Politecnico di Torino

Corso di Laurea Magistrale in Ingegneria Meccanica

A.a. 2024/2025

Sessione di Laurea Ottobre 2025

Approccio MBSE all'Integrazione tra Cameo Systems Modeler e MATLAB Simulink per sistemi complessi

Relatore:
Eugenio BRUSA

Tutor aziendale:
Setareh SAREMI

Candidato:
Leranto GOXHAJ

Index

Abstract

Nel contesto dell'ingegneria dei sistemi moderna, la crescente complessità dei prodotti tecnologici, l'interconnessione tra sottosistemi eterogenei e la necessità di garantire coerenza, tracciabilità e interoperabilità lungo l'intero ciclo di vita di un sistema hanno reso indispensabile l'adozione di approcci metodologici strutturati e integrati. In risposta a queste esigenze, l'ingegneria dei sistemi basata su modelli (Model-Based Systems Engineering, MBSE) si è affermata come una delle strategie più efficaci per affrontare la progettazione, lo sviluppo e la gestione di sistemi complessi e multidisciplinari. L'MBSE consente di sostituire la documentazione tradizionale con modelli digitali formali, che rappresentano in modo coerente e tracciabile gli aspetti funzionali, comportamentali e strutturali del sistema, favorendo una visione sistemica e condivisa tra tutti gli stakeholder coinvolti. [1]

All'interno di questo paradigma, SysML, ovvero un linguaggio di modellazione visuale pensato per l'ingegneria dei sistemi, consente di descrivere in modo rigoroso e multidisciplinare requisiti, architettura e comportamenti, facilitando la comunicazione tra stakeholder e supportando, tramite strumenti come Cameo Systems Modeler, l'analisi, la simulazione e la validazione del sistema. [19]

Il presente lavoro di tesi si inserisce nel contesto metodologico dell'ingegneria dei sistemi basata su modelli (MBSE) e nasce da un'esperienza di tirocinio svolta presso Capgemini Engineering, azienda leader nel settore della consulenza tecnologica e dell'ingegneria dei sistemi. Nella fase iniziale del tirocinio è stato affrontato un percorso formativo intensivo, finalizzato all'acquisizione delle competenze teoriche e pratiche necessarie per operare efficacemente in ambito MBSE; in particolare, sono stati approfonditi i principi fondamentali della Systems Engineering, le fasi del ciclo di vita di un sistema secondo la norma ISO/IEC 15288, i concetti di tracciabilità e gestione dei requisiti, nonché l'utilizzo del linguaggio SysML e degli strumenti software dedicati alla modellazione.

Questa fase formativa ha costituito la base metodologica su cui si è sviluppato il progetto di tesi, e ha trovato una sua naturale estensione nella prima parte del documento, dedicata all'analisi teorica dei fondamenti dell'ingegneria dei sistemi, ingegneria dei requisiti, dell'approccio MBSE e del linguaggio SysML; tale sezione ha lo scopo di fornire al lettore una panoramica strutturata e coerente dei concetti chiave che guidano la modellazione e la simulazione di sistemi complessi.

A partire da questa base, è stato definito il caso d'uso oggetto dello studio: la modellazione e simulazione di un sistema per il rilevamento e l'evitamento degli ostacoli, fondamentale per garantire l'autonomia e la sicurezza di un drone intelligente. Durante il tirocinio, il caso è stato analizzato in collaborazione con i team Capgemini, approfondendo il funzionamento del sistema e le sue logiche operative, e dedicando nella tesi una sezione specifica all'introduzione del drone e all'analisi del contesto tecnico.

Dopo aver definito il caso d'uso, è stato avviato un processo di modellazione che ha coinvolto due ambienti complementari: Cameo Systems Modeler, per la definizione dell'architettura funzionale e logica del sistema, e MATLAB Simulink, per la simulazione dinamica dei sottosistemi. L'obiettivo era costruire un modello integrato in grado di rappresentare coerentemente il comportamento del sistema e verificarne le funzionalità attraverso simulazioni eseguibili.

In particolare, è stato sviluppato un modello architetturale in Cameo Systems Modeler che rappresenta la struttura logica e funzionale del sistema, affiancato da un modello comportamentale in MATLAB Simulink capace di simularne la dinamica, con l'obiettivo di integrare i due ambienti in modo coerente e tracciabile. L'integrazione si è basata sull'identificazione delle variabili necessarie alla simulazione, analizzandone la struttura e la funzione, per poi riprodurle all'interno del modello Cameo e associarle ai sottosistemi di riferimento, così da garantire una corrispondenza tra architettura e comportamento.

Per permettere lo scambio di informazioni tra i sottosistemi è stato realizzato un diagramma interno che, attraverso connettori, ha messo in comunicazione le diverse parti del sistema, assicurando che ogni modello comportamentale ricevesse le variabili necessarie al proprio funzionamento. A supporto della cosimulazione sono stati creati diagrammi di attività che gestiscono l'intero flusso operativo, dalla ricezione delle variabili alla loro conversione per Simulink, fino all'attivazione del modello comportamentale, alla raccolta degli output e alla loro trasmissione al modulo successivo, delineando un processo in cui Cameo assume il ruolo di orchestratore del sistema mentre Simulink ne rappresenta il motore di simulazione.

Il percorso svolto ha permesso di approfondire le dinamiche di integrazione tra modellazione architetturale e simulazione comportamentale, evidenziando come l'interoperabilità tra Cameo Systems Modeler e MATLAB Simulink possa offrire un valido supporto alla progettazione di sistemi autonomi complessi. Nonostante le limitazioni riscontrate in termini di prestazioni e gestione delle risorse, l'approccio adottato ha dimostrato una forte efficacia dal punto di vista metodologico, consentendo di rappresentare in modo coerente l'architettura del sistema e di verificarne il comportamento attraverso simulazioni eseguibili.

CAPITOLO 1 Systems Engineering 1

L'ingegneria dei sistemi è un approccio multidisciplinare integrato per la realizzazione di un sistema ingegnerizzato considerando l'intero ciclo di vita di quest'ultimo ovvero il concetto, lo sviluppo, la produzione, l'utilizzo, il supporto tecnico e il pensionamento, utilizzando principi e concetti dei sistemi, metodi scientifici, tecnologici e di gestione. Bilanciando prestazioni, costi, tempi e rischi, l'ingegneria dei sistemi assicura che il prodotto finale non sia solo tecnicamente valido, ma anche in linea con le esigenze degli utenti e con gli obiettivi dell'organizzazione; infatti, l'obiettivo finale è fornire una soluzione di alta qualità, affidabile e sostenibile che soddisfi o superi le aspettative dei clienti. [1]

L'ingegneria dei sistemi si concentra sull'identificazione e la definizione delle esigenze del cliente e delle funzionalità richieste nelle prime fasi del ciclo di sviluppo, questo include la documentazione approfondita dei requisiti, seguita dalla sintesi strutturata dei progetti di sistema e da processi di convalida rigorosi. Durante il suo ciclo di vita, l'ingegneria dei sistemi considera l'intero spazio problematico, non solo aspetti tecnici isolati, ma anche obiettivi aziendali, ambienti operativi e aspettative delle parti interessate. [1, 2]

1.1 Che cos'è un sistema?

L'ingegneria dei sistemi si occupa dell'implementazione e della gestione di sistemi complessi, per questo motivo, è fondamentale definire cosa si intende per "sistema"; un sistema è un insieme di componenti interagenti che collaborano per svolgere una funzione comune, non realizzabile da ciascun componente singolarmente; questi componenti possono appartenere a diverse discipline tecnologiche, come meccanica, elettronica, informatica, ecc.[1] Tuttavia, non ogni aggregato di elementi può essere considerato un sistema: è necessario che tra essi esista una interazione funzionale, fisica o logica, che ne giustifichi l'integrazione. [1,3] Secondo lo standard ISO/IEC/IEEE 15288, *un sistema può includere elementi artificiali e naturali, e deve essere progettato considerando l'intero ciclo di vita, dalla concezione al pensionamento* [2]. Un sistema può essere definito come:

