero. Nessuna transizione può attendere operazioni di rete, archiviazione o registrazione, né un mutex con un tempo di attesa non limitato.

Il percorso decisionale è riassunto nella Figura 4.5. Il diagramma non ripete l'implementazione delle code, già trattata nel modello di comunicazione, ma mostra le precedenze, il controllo delle scadenze e il limite al numero di eventi consecutivi. Per ogni evento selezionato, il supervisore ne verifica la provenienza e la validità, rilegge i dati necessari e rende disponibile una nuova modalità soltanto se la guardia produce una transizione effettiva.

4.3.5 Comportamento dei sottosistemi destinatari

Una transizione restrittiva deve rendere inefficace il controllo prima di comunicare la nuova modalità. `ModeSupervisor` attiva quindi `ModeInhibitsControl`, invia la `ModeSnapshot` completa e soltanto dopo notifica *Control and Decision* e *Action Coordination*. Alla diagnostica viene inviato lo stesso esito attraverso un canale non bloccante, sempre dopo l'azione conservativa. Nel caso di una

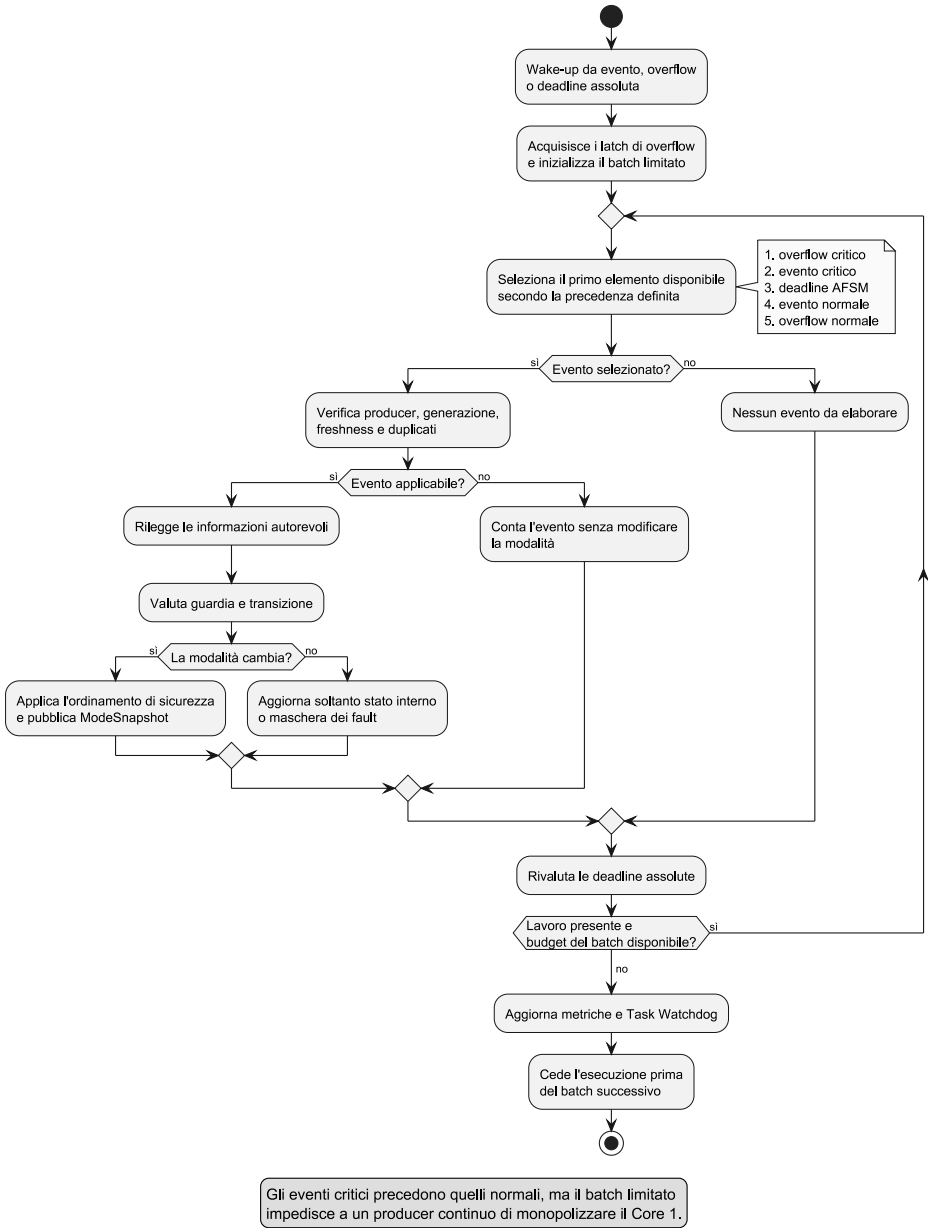

Figura 4.5: Arbitraggio e trattamento degli eventi nel task dedicato all'AFSM.

transizione permissiva, invece, vengono prima ricontrollati i dati e le generazioni, quindi viene resa disponibile la nuova istantanea; l'interlock di modalità può essere rimosso soltanto se la destinazione è *Running* o *DegradedRunning*.

Il solo ordinamento interno all'AFSM non sostituisce gli interlock dei sottosistemi che rilevano una condizione critica. Poiché `engine_control` ha priorità maggiore di `mode_supervisor`, il sottosistema che rileva il problema deve attivare il proprio bit di interlock prima di inviare l'evento. La perdita della sincronizzazione mostrata nella Figura 4.6 segue proprio questa regola: *State Interpretation* inibisce immediatamente il percorso che dipende dalla posizione angolare dell'albero motore, poi l'AFSM consolida la condizione portando la modalità in *ControlInhibited*. Il diagramma separa inoltre la decisione di modalità dall'attuazione, che rimane presso `ActionCoordinator`.

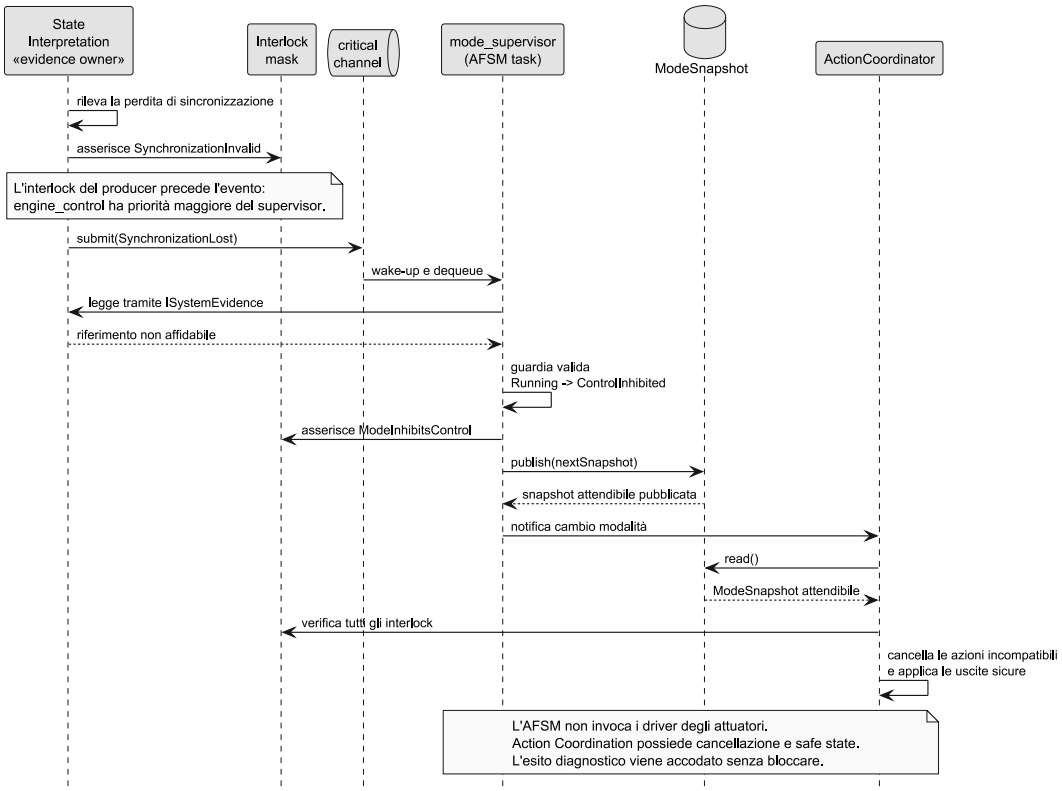

Figura 4.6: Propagazione ordinata di una transizione restrittiva causata dalla perdita di sincronizzazione.

Ogni decisione e ogni azione conserva la `mode_generation` sotto la quale è stata prodotta. Prima di applicarla, `ActionCoordinator` rilegge modalità, generazione, configurazione, sincronizzazione e interlock; un'istantanea indisponibile o una generazione cambiata rende l'azione non applicabile. Questo controllo

chiude la finestra tra la preparazione di un comando e la sua esecuzione, evitando che un'azione preparata in *Running* venga applicata dopo il passaggio a una modalità più restrittiva.

4.3.6 Memorizzazione dei guasti e recupero

L'ingresso in *FaultLatched* conserva la categoria del guasto e mantiene il controllo inibito anche quando l'evento che lo ha causato non è più presente nella coda. Ulteriori guasti estendono `latched_fault_mask` senza cancellare il motivo originario, così la diagnosi non viene ridotta all'ultimo problema osservato. La conferma dell'operatore esprime soltanto la volontà di tentare il recupero e non sostituisce i dati tecnici necessari; la segnalazione può essere rimossa esclusivamente quando il sottosistema responsabile rende disponibile una condizione valida, acquisita di recente e stabile per la categoria interessata.

I dati richiesti dipendono dalla natura del guasto. Una configurazione non valida richiede una nuova generazione completa e attiva; un problema del percorso di uscita richiede il raggiungimento dello stato sicuro e un test di riabilitazione; un errore temporale richiede che il sottosistema responsabile dimostri nuovamente il rispetto dei propri vincoli per l'intervallo stabilito. Dopo la saturazione di una coda dell'AFSM non basta, invece, verificare che il canale sia tornato disponibile: il supervisore deve rileggere le istantanee di tutti i sottosistemi responsabili, perché non può conoscere il contenuto dell'evento perso.

Il recupero mantiene una progressione conservativa. Da *FaultLatched* si ritorna prima a *Ready*, quindi si attraversa *Synchronizing* e soltanto una nuova sincronizzazione permette di entrare in *Running* o *DegradedRunning*. Da *ControlInhibited* il percorso può riprendere da *Ready* o da *Synchronizing* in base ai dati da riacquisire, ma non raggiunge direttamente una modalità che autorizza gli attuatori. L'ingresso in *Shutdown* interrompe ogni percorso di recupero, mantiene gli interlock e conclude il trattamento degli eventi fino al successivo avvio.

4.3.7 Verifica dell'AFSM

La separazione di `AfsmStateMachine` dal runtime permette di verificare su host tutte le righe della matrice di riferimento delle transizioni. Per ciascuna guardia vengono verificati sia l'esito vero sia quello falso, includendo eventi duplicati, generazioni obsolete, combinazioni stato–evento non valide e guasti simultanei. Gli stessi test controllano che `mode_generation` aumenti una sola volta per cambiamento, che *Running* richieda configurazione, osservazio-

ni e sincronizzazione valide e che la conferma dell'operatore non consenta di aggirare i requisiti di recupero.

I test del runtime e quelli HIL completano la verifica delle proprietà che non appartengono alla sola macchina a stati. Devono introdurre un evento critico durante il traffico normale, mantenere occupati i canali mentre scade un vincolo temporale, saturare separatamente la coda critica e quella normale e provocare una transizione restrittiva con un'azione CDI ancora in attesa. La verifica deve inoltre controllare il comportamento quando un'istantanea non è disponibile, la generazione cambia durante la lettura o l'AFSM entra in *Shutdown* con il Task Watchdog attivo. I risultati quantitativi e la copertura delle transizioni saranno riportati nella sezione di validazione; questa sezione definisce invece il comportamento che le prove devono dimostrare.