- "*Un insieme di elementi in interazione*". (von Bertalanffy 1968) [33]
- "*Combinazione di elementi interattivi organizzati per raggiungere uno o più scopi dichiarati (ISO/IEC/IEEE 15288:2015)*" [2]
- "*Un sistema è un insieme di parti o elementi che insieme mostrano comportamenti o significati che i singoli costituenti non hanno*". (INCOSE Fellows, 2019) [12]

Da queste definizioni, si può dedurre che un sistema deve avere almeno due componenti che interagiscono tra loro; inoltre, deve essere delimitato da un 'confine' reale o astratto che lo separa dall'ambiente esterno con cui interagisce (Fig.1). Si può dire che un sistema, nel suo ciclo di vita, trasforma uno o più ingressi in uno o più uscite utilizzando i suoi componenti; pertanto, un sistema

può essere definito come un insieme ben definito di componenti che interagiscono tra loro con un comune scopo dichiarato [1].

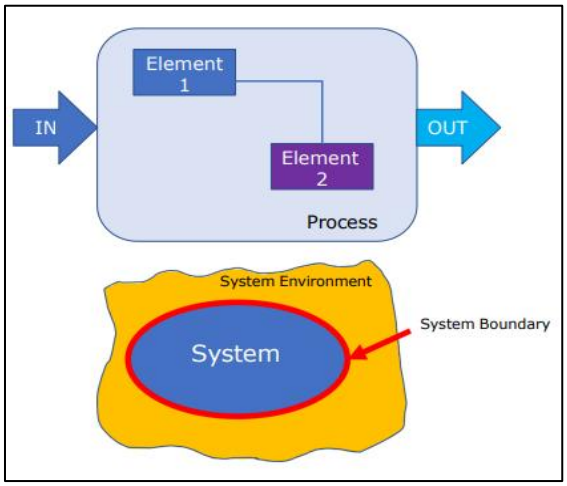

Fig.1 System's boundary and I/O [4]

1.2 Attività e punti cardine

Nella seguente sezione verranno definite quali sono le principali attività svolte dagli ingegneri dei sistemi durante il ciclo di vita del sistema.

Innanzitutto, si parte effettuando una business analysis, fase cruciale nell'ingegneria dei sistemi, in cui vengono comprese le esigenze del cliente e si definiscono gli obiettivi principali del sistema; si prosegue poi nel campo di competenza (specializzazione) ingegneristica, dove le competenze tecniche vengono applicate per progettare sistemi efficaci; inoltre, si considera una gestione tecnica del progetto, in cui si garantisce l'organizzazione, coordinazione, monitoraggio e adempimento delle attività necessarie nei tempi e nei costi previsti (fig.2) [3]. Si noti che queste attività sono in continua interazione tra loro e scambiano informazioni anche con l'ambiente esterno; questo approccio sistemico è ciò che permette di affrontare le complessità in modo strutturato ed efficace. [3]

Inoltre, si possono considerare tre pilastri principali su cui l'Ingegneria dei Sistemi si fonda [1,3]:

1. People, persone che rendono possibile la System Engineering (ingegneri dei sistemi, esperti del settore, utenti, ecc.) portano conoscenze specifiche, esperienza e intuizioni che nessun strumento o processo può eguagliare; tuttavia, le persone senza un processo chiaro rischiano di agire in modo disorganizzato e inefficace. (Fig.3) [1]
2. Tools, aumentano l'efficienza dei processi previsti in System Engineering (ne costituiscono il mezzo), tuttavia senza il giusto criterio risultano inutili. (Fig.3)[1]
3. Process, senza persone che lo comprendano e lo applichino rimane un'astrazione. (Fig.3)[1]

Come si nota questi cardini sono tutti fondamentali ed è altrettanto fondamentale il legame tra di essi, solo integrando questi elementi tra loro diventa possibile progettare sistemi complessi che siano davvero complessi, efficaci e orientati al valore. [1]

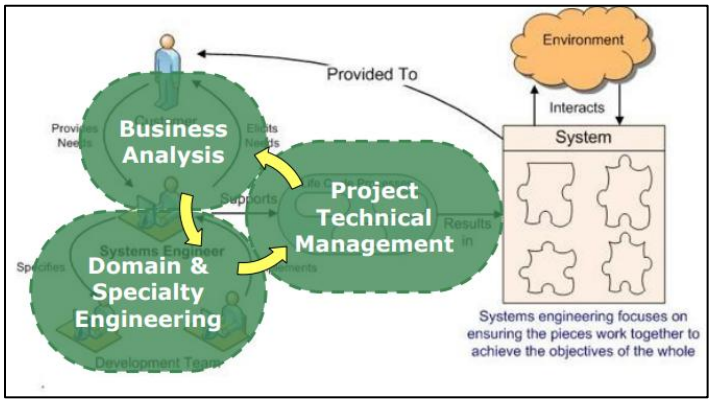

Fig.2 Systems Engineering Activities [3]

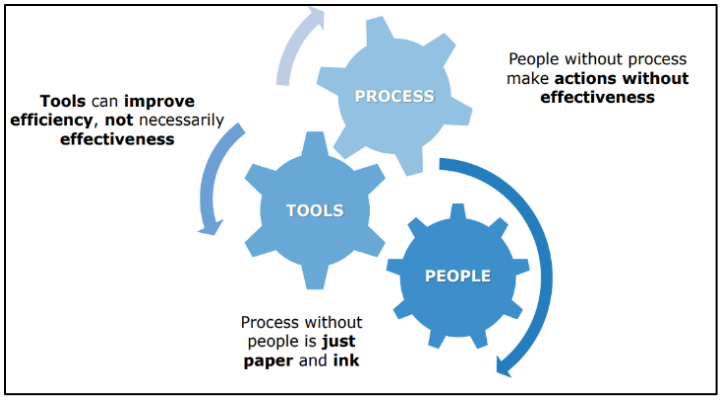

Fig.3 main pillar of System Engineering [4]

1.3 Evoluzione nel tempo dell'ingegneria dei sistemi

Si può procedere accennando come si è evoluta la System Engineering nel tempo e cosa è previsto per il futuro [1]. Sostanzialmente possiamo suddividerne l'evoluzione in quattro tappe fondamentali (Fig.4):

- *Classical System Engineering*: partiamo da un approccio classico, fortemente basato sulla documentazione. In questa fase, l'Ingegneria dei Sistemi si concentrava sulla gestione di progetti complessi attraverso una grande quantità di documentazione; ogni fase del ciclo di vita del sistema era tracciata su carta, con processi spesso rigidi e poco integrati; questo

approccio, seppur fondamentale per gettare le basi della disciplina, mostrava grandi limiti in termini di flessibilità e tracciabilità. [3,4]

- *Requirement-Based System Engineering*: successivamente si è passato ad un approccio basato sull'analisi di requisiti, ovvero analisi sistematica di obiettivi, funzioni e vincoli del sistema in studio; in questo metodo si introduce un'analisi più strutturata riguardo verifica e validazione (V&V) lungo tutto il ciclo di vita del prodotto. Questo approccio inoltre migliora la tracciabilità e la gestione dei cambiamenti, ma resta ancora fortemente legato a rappresentazioni testuali. [3,4]
- *Model-Based System Engineering (MBSE)*: in questo caso viene introdotta la modellazione come strumento centrale di analisi, dove i modelli diventano il linguaggio comune per descrivere requisiti, architetture logiche, funzionali e fisiche. Questo permette una maggiore coerenza, simulazione anticipata e una gestione più efficacie delle complessità; I modelli vengono sviluppati utilizzando un formalismo rigoroso e comune che permette a tutti i team coinvolti nella realizzazione del modello, così come agli stakeholders interessati, di interpretare le informazioni contenute nel modello nel medesimo modo senza ambiguità. In seguito, approfondiremo meglio il concetto di MBSE che verrà utilizzato anche per questo lavoro di tesi. [1,4]
- *Digital Continuity & Digital Twin*: infine, l'evoluzione ultima della System Engineering mira alla continuità digitale e al concetto di Digital Twin, quando si tratta di modelli diversi che sono integrati e condivisi tra strumenti e piattaforme, l'implementazione di Digital Threads robusti garantisce un flusso coerente e continuo di informazioni autorevoli tra team multidisciplinari che lavorano su diversi aspetti del modello di sistema. La continuità digitale forma la base per l'implementazione dei gemelli digitali, che è la rappresentazione virtuale che rispecchia accuratamente i sistemi del mondo reale; a questo punto l'intero ciclo di vita del sistema (progettazione, produzione...) è integrato in un unico modello digitale; questo consente una visione olistica e in tempo reale del sistema, migliorando la collaborazione, la qualità e la capacità di adattamento; in sintesi l'evoluzione si muove verso un ecosistema completamente connesso e intelligente [3,4]

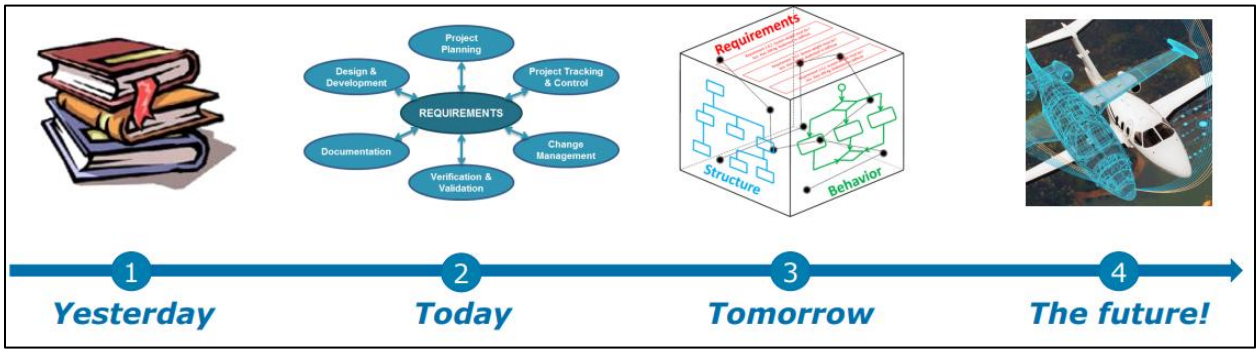

Fig.4 System Engineering time laps [4]

1.4 Lifecycle approaches and models

Nel contesto dell'Ingegneria dei Sistemi, i modelli di sviluppo (ciclo di vita) dei sistemi rappresentano strumenti fondamentali per la pianificazione, la gestione e il controllo di progetti complessi. Questi modelli forniscono una struttura metodologica che guida le attività da svolgere in ciascuna fase del ciclo di vita di un sistema, dalla concezione iniziale fino al pensionamento [1].

Nella seguente figura (Fig.5) sono indicati tra dei modelli più rappresentativi: *Waterfall, Vee* e *Spiral*. Ciascuno di essi riflette un diverso approccio alla gestione della complessità e del rischio. Inoltre, il loro uso dipende anche dal contesto progettuale, dal livello di incertezza e dalla maturità organizzativa [1].

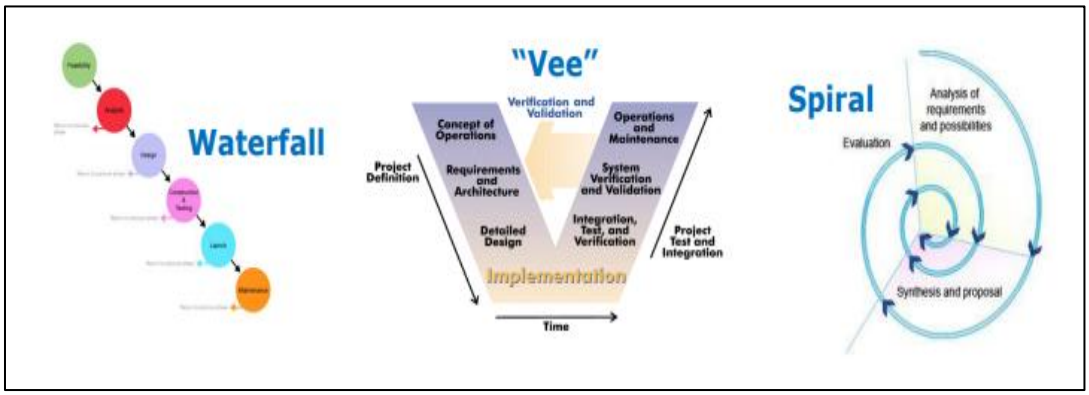

Fig.5 System development life-cycle model, creato nei documenti Capgemini unendo più immagini [4,26,27].

Il modello *Waterfall* (fig.5) è uno dei primi approcci formalizzati in ambito di sviluppo sistemico, esso prevede una sequenza lineare di fasi: concezione, inizio, analisi, progettazione, costruzione, test e manutenzione. [1] Ogni fase deve essere completata prima di passare alla successiva, inoltre questo modello è particolarmente adatto a progetti con requisiti ben definiti e stabili, ma poco flessibile in presenza di cambiamenti o incertezza; tuttavia, alcuni degli svantaggi di questo tipo di approccio sono la sua rigidità e la difficolta di addattamento ai cambiamenti. [1]

Il modello *Spiral* (fig.5) invece rappresenta uno sviluppo che segue un approccio iterativo e incrementale, fortemente orientato alla gestione del rischio; ogni iterazione del ciclo comprende attività di valutazione, analisi di requisiti, progettazione e verifica, con l'obiettivo della continua identificazione e mitigazione dei rischi. Questo modello è particolarmente adatto per progetti innovativi o con elevata incertezza tecnologica, dove la metodologia più logica prevede una esplorazione progressiva di soluzioni varie [1].

Infine, il modello *Vee* (fig.5) presenta una visione più strutturata, in cui le attività di sviluppo (lato sinistro della V) sono strettamente correlate alle attività di verifica e validazione (lato destro). Le fasi del includono la definizione del concetto operativo, la specifica dei requisiti, la progettazione

architetturale e dettagliata, seguite da implementazione, integrazione, verifica e validazione del sistema. Questo modello è ampiamente utilizzato in ambiti ad alta criticità, come aerospazio e difesa, dove la tracciabilità e conformità di requisiti sono essenziali [1].

Pertanto, l'adozione di uno dei modelli precedenti è una decisone strategica che incide profondamente sull'efficacia del progetto; si nota che la conoscenza approfondita di questi modelli consente agli ingegneri di selezionare l'approccio più idoneo in funzione delle caratteristiche da sviluppare, del contesto operativo e degli obiettivi di progetto.

In genere il modello più comunemente usato per rappresentare il processo di sviluppo dei sistemi è il modello a V (V-Model, Vee). Grazie alla struttura simmetrica di questo modello (lato sinistro si trovano le attività di definizione e progettazione, mentre sul lato destro le attività di integrazione verifica e validazione) si è in grado di mantenere una tracciabilità bidirezionale tra requisiti e soluzioni, facilitando la gestione delle complessità e la coerenza tra le diverse fasi del progetto [1]; inoltre, il processo rimane iterativo (come Spiral), permettendo revisioni e miglioramenti continui man mano che si prosegue con la fase di sviluppo e si acquisiscono nuove informazioni o cambiamenti nei requisiti.