4.4 Implementazione dell'Environment Observation

Nel capitolo precedente l'*Environment Observation* è stato introdotto come confine tra i segnali elettrici e le informazioni usate dal dominio della ECU. L'implementazione rende concreto quel confine separando l'acquisizione legata alle periferiche dalle regole con cui una misura viene convertita, validata e resa disponibile ai consumer. Il sottosistema rimane così responsabile dell'istante di acquisizione, dell'unità fisica, della qualità e della continuità di ogni sorgente, ma non ricostruisce lo stato motore né stabilisce quale strategia o azione debba essere applicata.

La descrizione seguente mantiene come riferimento il firmware `v1.1`. La PCB `v1.0` permette di esercitare fisicamente il pick-up, il TPS e gli ingressi digitali, mentre non integra ancora i circuiti dedicati a EGT, temperatura acqua e knock richiamati nella Figura 3.1. Per questi ultimi, il confine software, le conversioni e il comportamento in errore sono verificabili attraverso le source di test e gli adapter sostitutivi, ma non vengono presentati come una misura ottenuta dalla scheda di riferimento.

4.4.1 Realizzazione concreta e confini

Il codice è diviso tra i componenti ESP-IDF `sensor_drivers` e `sensors`. Il primo possiede gli handle delle periferiche e contiene gli adapter per ADC, GPIO, MCPWM e SPI; il secondo raccoglie i value object, i processor e gli application service che dipendono soltanto da port C++ tipizzati. Una callback trasferisce quindi un record raw già associato alla timebase comune, mentre il

processor decide se quel record possa diventare un'osservazione. La dipendenza procede dal dominio verso il port astratto e non verso ESP-IDF, consentendo di riprodurre la stessa elaborazione su host con sequenze deterministiche.

Anche il contesto di esecuzione resta distinto dal comportamento del sensore. `FastInputTask`, `SlowSensorTask` e `KnockProcessTask` traducono release, notification e deadline nelle chiamate ai rispettivi service, mentre `EngineControlObservationAdapter` colloca la fase del pick-up nel task `engine_control`. I task non applicano filtri né possiedono una seconda rappresentazione della health. Lo stato modificabile rimane nelle classi che lo interpretano: `PickupSensor` conserva l'ultima cattura accettata e i contatori di recupero, `TpsSensor` lo stato del filtro e `KnockFeatureExtractor` il background e l'hysteresis.

Le operazioni pubbliche riflettono la diversa natura delle sorgenti anziché forzarle dietro una generica `ISensor::read()`. Un frame ADC, un edge asincrono e una finestra knock correlata alla rivoluzione richiedono infatti trigger e failure mode differenti. I risultati condividono, invece, un envelope comune: `SensorReading<T>` rappresenta l'ultimo valore di uno stream, mentre `SensorEvent<T>` conserva un evento ordinato che non può essere sostituito dall'elemento successivo. Entrambi riportano timestamp, sequenza, `config_generation`, `health`, `quality`, `origin`, `use` e la fault mask della sorgente.

Il confine fra acquisizione hardware e validazione di dominio è reso esplicito anche dalle interfacce pubbliche riportate nel Listato 4.1. Il port espone soltanto record raw e operazioni sulla sorgente, mentre `PickupSensor` accetta il record convertito nella timebase comune e restituisce tipi del dominio.

Questi campi impediscono che un numero sostitutivo venga confuso con una misura corrente. `health` descrive la continuità della sorgente e `quality` la singola osservazione; `ObservationUse` stabilisce poi se il valore sia ammesso per `NormalControl`, soltanto per una strategia degradata o esclusivamente per la diagnostica. La generazione della modalità AFSM non compare nelle letture indipendenti dal controllo, perché l'acquisizione prosegue anche quando gli attuatori sono inibiti. Sarà il consumer a combinare osservazioni, configurazione e modalità in un contesto coerente.

I valori correnti sono comunicati mediante `VersionedSnapshot<T>` con un solo writer e più reader bounded. Il payload viene accettato soltanto se il contatore pari/dispari rimane invariato durante la copia; dopo tre tentativi falliti, il reader considera la snapshot non disponibile per la release corrente e non usa il contenuto parziale. Le catture del pick-up, le richieste del quick

```
class IEdgeCaptureSource {
public:
    virtual OperationStatus start() = 0;
    virtual ReadStatus read_capture(RawPickupCapture& out) = 0;
    virtual std::size_t pending_count() = 0;
    virtual std::size_t discard(std::size_t count) = 0;
    virtual void stop() = 0;
};

class PickupSensor {
public:
    SensorEvent<PickupCaptureEvent>
    process(const EdgeCapture& capture);

    SensorReading<PickupStatus>
    check_source_timeout(TimestampUs now);
};
```

Listato 4.1: Contratti tipizzati per l'acquisizione e la validazione del pick-up.

shifter e le transizioni di fault attraversano invece `StaticEventQueue<T,N>` a capacità limitata, perché la sostituzione di un record non letto nasconderebbe una discontinuità significativa.

La Figura 4.7 riunisce questi elementi senza rappresentare ogni sensore concreto. Il diagramma evidenzia la direzione delle dipendenze, separa gli execution adapter dai service e mostra che snapshot e queue appartengono al confine di uscita, non ai consumer che le leggono.

4.4.2 Comportamento a runtime e meccanismi di acquisizione

L'inizializzazione procede dagli oggetti interni verso le periferiche. Il composition root crea clock, processor, storage statico, queue e service, quindi configura ADC, GPIO, SPI e MCPWM e registra le callback. Le prime snapshot dichiarano la health `Stabilizing` e non sono ancora ammesse per il controllo; gli interrupt vengono abilitati soltanto quando i consumer e i relativi buffer sono disponibili. Una sorgente opzionale esclusa dalla configurazione dichiara la health `Disabled`, mentre il mancato avvio del pick-up, unica osservazione obbligatoria, impedisce la readiness e mantiene attivo l'interlock del riferimento angolare.

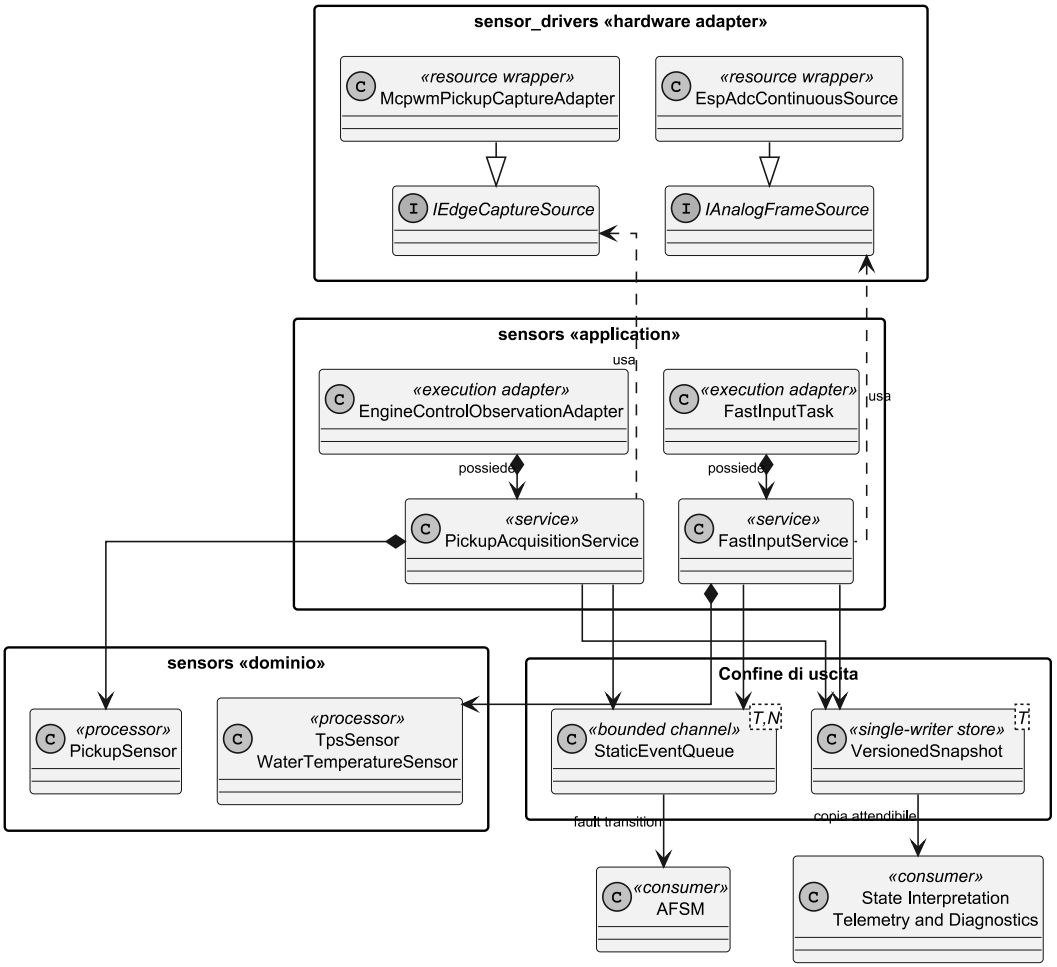

Figura 4.7: Classi, port e resource wrapper che realizzano il confine dell'Environment Observation.

La configurazione arriva come `ActiveConfigSnapshot` già validata e dotata di una sola `config_generation`. Ogni service la adotta al confine della propria release e sostituisce in modo atomico lo stato del processor, così un record non può dipendere da parametri aggiornati solo in parte. Il pick-up applica i nuovi limiti fra due batch e torna `Stabilizing`; la pipeline ADC completa il frame TPS–acqua in corso, mentre il knock attende la chiusura della finestra. Un cambio del binding hardware richiede invece *ControlInhibited*, arresto dell'adapter e riavvio esplicito della sorgente.

Il percorso del pick-up è quello nel quale l'ordine di esecuzione ha le conseguenze più immediate. Il falling edge condizionato su GPIO21 viene catturato da MCPWM con interrupt di livello 3; la callback copia tick, edge, status, `isr_sequence` e `config_generation` in un `RawPickupCapture`, lo inserisce nella queue da 16 record e notifica `engine_control`. Conversione in microsecondi, plausibilità e stima rimangono fuori dall'ISR, che usa esclusivamente storage già allocato e non esegue logging.

Il wake-up non porta subito alla validazione della nuova cattura. Come mostra la Figura 4.8, `engine_control` controlla prima gli interlock e il piano di accensione preparato nella rivoluzione precedente, quindi arma il one-shot della CDI. Solo dopo questo fast path legge il backlog, elabora in FIFO fino a quattro record e consegna allo `StateInterpreter` gli eventi accettati. La stima prodotta dalla cattura corrente contribuisce così al piano della rivoluzione successiva e non modifica retroattivamente l'azione già in scadenza.

`CaptureTickConverter` estende il contatore MCPWM a 32 bit e lo collega alla timebase di `esp_timer`. Una diminuzione superiore a metà range identifica un wrap, mentre una diminuzione minore rende il record non monotono; la conversione usa aritmetica intera e conserva il resto della divisione per non accumulare deriva. Dopo un reset, una modifica della risoluzione o un'assenza più lunga del periodo di wrap, il converter stabilisce un nuovo anchor e riporta la sorgente a `Stabilizing`, anziché tentare di ricostruire intervalli non osservati.

Con un impulso per rivoluzione, `PickupSensor` accetta intervalli compresi fra 2400 µs e 600 000 µs, che corrispondono al range inclusivo 100–25 000 RPM. Quattro edge, e quindi tre intervalli plausibili consecutivi, portano la health a `Valid`. Questa condizione attesta la continuità del solo percorso di acquisizione: fase, posizione angolare e sincronizzazione restano informazioni dello `StateInterpreter`. Prima che esista un intervallo valido, il timeout della sorgente è pari a 900 000 µs; in seguito, lo `StateInterpreter` applica un proprio timeout più rapido per revocare la sincronizzazione quando il controllo lo richiede.