1.5 System Thinking

Il system thinking è un modo di analizzare e comprendere la realtà concentrandosi su un insieme specifico e sulle relazioni tra le sue parti interne, piuttosto che dei singoli elementi isolati, inoltre, risulta particolarmente utile quando si ha a che fare con problemi complessi, dove cause ed effetti non sono immediate ed evidenti. [1]

Sulla base di questa definizione possiamo distinguere il system thinker dal linear thinker; il linear thinker si sofferma sulla scomposizione del sistema in più sottosistemi e componenti, concentrandosi sull'analisi di questi ultimi e non sull'interezza del sistema. Si nota che, all'occorrenza di un problema, il linear thinker si focalizza sul curare il sintomo (effetto) del problema piuttosto che investigare sulla causa reale, mentre il sistem thinker si concentra sull'intero sistema e sulle interfacce tra i componenti e l'ambiente circostante [5]. Quando si presentano problemi cerca di trovare la causa reale del problema, che risulterà essere un problema del processo di produzione o funzionamento del sistema e non di un responsabile [5]; inoltre, il sistem thinking, può essere considerato un approccio valido per affrontare anche la complessità del mondo moderno.

La Fig.6 fornisce una rappresentazione visiva e concettuale dei principali elementi che compongono questo approccio, evidenziando le relazioni dinamiche tra concetti, strumenti e applicazioni.[3]

Al centro della mappa troviamo il sistem thinker, ovvero colui che è in grado di "pensare e agire sistemicamente", ponendo l'accento sulla visione d'insieme e sulla comprensione delle interconnessioni. Questo approccio si fonda sull'idea che per comprendere un sistema complesso non sia sufficiente analizzarne le singole parti, ma sia necessario cogliere le relazioni, le interdipendenze e i modelli ricorrenti che lo caratterizzano. [3,6]

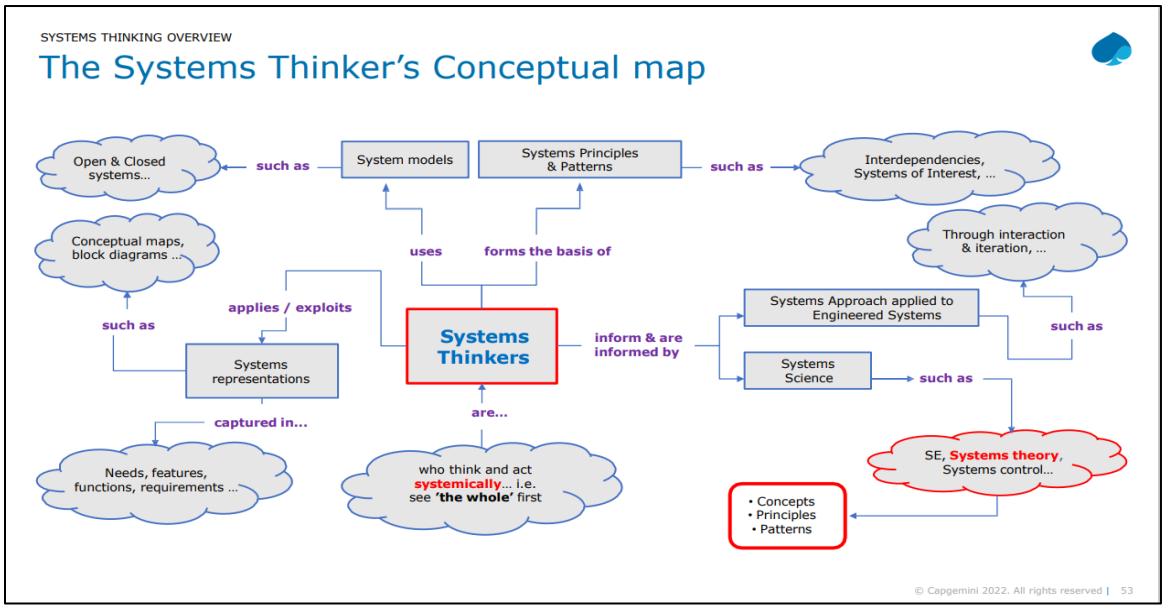

Fig.6 Systems thinker's conceptual map, rielaborata nelle presentazioni Capgemini[3,4].

La mappa distingue tra sistemi aperti e chiusi, concetti fondamentali per comprendere il grado di interazione di un sistema con l'ambiente esterno; a supporto della modellazione e dell'analisi sistemica, vengono utilizzati strumenti come le mappe concettuali, i diagrammi a blocchi e i modelli di sistema, che permettono di rappresentare graficamente le strutture e i flussi informativi. [3,6]

Un altro elemento chiave è rappresentato dai principi e pattern sistemici (fig.6), che costituiscono le regole e le configurazioni ricorrenti osservabili nei sistemi complessi; questi principi sono alla base della scienza dei sistemi e vengono applicati in contesti ingegneristici attraverso la Systems Engineering. [3]

La mappa evidenzia inoltre come il pensiero sistemico sia un processo iterativo e interattivo, in cui la comprensione del sistema evolve nel tempo attraverso l'osservazione, la modellazione e il feedback; le conoscenze acquisite informano e sono a loro volta informate da concetti, principi e pattern, in un ciclo continuo di apprendimento e adattamento. [6,1]

In sintesi, questa mappa concettuale non solo sintetizza i fondamenti teorici del pensiero sistemico, ma ne mostra anche l'applicabilità pratica in ambiti complessi e multidisciplinari. Essa rappresenta uno strumento prezioso per chiunque voglia sviluppare una visione sistemica e affrontare le sfide contemporanee con un approccio integrato e consapevole.

Nel contesto dell'ingegneria dei sistemi e della gestione della complessità, è fondamentale distinguere tra due approcci complementari ma distinti: il systemic thinking e il systematic thinking. La comprensione di queste due modalità di pensiero consente di affrontare i problemi in modo più efficace, integrando visioni diverse ma sinergiche. [7]

Il systemic thinking è considerabile un approccio non riduzionista, che considera il sistema nel suo insieme. Si parte dall'esterno verso l'interno (OUT-IN), cercando di comprendere lo scopo, il contesto e il comportamento globale del sistema; questo approccio è orientato alla sintesi, alla modellazione e all'evoluzione continua del sistema, ponendo l'accento sulle relazioni tra gli elementi e sulla loro capacità di generare comportamenti emergenti. Il systemic thinker è tipicamente transdisciplinare, esplora soluzioni in modo iterativo e ricorsivo, e si concentra sulla visione d'insieme. [1,7]

Al contrario, il systematic thinking adotta un approccio riduzionista, costruendo il sistema dal basso verso l'alto (IN-OUT), ovvero si parte dagli elementi costitutivi, analizzandoli in modo dettagliato e organizzandoli in una struttura gerarchica. Questo approccio è orientato all'analisi, alla prescrizione e alla standardizzazione, di fatto, Il systematic thinker è spesso multidisciplinare, focalizzato sulle parti, e tende a "congelare" le soluzioni una volta definite. [1,7]

L'ingegnere di sistemi efficace risulta colui che riesce a combinare systemic e systematic thinking, integrando la visione globale con l'attenzione ai dettagli; infatti, questa integrazione consente di progettare sistemi complessi che siano non solo funzionali, ma anche adattabili, sostenibili e coerenti con il contesto in cui operano. [1]