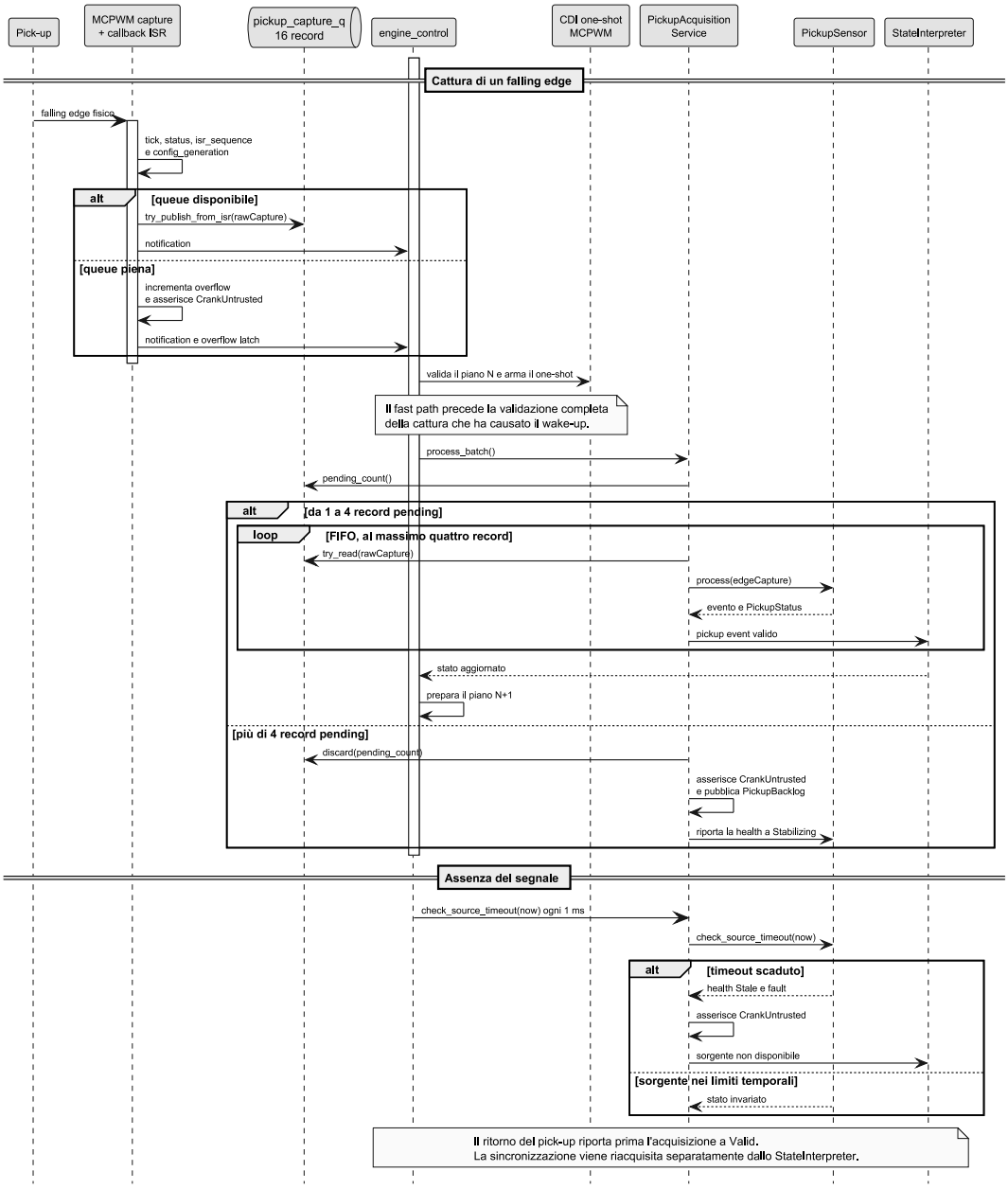

Figura 4.8: Acquisizione, validazione e trattamento dell'assenza del segnale di pick-up.

Gli ingressi analogici seguono una pipeline periodica distinta. ADC1 continuous alterna TPS e temperatura acqua a 300 sample/s complessivi, formando un frame ogni circa 6.667 ms; l'adapter ricostruisce il timestamp di ciascun sample dalla frequenza effettiva e da un contatore monotono, perciò i due canali non ricevono lo stesso istante convenzionale. La calibrazione ESP-IDF converte il codice in millivolt e un errore di calibrazione lascia il record non valido, senza ricorrere a una conversione lineare approssimata.

Per un sample elettricamente valido, `TpsSensor` calcola la posizione in permille con $((V - V_{closed})\,1000)/(V_{open} - V_{closed})$, usando intermedi signed a 64 bit. Il risultato viene arrotondato e saturato fra 0 e 1000 soltanto quando la tensione resta nel range elettrico ammesso; oltre i rail configurati viene comunicato un fault, non una posizione estrema apparentemente valida. Il filtro IIR conserva la stessa rappresentazione intera e viene reinizializzato quando cambia configurazione. Un TPS costante non basta invece a dichiarare `Stuck`, poiché una posizione immobile può essere un comportamento fisico legittimo.

Il quick shifter usa una queue di edge e produce una sola `QuickShiftRequest` dopo che il livello attivo ha superato il debounce; il rilascio stabile riarma l'ingresso senza generare una seconda richiesta. Il map switch non conserva invece ogni edge, ma viene campionato dal task ogni 10 ms e cambia posizione soltanto dopo la conferma del livello. Questa differenza evita di attribuire la stessa semantica a un comando one-shot e a uno stato selezionabile.

Le misure lente seguono il medesimo modello di interfaccia, ma non la stessa attivazione. `slow_sensor` esegue la lettura EGT a 10 Hz con timeout SPI di 2 ms, mentre la finestra knock viene armata da `engine_control` per una specifica rivoluzione e completata da `knock_process`. Il contesto conserva rivoluzione, configurazione e modalità effettivamente usate; una finestra sovrapposta viene rifiutata e un risultato tardivo non viene associato alla rivoluzione successiva. Il background del knock è aggiornato con un filtro intero 95/5 e l'hysteresis usa soglie 2,0 e 1,5, evitando divisioni floating-point nel percorso per rivoluzione.

4.4.3 Guasti, sicurezza e vincoli temporali

Ogni processor rileva soltanto le condizioni per le quali possiede l'evidenza e aggiorna la health della propria sorgente. Un `FaultTransition::Asserted` viene emesso quando il fault compare e un `FaultTransition::Cleared` quando il recupero è stato dimostrato; le occorrenze intermedie incrementano i contatori locali senza riempire i canali con eventi equivalenti. `SensorHealthService` acquisisce ogni 20 ms le snapshot source-specific e produce una sola vista aggregata per AFSM e diagnostica, ma gli eventi critici del pick-up raggiungono direttamente l'AFSM senza attendere tale aggregazione.

La macchina comune illustrata nella Figura 4.9 impedisce un recupero basato sulla sola scomparsa dell'errore. Dopo l'avvio o la reinizializzazione una sorgente attraversa `Stabilizing` e deve fornire il numero previsto di campioni validi prima di raggiungere `Valid`; anche il ritorno da `Failed` segue lo stesso passaggio. `Stale` dipende dal clock monotono e dall'ultimo timestamp accettato, non dalla frequenza con cui un consumer legge la snapshot.

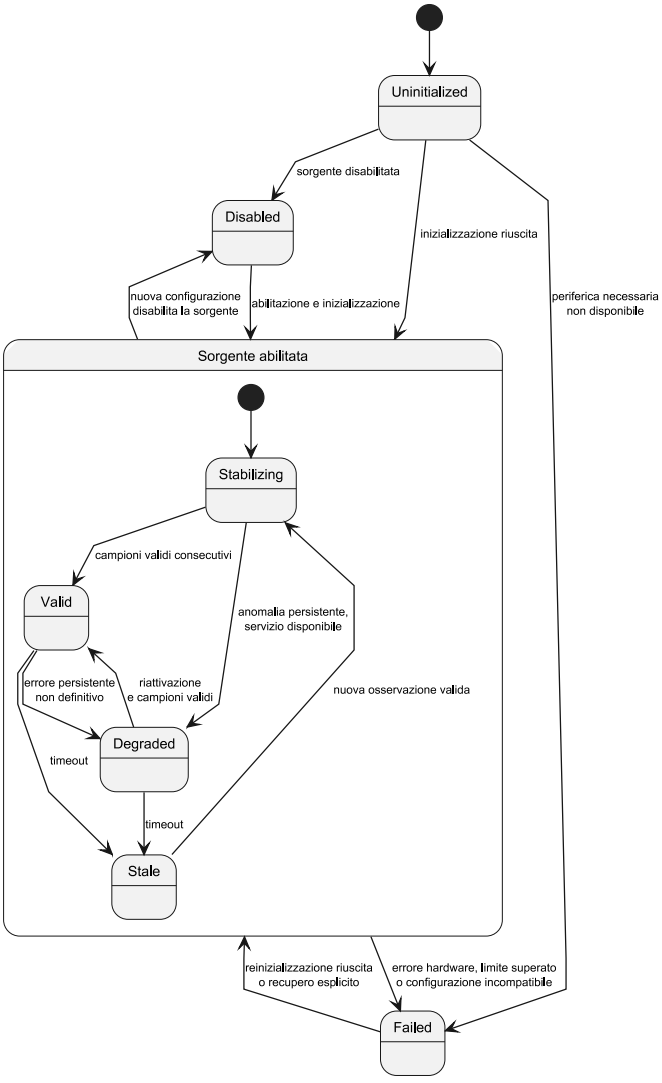

Figura 4.9: Stati comuni della salute di una sorgente e condizioni di recupero.

La perdita del pick-up segue una risposta più restrittiva rispetto a quella di un sensore opzionale. Se la queue è piena, la callback scarta il nuovo record, incrementa il contatore di overflow, attiva `CrankUntrusted` attraverso una porta set-only e risveglia comunque `engine_control`; non elimina un record prece-

dente, perché nasconderebbe il punto nel quale la sequenza è diventata incompleta. Se il task trova più di quattro record pending, invalida l'intero batch, svuota la storia non più affidabile e riporta `PickupSensor` a `Stabilizing`. In entrambi i casi l'evento `MandatoryObservationLost` consente all'AFSM di consolidare la modalità, ma l'interlock rende subito inutilizzabile il riferimento angolare.

La forma ridotta del meccanismo è mostrata nel Listato 4.2. La dimensione del batch viene verificata prima di estrarre qualsiasi record: il ramo di overload scarta quindi l'intera sequenza osservata, asserisce l'interlock e impedisce una validazione parziale.

```
PickupBatchResult PickupAcquisitionService::process_batch()
{
    const auto pending = source_.pending_count();

    if (pending > max_batch_size) {
        const auto discarded = source_.discard(pending);
        interlock_.assert_crank_untrusted();
        fault_sink_.publish(Fault::PickupBacklog);
        pickup_sensor_.reset_to_stabilizing();

        return PickupBatchResult::rejected(discarded);
    }

    return process_fifo(pending);
}
```

Listato 4.2: Applicazione atomica della strategia di backlog del pick-up.

La callback può soltanto attivare questo interlock. La sua rimozione viene richiesta da `engine_control` con il confronto della generazione del fault e della nuova sincronizzazione, dopo che l'acquisizione è tornata `Valid` e lo `StateInterpreter` ha prodotto un riferimento affidabile. Una generazione non coincidente lascia il bit attivo e rimanda la valutazione al ciclo successivo, evitando che un recupero precedente cancelli un fault comparso nel frattempo.

Per le sorgenti opzionali, invece, un overrun, un valore fuori range o un timeout rende non disponibile la sola osservazione interessata e può portare la health a `Degraded`, `Stale` o `Failed`. Un fallback è ammesso solo se compare nella configurazione validata e conserva `ValueOrigin::Fallback` e `ObservationUse::DegradedOnly`; l'ultimo valore misurato resta `LastKnown` per la diagnostica e non viene presentato come dato corrente. L'AFSM deci-

de l'eventuale modalità degradata, mentre *Control and Decision* verifica se la propria strategia dichiari esplicitamente quella dipendenza.

La stessa distinzione vale per la saturazione dei canali. La perdita di una richiesta quick shifter produce un fault di delivery e non viene ricostruita dallo stato elettrico successivo; un overrun ADC aggiorna la health senza inventare campioni, mentre un risultato knock mancante lascia invariato il background e azzera il percorso di recovery. Se il sink AFSM critico non può accettare un evento, il producer attiva l'inhibit locale, aggiorna un overflow latch persistente e risveglia il supervisor. La perdita di un record diagnostico incrementa invece un contatore e non ritarda l'acquisizione.