Il systems thinking si fonda su alcuni elementi fondamentali che guidano l'osservazione, l'analisi e l'intervento nei sistemi complessi (fig.7) [34]:

- *Multiple prospectives:* ogni sistema coinvolge una molteplicità di attori (stakeholder) che portano con sé visioni del mondo, valori, interessi e conoscenze differenti; riconoscere questa pluralità è essenziale per evitare soluzioni parziali o conflittuali e comprendere le diverse prospettive consente di costruire una visione condivisa del problema e di individuare soluzioni più inclusive e sostenibili.[34]
- *Interconnections:* in un sistema, gli elementi sono legati tra loro da relazioni, scambi, influenze reciproche e cicli di feedback. Analizzare queste connessioni permette di cogliere i pattern ricorrenti e di individuare i punti di leva, ovvero quei nodi strategici in cui un piccolo cambiamento può generare effetti significativi; infatti, è proprio attraverso la comprensione delle interconnessioni che si possono anticipare comportamenti emergenti e agire in modo più efficace. [34]
- *Influences:* ogni sistema è influenzato da forze interne ed esterne: tendenze, vincoli, opportunità, ostacoli. Individuare questi fattori (detti anche driver, enablers o blocks) aiuta a comprendere perché un sistema si comporta in un certo modo e in quale direzione potrebbe evolvere; in particolare, i punti di leva rappresentano occasioni preziose per intervenire in modo mirato e trasformativo. [34]
- *Boundaries:* definire i confini di un sistema è un atto tanto necessario quanto delicato. I confini stabiliscono cosa viene incluso nell'analisi e cosa resta fuori, quali attori sono coinvolti e quali problemi vengono considerati rilevanti. Tuttavia, i confini non sono fissi: possono (e devono) essere ridefiniti in base al contesto, agli obiettivi e alle prospettive in gioco. In un mondo fatto di sistemi dentro altri sistemi, imparare a riflettere criticamente sui confini è una competenza chiave. [34]

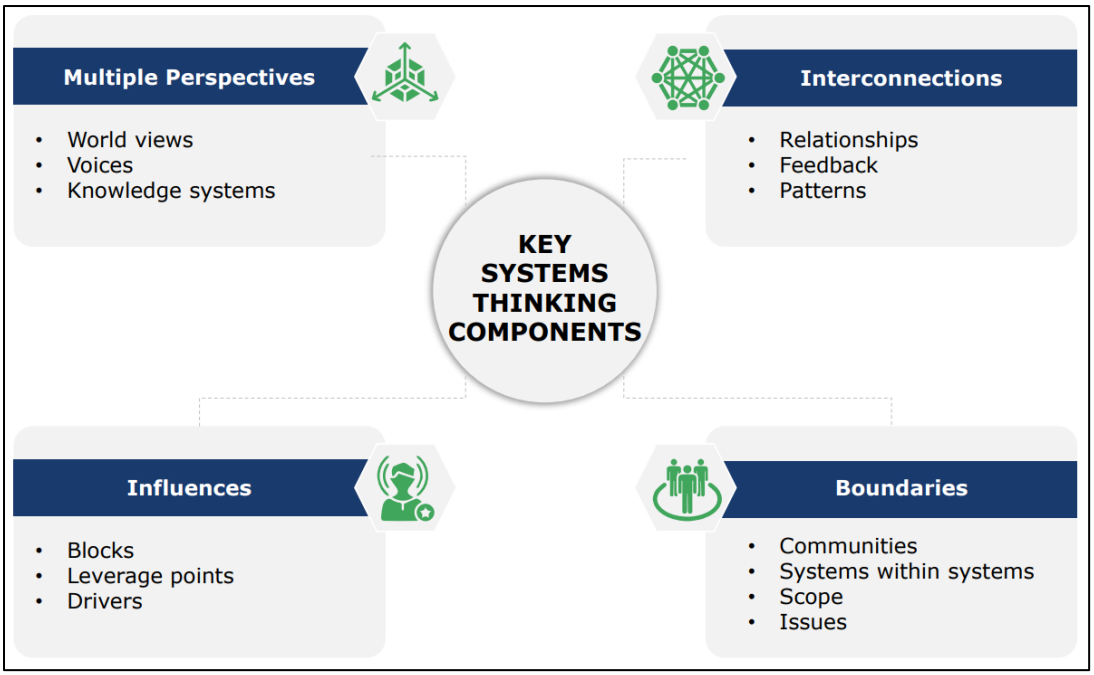

Fig.7 key systems thinking components [34]

In definitiva, il systems thinking non è solo un insieme di strumenti o tecniche, ma un vero e proprio modo di pensare, un approccio che invita a guardare oltre l'apparenza dei problemi.

Quindi, il processo di ingegneria dei sistemi, supportato dal V-Model, dai decisioni gates e dalla gestione delle baseline, costituisce un approccio robusto e metodico per affrontare la complessità dei progetti moderni. Questi strumenti permettono di mantenere il controllo tecnico e gestionale lungo tutto il ciclo di vita del sistema, assicurando qualità, tracciabilità e coerenza.

1.6 Lifecycle stages

Nel contesto dell'ingegneria dei sistemi, il concetto di ciclo di vita rappresenta una struttura fondamentale per guidare la progettazione, lo sviluppo, l'implementazione e la dismissione di un sistema complesso. Ogni sistema attraversa una serie di fasi distinte ma interconnesse, che ne definiscono l'evoluzione dalla concezione iniziale fino al ritiro definitivo. Comprendere e gestire queste fasi è essenziale per garantire che il sistema soddisfi le esigenze degli stakeholder, sia sostenibile nel tempo e mantenga un elevato livello di qualità e affidabilità. [1]

La prima fase è quella concettuale, in cui si riconosce un bisogno o un'opportunità, e si avvia un'esplorazione per definire il problema e valutare possibili soluzioni. In questa fase si svolgono studi di fattibilità, analisi di missione, simulazioni e prototipazioni preliminari; infatti, è un momento strategico, poiché le decisioni prese qui influenzano profondamente le possibilità future,

inoltre, è anche il momento in cui si iniziano a raccogliere i primi requisiti degli stakeholder e si delineano le prime architetture concettuali. [1]

Segue la fase di sviluppo, in cui le idee generate nella fase precedente vengono trasformate in una soluzione ingegneristica concreta. Attraverso attività di modellazione, analisi dei compromessi, simulazioni e progettazione architetturale, si definisce il cosiddetto "*System of interest*" (SOI). Questa fase culmina nella produzione di una baseline tecnica che include i requisiti di sistema, la documentazione progettuale e i piani per le fasi successive, come l'integrazione, la verifica, la validazione e il supporto logistico. [1]

Una volta completata la progettazione, si passa alla fase di produzione, in cui il sistema viene fisicamente realizzato. Questo passaggio richiede l'approvazione formale delle baseline sviluppate e coinvolge anche i sistemi abilitanti necessari alla produzione. Il sistema viene costruito, testato e preparato per l'installazione e la transizione verso l'ambiente operativo. La documentazione prodotta in questa fase è fondamentale per garantire la tracciabilità e il supporto nelle fasi successive. [1]

La fase di utilizzo inizia con la messa in servizio del sistema. Durante questo periodo, il sistema viene impiegato nel contesto previsto per fornire le capacità richieste. È una fase spesso lunga, durante la quale possono emergere necessità di modifiche, sia per correggere difetti, sia per migliorare le prestazioni o estendere la vita utile del sistema. È fondamentale mantenere attivi i processi di gestione tecnica, come la gestione della configurazione e del rischio, per assicurare che il sistema continui a operare in modo efficace e conforme ai requisiti. [1]