I budget riflettono queste diverse conseguenze. La callback del pick-up esegue un lavoro costante e il tratto callback–arm del one-shot deve terminare entro 75 µs; la fase completa di `engine_control` dispone di 500 µs. `fast_input` completa un frame entro 400 µs, `knock_process` un risultato entro 1.5 ms e `slow_sensor` la propria release entro 5 ms. Le latenze fisiche edge–callback, enqueue–release e validazione completa vengono registrate separatamente, così un valore aggregato non viene attribuito a una sola fase.

Queue, snapshot, descriptor DMA, stack e record delle callback sono allocati prima dell'apertura della barriera di startup. Dopo tale istante i task del Core 1 non eseguono allocazioni dinamiche e il percorso del pick-up non usa mutex; le capacità di 16 record per pick-up e GPIO, otto frame per il pool ADC, otto richieste quick shifter e otto contesti knock assorbono soltanto burst limitati. L'occupazione massima, i deadline miss e gli stack high-water mark sono aggiornati localmente e letti dalla diagnostica senza produrre log nel percorso critico.

4.4.4 Verifica e limiti

La separazione fra adapter e processor permette di verificare su host le regole di conversione senza dipendere dalla temporizzazione reale delle periferiche. Le source deterministiche esercitano configurazioni valide e non valide, timestamp duplicati o fuori ordine, limiti inclusivi, mismatch di generazione, transizioni di health e recupero. La matrice che combina `health`, `quality`, `origin` e `ObservationUse` viene controllata anche nei casi meno immediati, come un campione `Valid` ma `Suspect` o uno stato `Stale` accompagnato da un fallback configurato.

Per il pick-up, i test coprono wrap e re-anchor del contatore, gli intervalli limite di 2400 µs e 600 000 µs, il timeout iniziale e la progressione di quattro edge. Un batch da quattro record deve essere elaborato in FIFO, mentre da cinque record

in poi deve essere scartato integralmente; le prove dell'interlock controllano inoltre che un clear con una generazione obsoleta venga rifiutato. Le pipeline analogiche vengono esercitate con una frequenza effettiva diversa da quella richiesta, errori di calibrazione, frame incompleti e pool pieno, verificando che i timestamp dei due canali rimangano distinti e non accumulino deriva.

Le prove target e HIL collegano poi i contratti software alle periferiche reali. Il generatore del pick-up deve coprire l'intero range 100–25 000 RPM e includere interruzioni, rampe e almeno un milione di eventi consecutivi al regime massimo; gli ingressi digitali vengono sottoposti a bounce e burst oltre la capacità delle queue, mentre l'ADC viene portato ai limiti e in overrun. Ogni misura di latenza deve indicare strumento, carico, numero di osservazioni e massimo rilevato. I risultati quantitativi appartengono alla sezione di validazione e non vengono anticipati qui come se i budget di progetto fossero misure.

Rimane inoltre il limite hardware già richiamato all'inizio della sezione. Sulla PCB `v1.0` le prove strumentali possono sostenere le asserzioni relative al pick-up, al TPS e agli ingressi digitali; per EGT, acqua e knock dimostrano soltanto il comportamento del firmware con source controllate finché una revisione successiva non integra i relativi circuiti. Questa distinzione mantiene tracciabile il livello di evidenza senza rimuovere dal modello software i port necessari alla configurazione completa dell'ECU.

4.5 Sviluppo hardware

La parte hardware realizza fisicamente le interfacce logiche dell'ECU mediante circuiti che acquisiscono i segnali del motore e comandano i relativi attuatori. La descrizione fa riferimento alla PCB `v1.0`, già introdotta all'inizio del capitolo, ed è accompagnata da porzioni dello schema elettrico e da viste del layout e della scheda assemblata.

La scheda comprende anche un simulatore del segnale di pick-up, costruito attorno al trasformatore `T201`, e un circuito indipendente per il quick shifter basato sui componenti `CPC2125N` e `AT223-CuH-S`. Questi blocchi sono stati sviluppati per prove su un'applicazione separata, dotata di un'ECU preesistente; non appartengono quindi alla centralina qui descritta e non vengono trattati ulteriormente.

4.5.1 Requisiti e interfacce della scheda

La scheda riceve dal veicolo un'alimentazione nominale di 12 V attraverso il connettore principale `MX23A18NF1` a 18 poli. L'ingresso è protetto contro l'in-

versione di polarità e filtrato da una rete LC che limita i disturbi presenti sulla linea. La tensione di ingresso, purché compresa tra 6.5 V e 25 V, alimenta un convertitore buck che fornisce al resto della scheda una tensione di 5 V; il modulo Wemos ricava poi internamente i 3.3 V richiesti dalla logica dell'ESP32-S3. Questa successione separa l'alimentazione del veicolo dai segnali logici e fornisce anche la tensione necessaria ai circuiti integrati ausiliari.

Nel prototipo, il segnale analogico del TPS, compreso fra 0 e 3.3 V, raggiunge `GPIO7/ADC1_CH6`; quick shifter e map switch sono ingressi digitali con pull-up collegati rispettivamente a `GPIO9` e `GPIO14`. Il sensore VR dell'albero motore non raggiunge direttamente il SoC: il suo segnale differenziale viene condizionato e trasformato in un fronte digitale acquisito su `GPIO21` dalla periferica MCPWM.

Sul lato delle uscite, la scheda pilota la scarica della CDI e i carichi induttivi associati al Power Jet e all'Exhaust Valve senza esporre i GPIO alle tensioni dei rispettivi stadi di potenza. Wi-Fi e Bluetooth LE sono già integrati nel modulo, mentre il collegamento USB-C del Wemos permette programmazione e debug senza occupare ulteriori pin del connettore principale. I circuiti dedicati a EGT, knock sensor e temperatura dell'acqua non sono presenti sulla PCB `v1.0` e non vengono quindi inclusi tra le interfacce fisicamente disponibili.

4.5.2 Scelta dei componenti e criteri di dimensionamento

Il Wemos LOLIN ESP32-S3 Mini, scelto anche per la sua sostituibilità, integra il SoC dual-core, 4 MB di flash, 2 MB di PSRAM e la connettività wireless. È montato tramite pin header e riceve la tensione di 5 V sul proprio ingresso VBUS; non è quindi necessario replicare sulla PCB il regolatore a 3.3 V e il circuito RF.

Per il condizionamento del pick-up è stato adottato il `MAX9924UAUB+`, un integrato progettato specificamente per l'interfacciamento di sensori a riluttanza variabile. Al suo interno, un amplificatore differenziale di precisione, un comparatore con soglia adattiva e un circuito di zero-crossing trasformano l'ampiezza variabile del segnale VR in impulsi digitali robusti, anche con segnali deboli o in presenza di rumore elevato.

Lo stadio CDI impiega SCR `TN1215-800G-TR`, caratterizzati da una corrente nominale di 12 A e da una tensione di blocco di 800 V. Il comando di gate è generato da un transistor PNP high-side, a sua volta pilotato da un NPN, mentre i diodi `US1M` proteggono i nodi di commutazione e gli `SS54` svolgono la rettifica prevista dallo stadio. La tensione di blocco degli SCR supera l'in-

tervallo di tensione continua indicato per la CDI; questo dato, tuttavia, non costituisce una qualifica rispetto a tutti i transitori possibili, che richiederebbe una caratterizzazione dedicata.

Il comando del Power Jet e della valvola di scarico impiega infine un IGBT `STGB20M65DF2`, pilotato dal MOSFET N-channel `AO3400A` e protetto da un diodo di ricircolo `US1M`. Questa organizzazione mantiene separata la generazione del comando dalla corrente assorbita dall'attuatore e consente allo stesso circuito di adattarsi anche a una futura implementazione di un'ECU TCI.

4.5.3 Sviluppo dello schema elettrico

Lo schema elettrico è stato sviluppato con KiCad 9 e suddiviso in fogli, uno per ciascun blocco funzionale; di seguito vengono richiamati i principali. Nel foglio di alimentazione, i 12 V provenienti dal filtro LC raggiungono il buck `TPS54202DDC`; la tensione di 5 V viene poi distribuita al resto della scheda. Sullo stesso foglio è presente anche un convertitore flyback basato su `UCC28910DR`, progettato per ricavare energia dallo statore del motore per applicazioni senza batteria esterna.

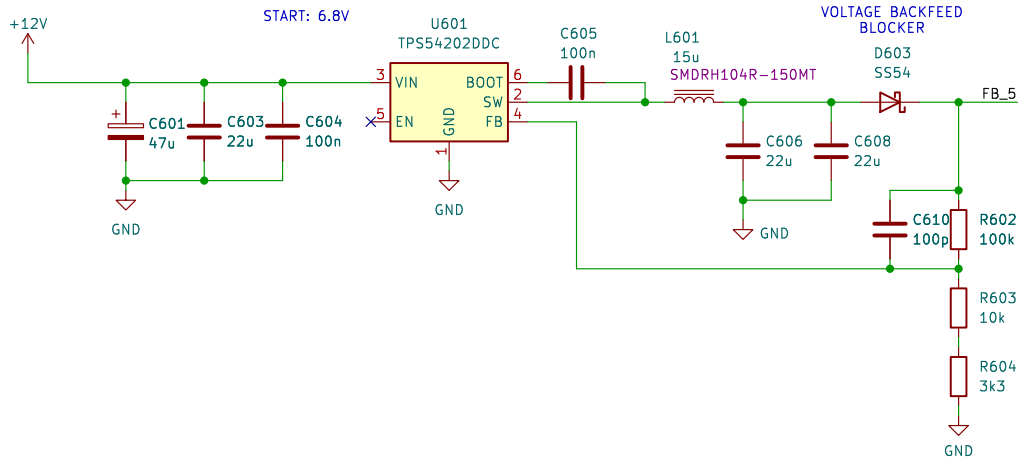

Figura 4.10: Alimentazione della scheda da 12 V al rail regolato a 5 V.

Nel foglio del pick-up, il segnale VR entra negli ingressi differenziali del `MAX9924UAUB+`, viene confrontato con una soglia adattiva e raggiunge il buffer che pilota il GPIO. Alcune piazzole saldabili permettono di selezionare la modalità di funzionamento dell'integrato; la configurazione predefinita mantiene abilitati lo zero-crossing e la soglia adattiva.

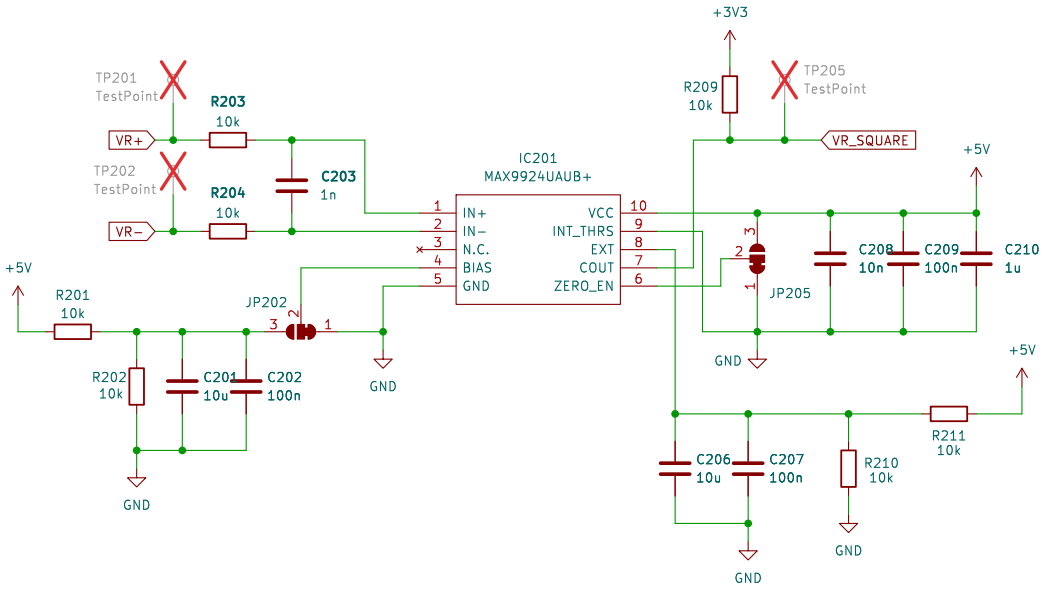

Figura 4.11: Condizionamento differenziale del pick-up mediante `MAX9924UAUB+`.