Parallelamente all'utilizzo, si sviluppa la fase di supporto, che comprende tutte le attività necessarie a mantenere il sistema operativo. Questo include la manutenzione, la gestione delle anomalie, l'aggiornamento della documentazione e l'eventuale introduzione di modifiche evolutive. Le modifiche devono essere attentamente valutate per evitare impatti negativi sulle prestazioni o sulla conformità normativa. La fase di supporto termina quando si decide che il sistema ha raggiunto la fine della sua vita utile o non è più economicamente sostenibile. [1]

Infine, si giunge alla fase di dismissione, in cui il sistema o sue parti vengono ritirati dall'operatività. Questa fase richiede una pianificazione accurata, che dovrebbe iniziare già nelle fasi iniziali del ciclo di vita. Le attività includono la rimozione fisica, lo smaltimento dei materiali, la cessazione dei servizi associati e l'archiviazione della documentazione. In molti contesti normativi, il produttore è responsabile della corretta gestione del fine vita del sistema, anche in termini ambientali e legali. [1]

In sintesi, il ciclo di vita di un sistema non è solo una sequenza di fasi tecniche, ma un processo continuo e dinamico che richiede una visione sistemica, una gestione rigorosa e un coinvolgimento costante degli stakeholder. Ogni fase contribuisce in modo determinante alla qualità complessiva del sistema e alla sua capacità di rispondere efficacemente alle esigenze per cui è stato concepito.

1.7 Decision gates

Nel ciclo di vita di un sistema, i decision gates rappresentano momenti strategici di valutazione e approvazione. Questi punti decisionali, collocati all'inizio e alla fine di ciascuna fase, servono a garantire che ogni attività sia completata in modo soddisfacente prima di procedere con la successiva, infatti, la loro funzione principale è quella di gestire il rischio, assicurare la maturità del sistema e verificare la conformità agli obiettivi di progetto [1].

Ogni decision gate è associato a criteri di ingresso e uscita, che devono essere definiti in fase di pianificazione e approvati dal management o dagli stakeholder [1]. Questi criteri includono:

- la verifica del raggiungimento della maturità tecnica prevista;
- la validazione dei deliverable rispetto al business case;
- la disponibilità di risorse adeguate per le fasi successive;
- la risoluzione di eventuali problemi aperti;
- la valutazione del rischio complessivo per il proseguimento.

Le decisioni prese in corrispondenza di questi gate possono includere l'avvio della fase successiva, la continuazione della fase corrente con eventuali modifiche, il ritorno a una fase precedente, la sospensione temporanea del progetto o la sua terminazione. In sistemi complessi, è possibile che le fasi non siano sequenziali e che alcune attività si svolgano in parallelo, inoltre, in certi casi si può decidere di far avanzare solo una parte del lavoro, sospenderne un'altra o riformularne una terza [1].

L'approvazione di un decision gate avviene dopo una revisione da parte di esperti qualificati e stakeholder coinvolti, basata su evidenze di conformità ai criteri stabiliti; si nota che una gestione superficiale o l'omissione di discipline critiche può comportare ritardi significativi e costi aggiuntivi, inoltre, eventuali cambiamenti nel contesto del progetto (ambito, risorse, obiettivi) devono essere considerati e i criteri aggiornati di conseguenza [1].

Al termine di un decision gate, gli artefatti approvati (come documenti, modelli, risultati di analisi) vengono posti sotto gestione della configurazione, insieme alle decisioni prese, alle assunzioni e alle motivazioni che le hanno supportate [1].

Un altro concetto chiave è quello di baseline, ovvero l'identificazione di una configurazione del sistema in un determinato istante temporale, utilizzata come riferimento per confronti futuri. Le baseline rappresentano lo stato ufficiale del sistema in un certo momento e sono fondamentali per il controllo delle modifiche e la gestione della configurazione [1].

Nel contesto del modello a V, le baseline si evolvono progressivamente da sinistra verso destra, accompagnando l'avanzamento del progetto [1]. Ogni baseline può includere:

- documentazione tecnica

- specifiche
- modelli
- codice sorgente
- altri artefatti rilevanti
- La loro gestione consente di:
- Tracciare l'evoluzione del sistema nel tempo
- Garantire la coerenza tra le versioni
- Supportare le attività di verifica e validazione

1.8 L'Ingegneria dei Sistemi secondo la Norma ISO/IEC 15288 [2]: una visione integrata dei processi

Nel campo dell'ingegneria dei sistemi, la norma ISO/IEC 15288:2015 [2] rappresenta un punto di riferimento fondamentale per la gestione dell'intero ciclo di vita di un sistema, dalla sua concezione iniziale fino alla dismissione. Questa norma non si limita solo a fornire linee guida tecniche, ma propone una vera e propria struttura concettuale e operativa per affrontare la complessità dei sistemi moderni in modo sistemico e coerente. [2]

La norma suddivide i processi in quattro grandi categorie, come riportato in figura 8, che rispondono a esigenze specifiche e complementari: i processi tecnici, i processi di gestione tecnica, i processi di accordo e i processi abilitanti dell'organizzazione. Questa classificazione non è solo funzionale, ma riflette una visione integrata del sistema come entità tecnica, gestionale, contrattuale e organizzativa.

I "*technical processes"* costituiscono la spina dorsale dell'ingegneria dei sistemi, guidano la trasformazione dei bisogni iniziali in un prodotto o servizio concreto, funzionante e sostenibile. Si parte dall'analisi della missione e dalla definizione dei requisiti, per poi passare alla progettazione dell'architettura, alla realizzazione, all'integrazione e alla verifica del sistema, mentre successivamente vi sono le fasi di transizione, operazione, manutenzione e infine smaltimento. [1,2]

Questi processi non sono lineari, ma si intrecciano in un ciclo continuo di feedback e adattamento, infatti, ogni fase è pensata per garantire che il sistema risponda in modo efficace alle esigenze degli stakeholders, mantenendo al contempo coerenza, tracciabilità e qualità.

Accanto ai technical processes, la norma introduce una serie di "*technical management processes"* dedicati alla gestione tecnica del progetto; questi includono la pianificazione, il monitoraggio, la gestione dei rischi, delle decisioni, della configurazione e delle informazioni. Questi sono processi trasversali, che accompagnano tutte le fasi del ciclo di vita e che permettono di mantenere il controllo su tempi, costi, risorse e qualità. [2]

Si nota che in un contesto complesso e dinamico, questi processi svolgono un ruolo cruciale nel garantire che le attività tecniche siano svolte in modo coordinato, efficiente e allineato agli obiettivi strategici.