Il modulo Wemos e le reti necessarie al suo collegamento sono confinati nel foglio MCU, mentre i due fogli di uscita separano l'innesco della CDI dai driver degli attuatori ausiliari. Nel foglio CDI il comando logico attraversa la coppia di transistor di pilotaggio prima di raggiungere il gate dell'SCR. Sono inoltre previsti diodi di protezione, che possono essere inseriti nel circuito saldando le relative piazzole, per poter effettuare test aggiuntivi.

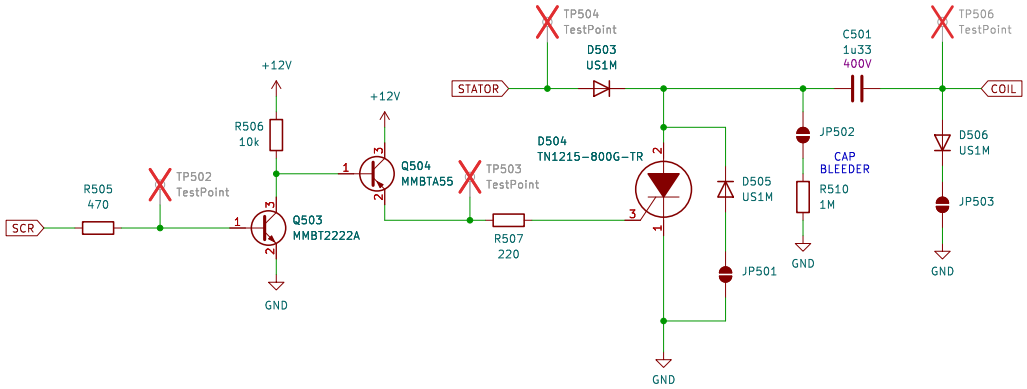

Figura 4.12: Pilotaggio degli SCR della CDI.

4.5.4 Layout PCB e sviluppo CAD

Anche il layout è stato sviluppato nello stesso progetto KiCad 9. Il modulo Wemos occupa la parte alta della scheda, in posizione centrale, e rimane accessibile tramite i pin header; i connettori `J101` e `J102` si trovano sul bordo inferiore, circondati dai blocchi che trattano segnali e livelli di potenza differenti.

La Figura 4.13 mostra il render del prototipo finale della scheda popolata, completo solo in parte a causa della mancanza di alcuni modelli 3D. La rappresentazione isometrica evidenzia l'altezza dei componenti, l'orientamento dei connettori e la densità di popolamento; la Figura 4.14 mostra invece l'enclosure della centralina.

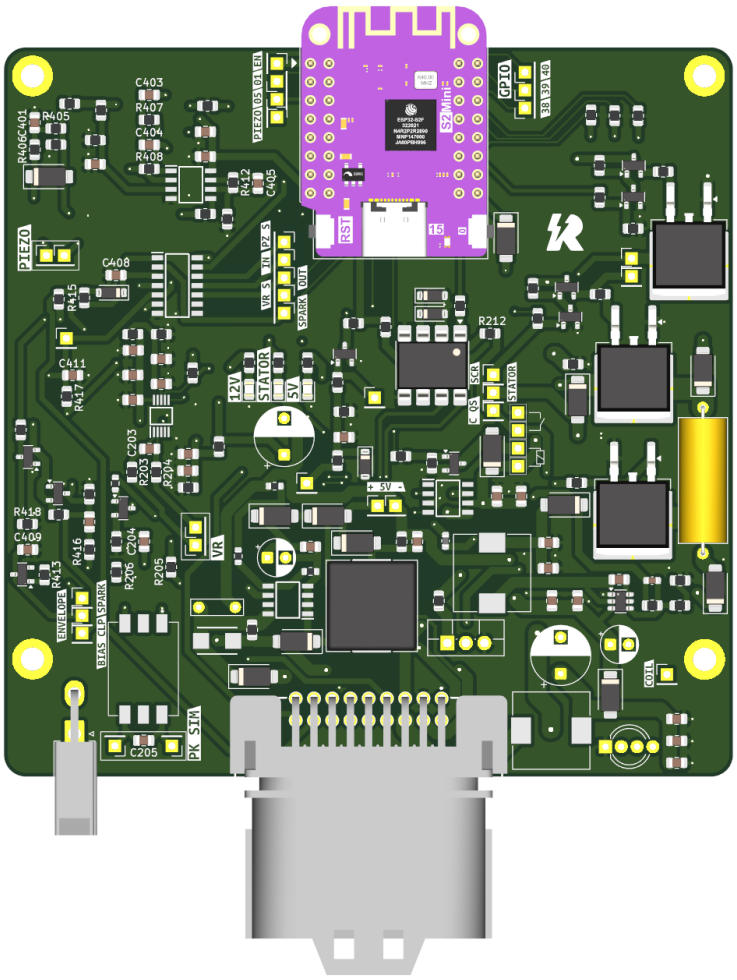

Figura 4.13: Render 3D della PCB popolata, impiegato per controllare orientamento e ingombro dei componenti.

Figura 4.14: Enclosure della centralina.

I file destinati alla produzione vengono generati mediante il plugin KiCad *fabrication-toolkit*, che raccoglie Gerber, BOM e coordinate di posizionamento a partire dalla stessa revisione CAD.

Capitolo 5

Validazione e test

I capitoli precedenti hanno definito l'architettura dell'ECU e del Digital Twin, le scelte con cui tali elementi sono stati realizzati nel prototipo e il progetto della PCB. La descrizione del progetto, tuttavia, stabilisce soltanto il comportamento atteso: per comprendere quali proprietà siano effettivamente sostenute dal sistema realizzato occorre collegare ogni affermazione a una prova ripetibile, a un risultato osservato e alle condizioni nelle quali quel risultato conserva validità.

Questo capitolo ha come obiettivo la verifica delle caratteristiche principali del lavoro svolto, a partire dal solo software fino alle prove funzionali dell'ECU. La trattazione definisce dapprima il perimetro delle verifiche e introduce il banco prova; prosegue con i test eseguibili su host generico e sul microcontrollore, con il collaudo elettrico della scheda e con la prova della CDI; si chiude con la verifica esplorativa della WebUI e del Digital Twin e con la discussione della tracciabilità e dei limiti dei risultati.

5.1 Obiettivi e metodologia di verifica

5.1.1 Configurazione di riferimento

L'oggetto principale della verifica è il prototipo composto dalla PCB `v1.0` e dal firmware `v1.1`, già assunti come riferimento nel capitolo precedente. I requisiti definiti nel secondo capitolo costituiscono l'insieme delle funzioni e delle proprietà da verificare.

Le prove che coinvolgono la PCB non coprono tutte le funzioni previste dall'architettura. La scheda consente di verificare il pick-up, il TPS e gli ingressi

digitali, mentre EGT, temperatura dell'acqua e knock non sono disponibili nella configurazione di riferimento; per questi ultimi è possibile verificare soltanto il comportamento del dominio software mediante emulazione.

5.1.2 Livelli di verifica

La validazione del prototipo è stata suddivisa su più livelli, poiché una singola prova non permette di risalire con precisione alla causa di un errore. Si parte dai test su host, che isolano le regole deterministiche dalle periferiche e dal runtime; si passa poi alle prove sul target, dove entrano in gioco ESP-IDF, FreeRTOS, interrupt, timer e risorse limitate; segue il collaudo della PCB, che verifica la conformità dei circuiti alle specifiche; si arriva infine alle prove funzionali sul banco, limitate ai percorsi e ai carichi effettivamente collegati all'ECU.

Questa separazione definisce anche il tipo di dato che ciascun livello può fornire. Un test su host, ad esempio, può dimostrare che una condizione dell'AFSM rifiuti una versione obsoleta, ma non è in grado di misurare la latenza di un interrupt; allo stesso modo, un log può mostrare l'ordine degli eventi, ma non l'istante esatto in cui un'uscita elettrica cambia stato.

Livello	**Aspetto verificato**	**Dati prodotti**
Host	Logica deterministica dell'architettura	Esiti dei test
Target	Esecuzione del firmware sull'ESP32-S3	Log seriali e stack trace di esecuzione
PCB	Conformità dei circuiti della scheda alle specifiche	Misure elettriche su test point
ECU	Comportamento end-to-end della centralina	Telemetria su client e funzionamento elettrico

Tabella 5.1: Livelli di verifica impiegati per il prototipo.

5.2 Banco prova e strumentazione

5.2.1 Ruolo del banco prova

Il banco prova sostituisce il motore con sorgenti controllabili, che consentono di osservare l'ECU senza esporre il prototipo e la strumentazione alle condizioni

di un'installazione reale. Il suo impiego rende le prove ripetibili e, al tempo stesso, più semplici da eseguire.

Un motore reale richiederebbe infatti un ambiente dedicato, per via dei gas di scarico e del rumore prodotti, oltre a un banco motore vero e proprio, a un sistema di raffreddamento e alle relative dotazioni di sicurezza. A ciò si aggiunge la difficoltà di riprodurre più volte lo stesso punto di funzionamento: il regime di un motore reale dipende da fattori esterni difficili da mantenere costanti, come la temperatura ambiente o il carico applicato; inoltre, un malfunzionamento della centralina in prova potrebbe, nel caso peggiore, danneggiare il motore stesso. Il banco prova elimina queste dipendenze e trasforma ogni prova in un esperimento controllato, in cui le uniche variabili sono quelle introdotte deliberatamente.

La strumentazione del banco è stata scelta per riprodurre fedelmente i segnali che l'ECU riceverebbe da un motore reale, mantenendo però la possibilità di modificarli in tempo reale e in modo indipendente l'uno dall'altro. Gli ingressi analogici, come il TPS, sono emulati mediante potenziometri, quelli digitali mediante semplici pulsanti. Per le misure elettroniche è stato impiegato un oscilloscopio a due canali.

5.2.2 Progettazione e realizzazione

Il banco prova è stato progettato e realizzato appositamente per l'ECU in esame: il comportamento dell'albero motore è simulato da un motore elettrico, affiancato dal relativo controller. A tale scopo è stato scelto un motore brushless da 600 W, capace di raggiungere i 12 000 RPM, fissato su una struttura in acciaio progettata per accogliere, tramite un apposito supporto, diverse tipologie di statori. Nel caso specifico è stato impiegato uno statore Selettra PVL, completo del relativo rotore magnetico, il cui accoppiamento all'albero del motore elettrico è stato reso possibile dalla realizzazione di un perno conico in alluminio. L'intero progetto del banco è stato sviluppato mediante CAD 3D, in modo da poterlo replicare facilmente, in caso di future realizzazioni, tramite macchinari CNC.

5.3 Verifica del software

5.3.1 Test deterministici su host

La separazione tra la logica indipendente dall'hardware e i componenti che interagiscono con le primitive ESP-IDF consente di verificare una parte significativa del comportamento eseguendo i test dei sorgenti C++ direttamente

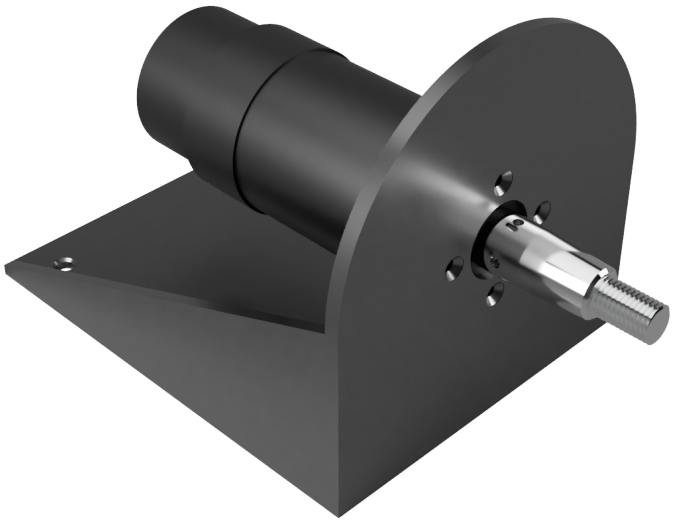

Figura 5.1: Render 3D della struttura con motore e adattatore conico.

sul computer. Nel sottosistema dei sensori, le suite disponibili verificano la conversione dei timestamp, il wrap dei contatori, la validazione delle misure analogiche, le condizioni di stale, il debounce degli ingressi digitali, l'acquisizione e la perdita della sincronizzazione, il recupero del pick-up e la gestione di backlog e overflow.