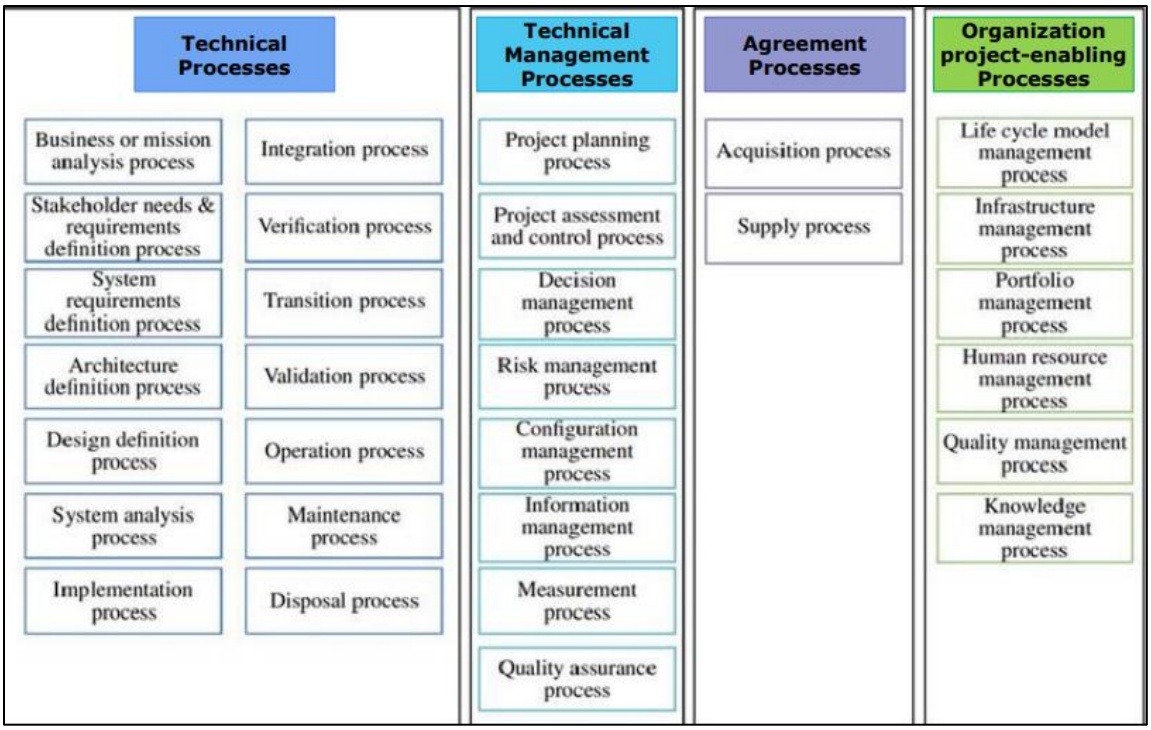

Fig.8 Systems life cycle process for ISO/IEC/IEEE15288:2015. This figure is excerpted from ISO/IEC/IEEE15288:2015 [2]

Un sistema è sempre il frutto di interazioni tra più attori, spesso appartenenti a organizzazioni diverse; gli agreement processes regolano proprio queste relazioni, definendo le modalità con cui si stabiliscono e si gestiscono i contratti tra committenti e fornitori. [2]

La norma, inoltre, distingue tra *"acquisition process"*, che riguarda l'organizzazione che richiede un prodotto o servizio, e supply process, che riguarda chi lo realizza; questi processi sono fondamentali per garantire chiarezza, trasparenza e responsabilità lungo tutta la catena del valore tra l'ente che richiede un prodotto/servizio e l'ente che si occupa di sviluppare tale prodotto/servizio.[2]

Infine, la norma riconosce che nessun progetto può avere successo senza un'organizzazione capace di sostenerlo e abilitarlo, per questo vi sono i processi abilitanti che riguardano aspetti come la gestione del ciclo di vita, delle infrastrutture, del portafoglio progetti, delle risorse umane, della qualità e della conoscenza. [2]

Questi processi non agiscono direttamente sul sistema, ma creano le condizioni organizzative e culturali affinché i progetti possano essere avviati, gestiti e conclusi con successo; in altre parole, rappresentano il terreno fertile su cui può crescere un'ingegneria dei sistemi realmente efficace [2]

Quindi la norma ISO/IEC 15288 non si limita a elencare processi: essa propone una visione sistemica e integrata dell'ingegneria dei sistemi, in cui ogni attività è parte di un insieme più ampio e interconnesso. Utilizzare la tabella presente nella figura 8 consente di visualizzare chiaramente questa struttura, facilitando la comprensione e l'applicazione pratica dei concetti.

In un mondo in cui i sistemi diventano sempre più complessi, interdipendenti e critici, adottare un approccio basato su questa norma significa dotarsi di una guida solida e flessibile per affrontare le sfide del presente e del futuro [2].

Nel contesto dell'ingegneria dei sistemi, la fase iniziale del ciclo di vita riveste un'importanza cruciale, infatti, è in questo momento che si gettano le basi per tutto ciò che seguirà: dalla progettazione alla realizzazione, fino all'implementazione e alla gestione operativa del sistema. Pertanto, di seguito viene fornito uno sguardo rapido ai technical processes, che in figura 9 vengono rappresentati all'interno di un modello a V.

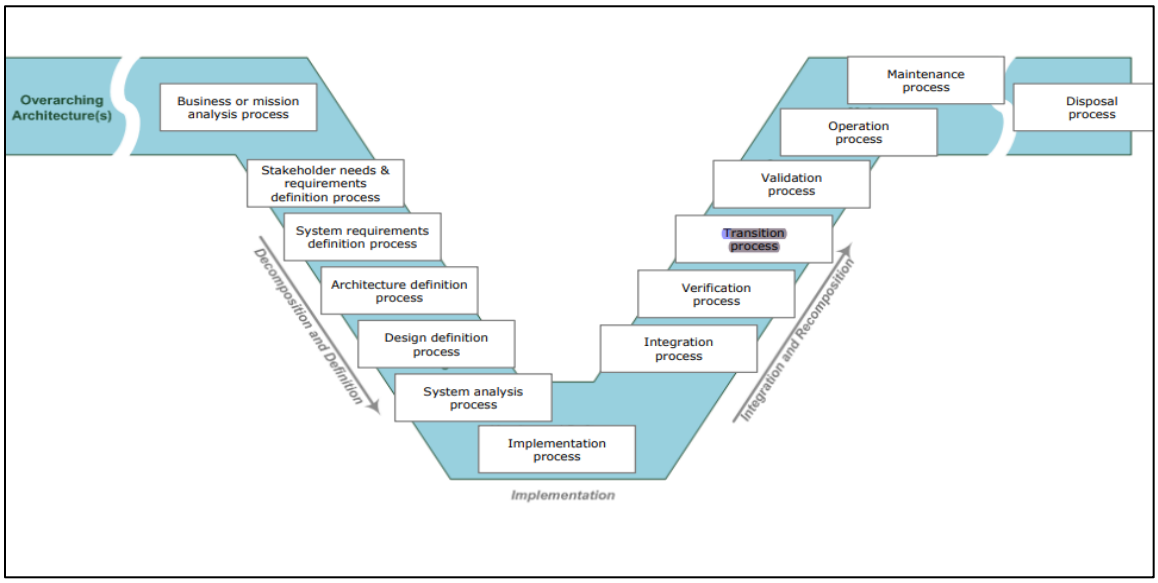

Fig.9 overview of the technical processes [8]

1.8.1 Mission Analysis Process

Si inizia con *Business or Mission Analysis Process*, che rappresenta il punto di partenza dell'intero approccio sistemico, con obiettivo quello di comprendere il problema o l'opportunità che ha dato origine all'iniziativa progettuale. Si parte quindi da documenti strategici, piani operativi e vincoli organizzativi per delineare il contesto in cui il sistema dovrà operare, attraverso attività come la caratterizzazione dello spazio delle soluzioni, la valutazione delle alternative e la gestione dell'analisi stessa, si arriva a definire una strategia di analisi, una prima versione del *concept of operations* (CONOPS) e una mappatura preliminare degli stakeholders. È in questa fase che si iniziano a identificare le mesure of effectiveness (MOE) e le mesure of performance (MOP), strumenti fondamentali per valutare il successo del sistema. [1,2]

1.8.2 Stakeholder Needs & Requirements Definition Process

Si continua poi con *Stakeholder Needs & Requirements Definition Process*; una volta compreso il contesto e chiarita la missione, il passo successivo è quello di ascoltare e interpretare i bisogni degli stakeholders. Questo processo ha lo scopo di tradurre le aspettative, i vincoli e le esigenze degli attori coinvolti in requisiti chiari e tracciabili e le attività previste includono la definizione e analisi dei bisogni, la costruzione del operational concept, la trasformazione dei bisogni in requisiti tecnici e la gestione dell'intero processo di definizione. Il risultato è un insieme strutturato di output: dalla strategia di definizione dei requisiti alla matrice di tracciabilità, fino ai documenti di verifica e validazione. Questa fase è fondamentale per garantire che il sistema risponda realmente alle esigenze di chi lo utilizzerà o ne sarà impattato [1,2].