I test dedicati alla telemetria verificano una responsabilità differente: controllano la composizione dello stato corrente, l'ordinamento e la capacità finita degli eventi e la serializzazione dei messaggi esposti alla WebUI; verificano inoltre che `TelemetryPump` non raccolga nuovi dati quando il trasporto non è disponibile o applica *backpressure*. Queste prove verificano che le politiche di ordinamento, limite e sostituzione possano essere valutate in modo deterministico.

La verifica dell'AFSM segue la topologia della Figura 3.2 e la matrice di riferimento descritta nel capitolo precedente. Per ciascuna transizione viene verificata la condizione sia quando i dati la autorizzano sia quando una configurazione, una versione o un'osservazione la rendono non valida; vengono inoltre controllati gli eventi duplicati, le combinazioni stato evento non ammesse, la progressione di `mode_generation` e il recupero da *ControlInhibited* e *FaultLatched*.

5.3.2 Verifica sul target ESP32-S3

Il test sul target si pone come obiettivo quello di verificare il comportamento del sistema aggiungendo dinamiche che non possono essere replicate nei test eseguiti su host. All'avvio, il firmware deve creare i task, inizializzare le periferiche e mantenere gli interlock fino a quando configurazione, informazioni obbligatorie e sincronizzazione non risultano disponibili. La console seriale consente di seguire questa progressione, ma la sola sequenza dei messaggi non dimostra la core affinity o la durata delle ISR.

La prova sul target si concentra sui confini nei quali il runtime può alterare il comportamento del dominio. Rientrano in questo insieme il trasferimento dall'ISR al task, la capacità finita delle queue, la lettura coerente delle istantanee, la programmazione dei timer e la cancellazione delle azioni quando la modalità diventa più restrittiva. Verificare separatamente ogni primitiva FreeRTOS produrrebbe molto dettaglio senza rafforzare la conclusione; viene invece osservato se la politica documentata rimanga valida quando quelle primitive operano insieme.

5.4 Validazione della PCB

5.4.1 Ispezione e prima alimentazione

Prima di effettuare test del firmware con ingressi ed uscite reali, la scheda deve essere verificata nei percorsi dei segnali e degli attuatori. Questa sequenza separa i difetti di assemblaggio dagli errori del firmware. La prima ispezione riguarda la misura della tensione, osservando assorbimento, rail a 5 V e rail a 3.3 V prima di collegare carichi esterni.

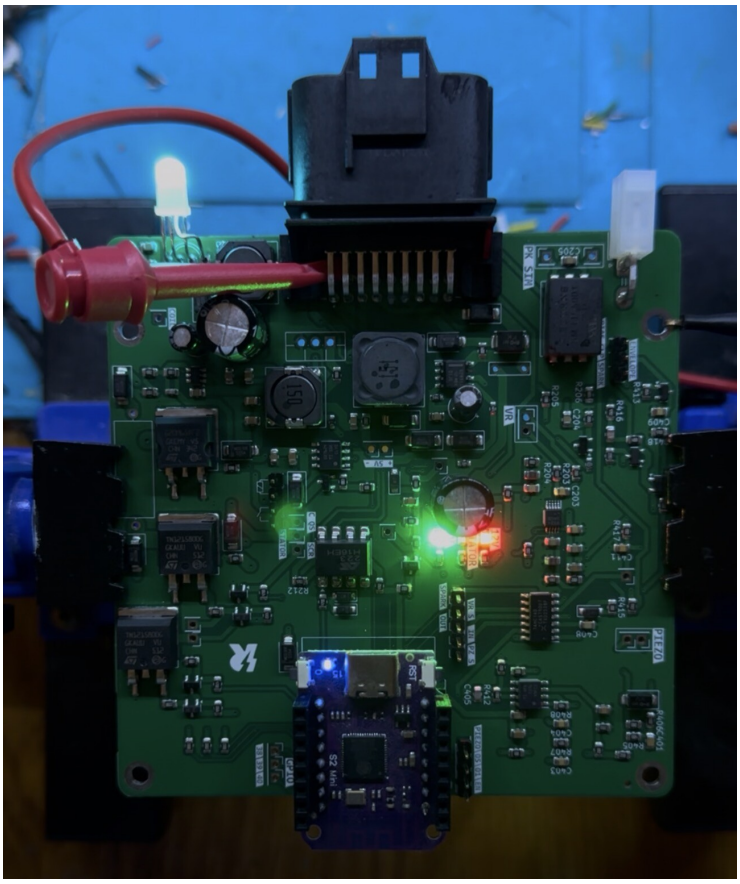

Figura 5.2: PCB `v1.0` con componenti montati e LED di stato accesi.

5.4.2 Verifica degli ingressi

La prima verifica riguarda il pick-up, in quanto bisognava accertarsi che sia il circuito integrato che la lettura dell'ESP32-S3 funzionassero correttamente. Per testare l'integrato è stato utilizzato un sensore pick-up commerciale con un volano magnetico collegato al banco prova. La Figura 5.3 mostra le forme

d'onda osservate sull'oscilloscopio del segnale in uscita dal pick-up e del segnale condizionato in ingresso al microcontrollore.

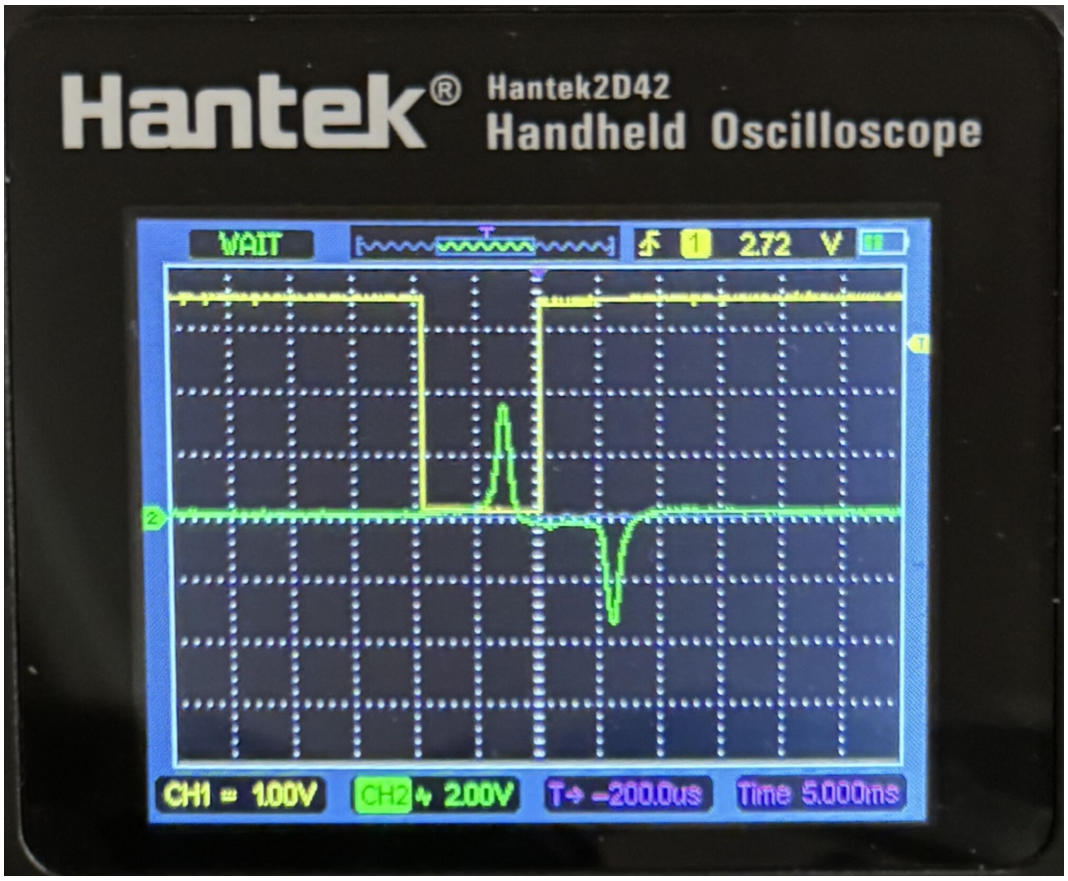

Figura 5.3: Condizionamento del segnale di pick-up: stimolo applicato e uscita condizionata sulla stessa base temporale, con ampiezza, frequenza e soglia indicate.

Il TPS e gli ingressi digitali richiedono criteri differenti. Per il TPS la tensione applicata è stata confrontata con il valore letto e convertito, senza ignorare calibrazione, saturazione e tolleranza. Per il quick-shifter e il map switch sono stati verificati i collegamenti elettrici e il circuito RC di debounce. I test di EGT, temperatura dell'acqua e knock rimangono esclusi dalla validazione elettrica dell'attuale revisione.

5.4.3 Verifica delle uscite

Le uscite sono state testate con carichi che hanno riprodotto le condizioni elettriche necessarie alla prova, senza introdurre i rischi di un attuatore reale.

Sono stati verificati sia i circuiti di pilotaggio sia quelli di potenza. Questi ultimi comprendono la scarica del condensatore per l'accensione della scintilla, il power jet e la valvola di scarico in modalità peak and hold.

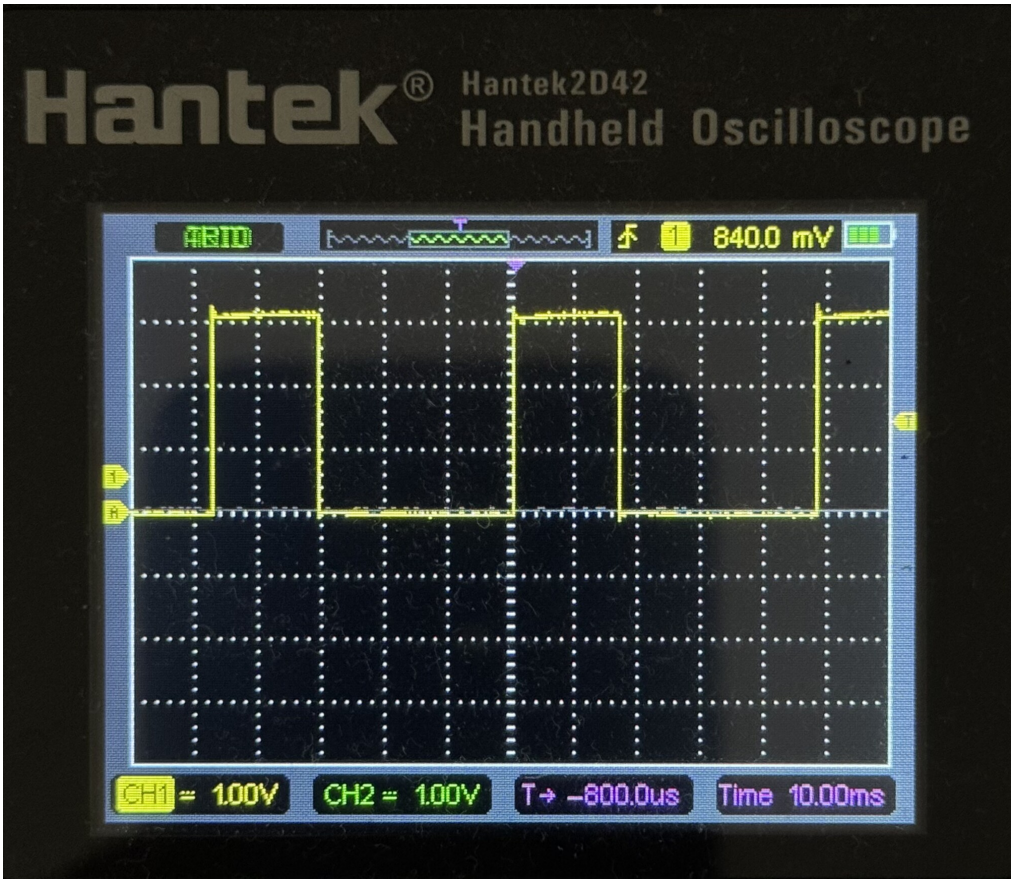

Figura 5.4: Modalità peak and hold del power jet con duty cycle al 30%.

5.5 Validazione dell'ECU

5.5.1 Configurazione del banco per i test end-to-end

I test end-to-end hanno lo scopo di verificare il comportamento dell'ECU in una configurazione reale, osservando congiuntamente l'acquisizione degli ingressi e il comando delle uscite. L'ECU è stata collegata ad una bobina di accensione con la relativa candela, un potenziometro per simulare la posizione del TPS e un solenoide che riproduce il carico del power jet. Questa configurazione permette di esercitare la catena di controllo senza installare l'ECU su un

motore reale e rende direttamente osservabili gli effetti dei comandi generati dal firmware.

La fotografia del banco documenta la configurazione fisica utilizzata e consente di identificare l'ECU, i dispositivi collegati e la strumentazione di misura. I risultati riportati nelle prove successive sono pertanto riferiti esclusivamente a tale configurazione.

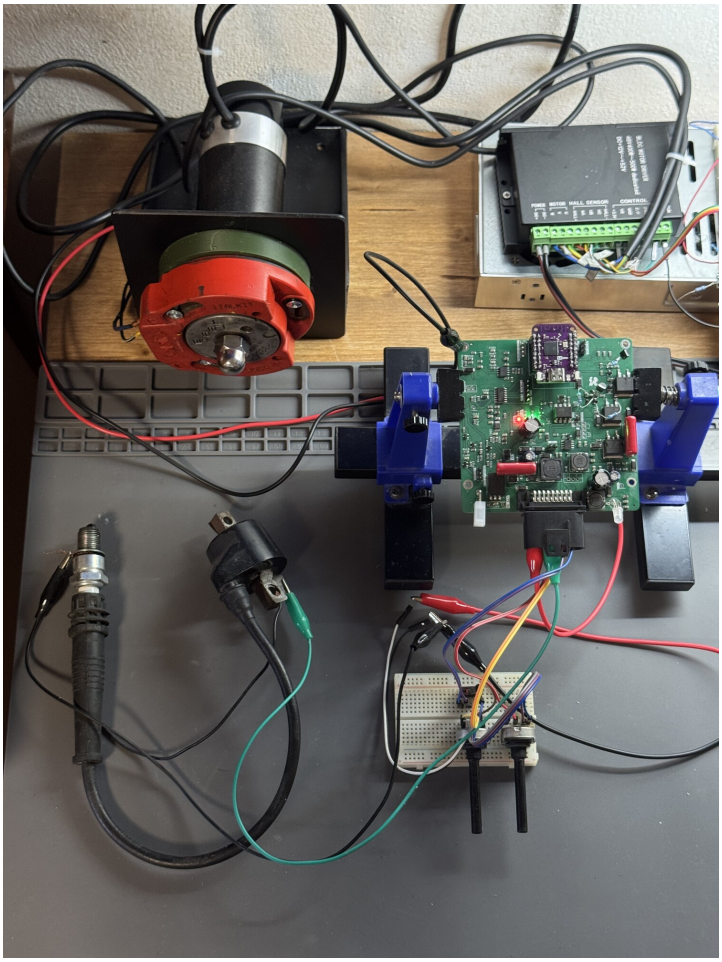

Figura 5.5: Banco prova per i test end-to-end dell'ECU, con bobina e candela di accensione, potenziometro per la simulazione del TPS e solenoide impiegato come carico del power jet.

5.5.2 Test della CDI

La prova della CDI verifica il funzionamento della catena di accensione, dalla carica del condensatore attraverso lo statore del banco prova, passando per

il comando generato dall'ECU fino alla scarica dell'alta tensione sul circuito primario della bobina. La bobina e la candela collegate al banco consentono di osservare anche l'effetto finale del comando, mentre l'oscilloscopio permette di rilevare l'andamento elettrico della scarica. La forma d'onda della Figura 5.6 documenta l'avvenuta scarica del condensatore misurata all'uscita dell'ECU e risulta particolarmente interessante perché dimostra come, nelle centraline CDI, la tensione sul primario della bobina risulti essere negativa rispetto alla massa.

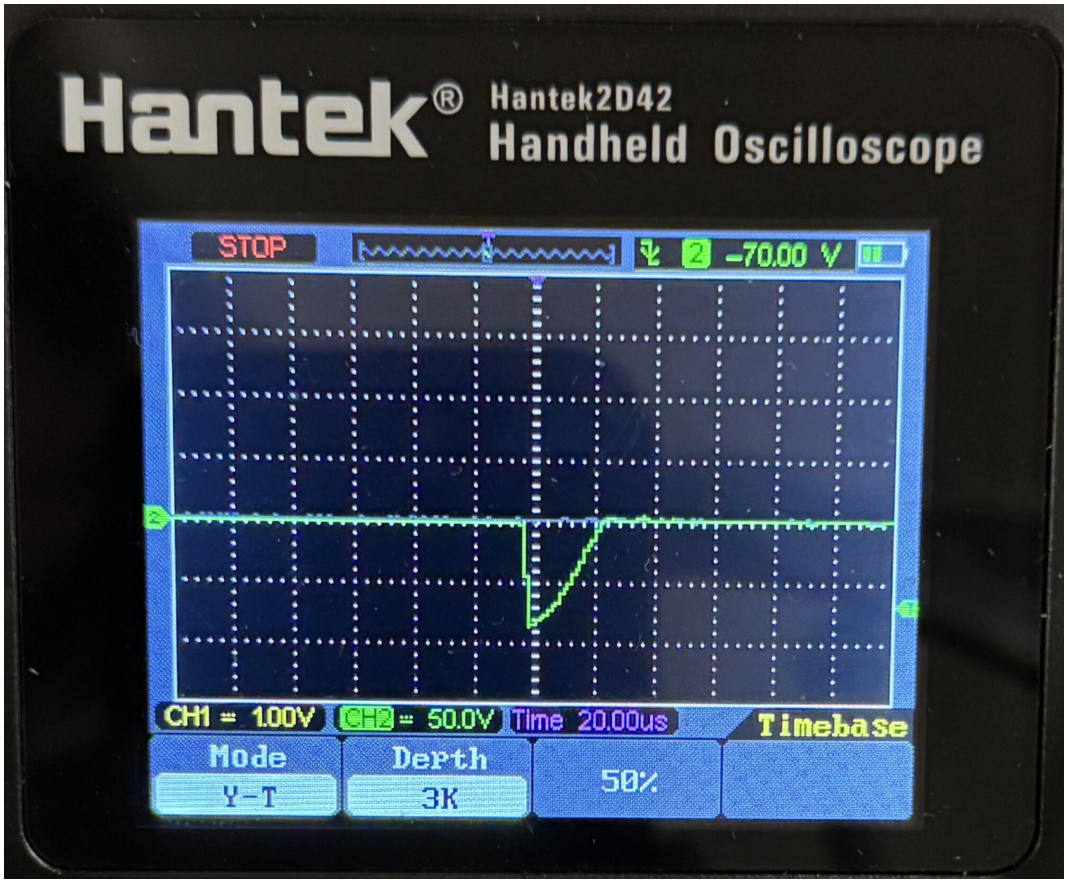

Figura 5.6: Forma d'onda acquisita durante la scarica del condensatore della CDI.

5.6 Verifica esplorativa della WebUI e del Digital Twin

5.6.1 WebUI e collegamento locale

La WebUI appartiene al percorso di supervisione rappresentato nella Figura 3.10 e deve essere valutata senza trasformarla in una dipendenza del controllo. I test condotti riguardano la comunicazione websocket, la rappresentazione usata dalle pagine di telemetria, lo storico locale e il client del Digital Twin. Accanto ai test automatici, l'uso nel browser può mostrare la ricezione della telemetria, l'aggiornamento dei valori nelle pagine e la gestione di una disconnessione.

La priorità rimane l'indipendenza dell'ECU. La chiusura del browser o la perdita del collegamento col server possono rendere non aggiornate le informazioni mostrate e interrompere la registrazione, ma non devono modificare la configurazione già attiva né bloccare l'interfaccia.

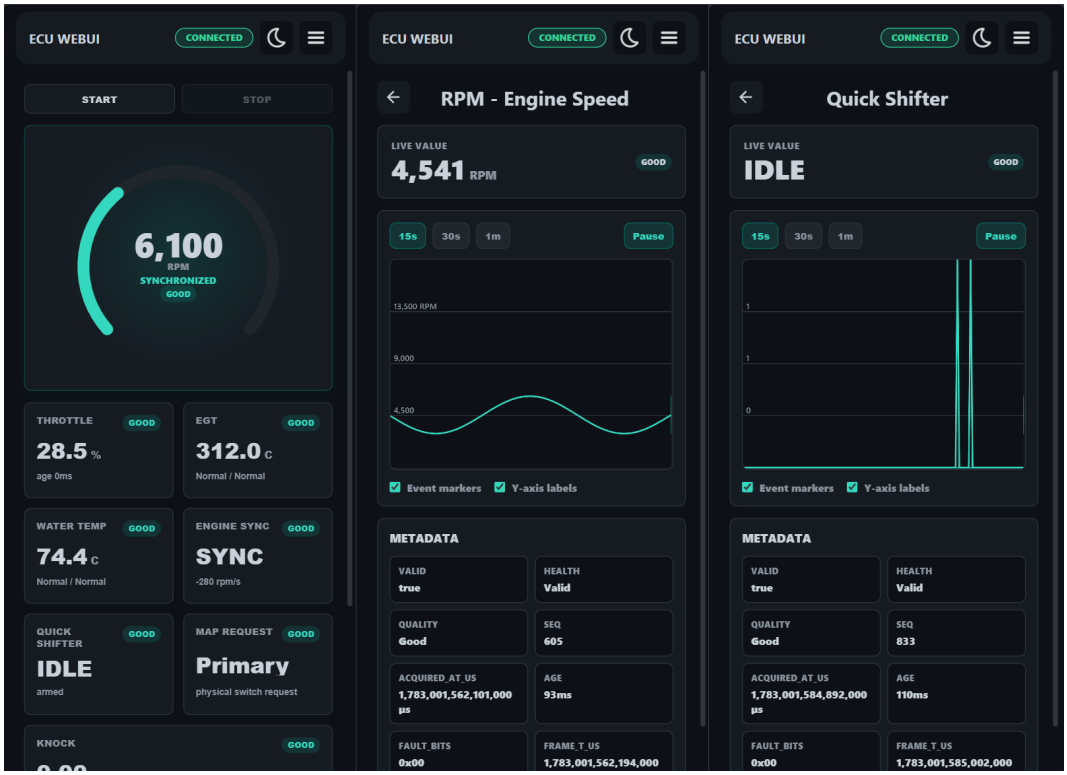

Figura 5.7: Pagine principali della webui.

5.6.2 Digital Twin

Il Digital Twin riceve informazioni attraverso il client e ne conserva un suo stato seguendo le definizioni della Figura 3.11, associandolo alla specifica ECU. I test eseguiti hanno verificato sia il salvataggio dei dati ricevuti sia la gestione concorrente di più ECU che trasmettono dati contemporaneamente. In particolare, i record sono stati correttamente associati all'identificativo dell'ECU di origine e mantenuti separati durante la memorizzazione e la successiva consultazione, senza sovrascritture o commistioni tra i diversi flussi.

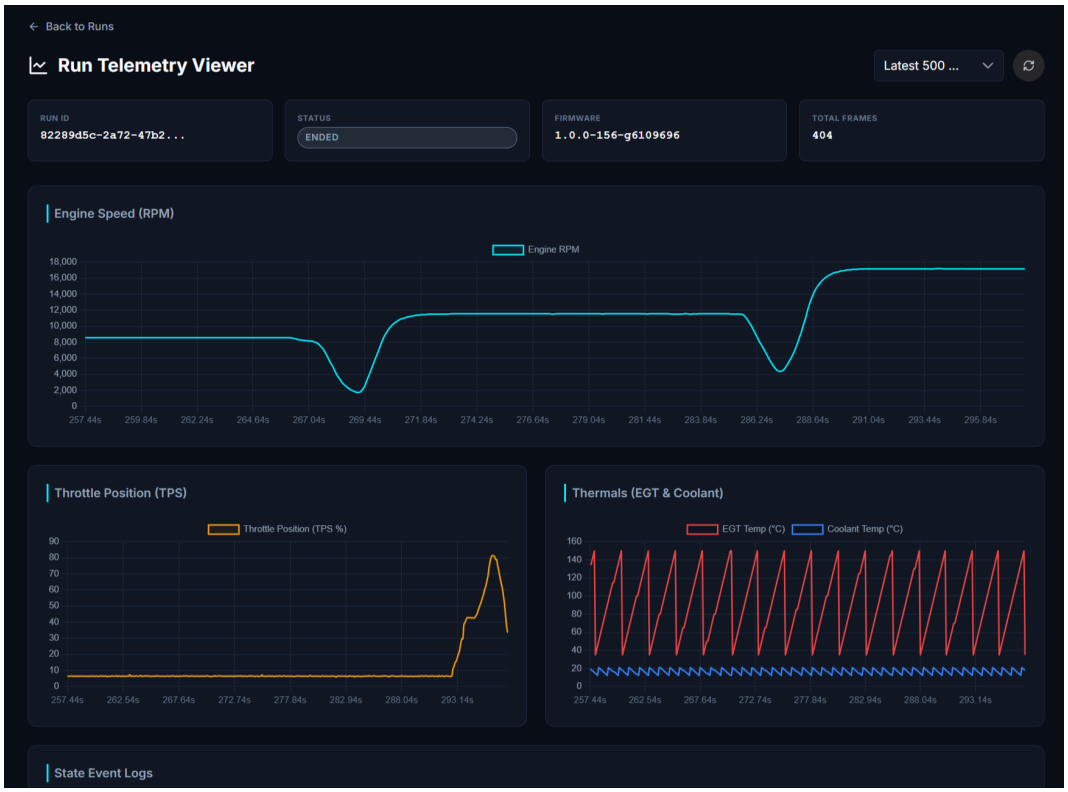

Figura 5.8: Sessione registrata nel Digital Twin con identificativo dell'ECU.

5.7 Difetti riscontrati e limiti dei risultati

Le prove descritte in questo capitolo non si sono concluse tutte al primo tentativo, né sono state tutte superate. Alcuni test hanno fatto emergere dei difetti di progettazione hardware, mentre altri hanno evidenziato problemi minori di implementazione software. Questa sezione raccoglie tali osservazioni, così che le affermazioni dei capitoli precedenti possano essere valutate di conseguenza.

5.7.1 Circuito CDI

Il problema più rilevante è emerso durante il collaudo delle uscite. Il circuito di pilotaggio della CDI non rispondeva ai comandi generati dal firmware e l'analisi del circuito ha ricondotto il malfunzionamento a un errore nello schema elettrico: l'assegnazione dei pin del transistor PNP e i valori delle resistenze di base e di pull-up associate a `Q501` e `Q502` non erano conformi alle specifiche. La correzione è stata dapprima applicata a mano sulla scheda, mediante alcune saldature che hanno permesso di proseguire con le prove di accensione, e successivamente riportata nello schema elettrico in vista della revisione successiva.

5.7.2 Alimentazione batteryless

Il convertitore flyback configurato per ricavare energia dallo statore non ha funzionato adeguatamente, quindi tutte le prove descritte hanno richiesto l'alimentazione della scheda attraverso il connettore a 12 V. L'origine del problema non è stata ancora individuata nonostante siano state effettuate diverse prove e misurazioni.

5.7.3 Limiti della configurazione provata

Accanto al difetto corretto rimangono le assenze, che non derivano da errori ma dalle scelte con cui la prima revisione è stata realizzata: i circuiti per EGT, knock sensor e temperatura dell'acqua non sono presenti sulla scheda realizzata, e di questi ingressi è stato possibile verificare soltanto il dominio software mediante emulazione, come anticipato nella Sezione 5.1. Il prototipo copre dunque i soli percorsi di ingresso e uscita descritti nelle sezioni precedenti, e ogni conclusione tratta in questo capitolo vale entro tale perimetro.

Va infine ricordato che la scheda rimane un prototipo di sviluppo e che tutte le prove si sono svolte al banco. Le verifiche ambientali, EMC e sugli standard automotive, insieme alle prove di affidabilità non fanno parte dello scopo dei test descritti.

Conclusioni

Il lavoro presentato in questa tesi ha affrontato la distanza che separa l'elettronica di controllo dei motori monocilindrici a combustione interna dalle piattaforme software che caratterizzano le ECU automotive moderne. I risultati ottenuti indicano che tale distanza non dipende da un limite tecnologico intrinseco: un'architettura *software-defined*, affiancata da un Digital Twin server-side, può essere realizzata su un SoC comunemente reperibile rispettando i vincoli temporali imposti dal controllo di un motore a combustione interna, a condizione che la separazione tra percorso critico e servizi di supporto venga trattata come un vincolo di progetto e non come un'ottimizzazione successiva. Il percorso svolto ha coperto l'intero arco dello sviluppo, dall'analisi dei requisiti alla definizione dell'architettura, dall'implementazione del firmware alla progettazione della PCB, fino alla costruzione di un banco prova dedicato e alla definizione di una metodologia di validazione articolata su più livelli.

Una prima conclusione, di natura sia architetturale sia metodologica, riguarda la praticabilità dei principi adottati. La proprietà univoca dello stato, la qualificazione di ogni lettura dei sensori mediante metadati e la separazione tra logica di dominio e di piattaforma hanno trovato realizzazione concreta sull'ESP32-S3 e hanno restituito ciascuna un beneficio distinto: la prima ha semplificato il ragionamento sul comportamento del sistema, poiché ogni informazione ha un solo proprietario e ogni anomalia un percorso di propagazione definito; la seconda ha reso possibili funzioni come la degradazione controllata, poiché l'AFSM stabilisce la modalità operativa a partire da dati verificati e il controllo può rifiutare piani preparati in un contesto ormai superato; la terza ha permesso di esercitare su host una parte significativa del comportamento dell'ECU, strutturando la campagna di verifica su livelli distinti e rendendo ogni errore attribuibile al livello che lo produce. La protezione del motore e la testabilità non risultano pertanto funzioni aggiunte al sistema, ma proprietà che emergono dalla struttura delle informazioni scambiate e dei confini tracciati tra i sottosistemi.

Sul piano sperimentale, il prototipo finale ha consentito di convalidare l'ac-

censione della scintilla in condizioni controllate e ripetibili. Il banco prova realizzato appositamente ha permesso di sottoporre l'ECU a segnali specifici e modificabili in modo indipendente. Sia il circuito di condizionamento del pick-up sia la gestione degli attuatori ausiliari si sono dimostrati robusti e affidabili, rendendoli riutilizzabili nella progettazione di future revisioni della PCB.

Sviluppi futuri

L'evoluzione naturale del progetto riguarda in primo luogo l'hardware. Una revisione successiva della PCB dovrà integrare i sensori rinviati dalla prima versione, adottare il driver `DRV8874` per gestire tutti i tipi di valvola di scarico, correggere in modo definitivo la sezione CDI incorporando le correzioni già applicate e rendere funzionale il circuito di alimentazione direttamente dallo statore, così da permettere l'applicazione del prototipo su motori sprovvisti di batteria.

Il Digital Twin, infine, costituisce la direzione di sviluppo con il maggiore potenziale. La sua configurazione attuale conserva sessioni, revisioni e risultati mantenendone la provenienza; su questa base potranno essere introdotte funzionalità come le stime termiche, la valutazione del rischio di detonazione, l'analisi del degrado degli attuatori e la diagnostica predittiva. Inoltre, l'aggiunta di aggiornamenti OTA autenticati integrati con la gestione delle versioni trasformerebbe il prototipo in una piattaforma completa per lo sviluppo e la calibrazione di strategie di controllo.

Nel complesso, il progetto costituisce una base tecnica estendibile che introduce osservabilità, configurabilità e tracciabilità in un dominio che ne era rimasto privo. Il valore del lavoro risiede nel metodo prima ancora che nelle singole funzioni: confini espliciti tra i sottosistemi, dati accompagnati dalla propria provenienza e una validazione che dichiara le condizioni entro le quali i risultati conservano validità. È questa impostazione, più di ogni specifica caratteristica del prototipo, a rendere il sistema una piattaforma sulla quale le funzionalità mancanti possono essere aggiunte senza rimettere in discussione la struttura che le ospita.

Ringraziamenti

Desidero ringraziare innanzitutto i miei genitori, per avermi sostenuto durante tutto il percorso di studi e per avermi dato la possibilità di affrontarlo con serenità e determinazione.

Ringrazio mio fratello, le cui competenze e il cui esempio hanno rappresentato per me una preziosa fonte di ispirazione.

Un ringraziamento speciale va a mia nonna, per il costante sostegno morale, l'affetto e l'incoraggiamento che non mi ha mai fatto mancare.

Ringrazio sinceramente il mio relatore per la disponibilità, l'aiuto e la tempestività con cui mi ha seguito durante la realizzazione di questo lavoro.

Desidero inoltre ringraziare il mio correlatore per la passione che ha saputo trasmettermi e per le competenze tecniche condivise, fondamentali per lo sviluppo di questa tesi.

Infine, ringrazio i miei amici, per aver condiviso con me questo percorso e per essere stati una presenza importante nei momenti di studio, nelle difficoltà e nei traguardi raggiunti.

Bibliografia

[1] AiM Technologies. Motorbike data logging systems. https://www.aimtechnologies.com/products/bike/, 2026. Sistemi di data logging motociclistico, dashboard configurabili e acquisizione dati.

[2] AUTOSAR. Autosar platforms. https://www.autosar.org/, 2026. AUTOSAR Classic Platform e AUTOSAR Adaptive Platform.

[3] Bosch Mobility. E/e architecture. https://www.bosch-mobility.com/en/mobility-topics/ee-architecture/, 2026. Evoluzione delle architetture elettroniche automotive verso strutture centralizzate e zonali.

[4] Espressif Systems. Esp-idf programming guide v5.5.4 for esp32-s3. https://docs.espressif.com/projects/esp-idf/en/v5.5.4/esp32s3/index.html, 2026. Documentazione ufficiale di ESP-IDF per ESP32-S3.

[5] Espressif Systems. Freertos (idf) for esp32-s3. https://docs.espressif.com/projects/esp-idf/en/v5.5.4/esp32s3/api-reference/system/freertos_idf.html, 2026. Documentazione ufficiale dell'implementazione dual-core di FreeRTOS in ESP-IDF.

[6] Mectronik. Motorsport electronics. https://www.mectronik.com/motorsport/, 2026. ECU motorsport, acquisizione dati, calibrazione e comunicazione.

[7] NXP Semiconductors. Software-defined vehicle. https://www.nxp.com/applications/automotive/software-defined-vehicle:SOFTWARE-DEFINED-CARS, 2026. Siattaforme SDV, central compute, zonal controller, aggiornamenti software e digital twin.

[8] SparkEVO. itronix ecu. https://sparkevo.racing/products/itronix-standalone/, 2026. ECU per motori 2T e 4T con mappe, power valve, sensori, datalogger e configurazione da PC o smartphone.

[9] TEXA. Navigator txb 2. https://www.texa.it/prodotti/navigator-txb-2/, 2026. Strumenti diagnostici nel settore bike e marine, supporto CAN FD, PassThru e funzioni di configurazione.

[10] Vector Informatik. Software-defined vehicle. https://www.vector.com/int/en/products/solutions/software-defined-vehicle/, 2026. Software-defined vehicle e architetture automotive moderne.

[11] Vortex CDI. Vortex ecu products. https://www.vortexcdi.com/product_generic.php?cid=3, 2026. ECU aftermarket configurabile.

[12] WEMOS. S3 mini. https://www.wemos.cc/en/latest/s3/s3_mini.html, 2024. Documentazione ufficiale del modulo basato su ESP32-S3FH4R2.