1.8.3 System Requirements Definition Process

Dopo aver definito gli stakeholder needs and requirements si procede con System Requirements Definition Process; questo terzo processo analizzato consente di trasformare la visione orientata all'utente in una visione tecnica, in altre parole, si passa da "cosa serve" a "come si realizza". Questo processo prende in input i requisiti e i bisogni degli stakeholders, i vincoli di ciclo di vita e i risultati delle fasi precedenti, e li elabora per produrre un insieme coerente di requisiti di sistema. Le attività principali includono la definizione, l'analisi e la gestione dei requisiti di sistema e il risultato è una base solida su cui costruire l'architettura e il design del sistema, assicurando coerenza, tracciabilità e allineamento con gli obiettivi iniziali. [1,2]

1.8.4 Architecture Definition Process

Successivamente si prosegue con *Architecture Definition Process*, ovvero il processo di definizione dell'architettura ha lo scopo di generare, valutare e selezionare alternative architetturali che siano in grado di soddisfare i requisiti del sistema e di rispondere alle preoccupazioni degli stakeholder. L'architettura non è solo una struttura tecnica, ma un insieme coerente di visioni (views) che rappresentano il sistema da diverse prospettive: operativa/operazionale, funzionale, logica e fisica. Le attività principali includono lo sviluppo di visioni architetturali, l'identificazione e l'allocazione delle funzioni ai vari elementi, la valutazione delle alternative e la gestione dell'architettura selezionata, mentre gli output di questo processo (come la *system architecture definition strategy* e la *system interface definition*) costituiscono la base per la progettazione dettagliata. [1,2]

1.8.5 Design Definition Process

Dopodichè si ha il *Design Definition Process*, dove si traducono le entità architetturali in elementi progettuali concreti; questo processo di definizione del design fornisce i dettagli necessari per implementare il sistema in modo coerente con l'architettura. Le attività comprendono la definizione delle caratteristiche progettuali, l'allocazione degli elementi, la valutazione delle alternative di realizzazione e la gestione del design e i risultati includono descrizioni dettagliate degli elementi del sistema, aggiornamenti dei parametri tecnici (TPM), tracciabilità del design e documentazione

analitica. Si nota che questo processo è fondamentale per garantire che ogni parte del sistema sia progettata in modo coerente, verificabile e integrabile, riducendo i rischi di incoerenze o errori nelle fasi successive. [1,2]

1.8.6 System Analysis Process

Il *System Analysis Process*, ovvero il processo di analisi del sistema fornisce una base rigorosa di dati e informazioni per supportare il processo decisionale lungo tutto il ciclo di vita; l'analisi del sistema non è un'attività isolata, ma un processo continuo che accompagna tutte le fasi del progetto, fornendo valutazioni quantitative e qualitative per guidare le scelte tecniche. Le attività principali sono l'esecuzione dell'analisi del sistema e la sua gestione, con l'obiettivo di produrre strategie, report e registri che documentino in modo trasparente le valutazioni effettuate. Questo processo è essenziale per garantire che le decisioni siano basate su evidenze solide e non su intuizioni o supposizioni. [1,2]

Dopo aver definito l'architettura e il design del sistema, l'ingegneria dei sistemi entra nella fase realizzativa, in cui le idee e i modelli si trasformano in elementi concreti. È in questa fase che si attuano i processi di implementazione, integrazione e verifica, fondamentali per garantire che il sistema non solo venga costruito, ma anche che funzioni come previsto e soddisfi i requisiti definiti.

1.8.7 Implementation Process

L' *Implementation Process* ha come obiettivo la realizzazione fisica degli elementi del sistema. Si parte da input fondamentali come la descrizione dell'architettura del sistema, il design dettagliato del sistema, le interfacce e le specifiche degli elementi da costruire e attività principali consistono nell'eseguire l'implementazione vera e propria e nel gestire i risultati ottenuti. Gli output di questo processo sono molteplici: dalla strategia di implementazione alla documentazione tecnica, fino ai materiali formativi per operatori e manutentori; particolarmente importante è la tracciabilità dell'implementazione, che consente di collegare ogni elemento realizzato ai requisiti e alle decisioni progettuali da cui deriva. [1,2]

1.8.8 Integration Process

Una volta che i singoli elementi del sistema sono stati realizzati, è necessario integrarli in un sistema coerente e funzionante, a tal proposito vi è l'*Integration Process* che tratta la sintetizzazione degli elementi in un insieme che rispetti i requisiti, l'architettura e il design definiti. L'integrazione non è un'attività banale: richiede una strategia ben pianificata, una gestione attenta delle interfacce e una verifica continua della coerenza tra le parti, gli output includono la strategia di integrazione, i risultati delle verifiche e validazioni integrate, gli aggiornamenti delle definizioni di interfaccia e la documentazione finale. Questo processo è cruciale per garantire che il sistema funzioni come un prodotto ingegneristico armonico, e non come una somma disorganica di componenti. [1,2]

1.8.9 Verification Process

Dopodichè, vi è il *Verification Process* che ha il compito di fornire evidenze oggettive che dimostrino che il sistema (o i suoi elementi) soddisfa i requisiti specificati. Questa è un'attività

fondamentale per assicurare la qualità e la conformità del prodotto finale, infatti, la verifica si basa su criteri ben definiti, su una matrice di tracciabilità dei requisiti (RTM) aggiornata e sui risultati dell'integrazione. Le attività comprendono l'esecuzione delle verifiche e la gestione dei risultati, che vengono documentati in report dettagliati e gli output includono la strategia di verifica, le procedure, i vincoli, gli strumenti abilitanti, e soprattutto lo stato verificato degli elementi, che certifica la loro conformità. [1,2]

1.8.10 Transition Process

Il *Transition Process* ha lo scopo di trasferire il sistema dall'ambiente di sviluppo a quello operativo, garantendo che sia pronto per fornire i servizi richiesti dagli stakeholders; si nota che questo passaggio è delicato, perché implica l'installazione del sistema, la formazione degli operatori e la predisposizione delle condizioni per l'uso effettivo. Tra gli input si possono trovare il sistema verificato, i materiali formativi e i report di verifica, mentre le attività principali sono l'esecuzione della transizione e la gestione dei risultati, che portano alla produzione di output come la strategia di transizione, le procedure di installazione, i vincoli operativi e la documentazione degli operatori formati. Questa fase è fondamentale per garantire che il sistema sia effettivamente utilizzabile nel contesto reale, senza interruzioni o sorprese. [1,2]

1.8.11 Validation Process

Una volta che il sistema è stato installato e messo in funzione, è necessario verificare che soddisfi realmente gli obiettivi della missione e le aspettative degli stakeholders, per questo modivo vi è il *Validation Process*, che fornisce evidenze oggettive del fatto che il sistema, nel suo ambiente operativo, funziona come previsto. Gli input comprendono i requisiti degli stakeholders, il sistema installato, i criteri di validazione e i report di transizione, mentre le attività includono l'esecuzione della validazione e la gestione dei risultati, con output come la strategia di validazione, il sistema validato, i vincoli di validazione e i report finali. La validazione è ciò che consente di dire, con certezza, che il sistema è pronto per l'uso reale e che risponde agli scopi per cui è stato progettato. [1,2]

1.8.12 Operation Process

L'*Operation Process* rappresenta il momento in cui il sistema entra pienamente in funzione, erogando i servizi per cui è stato creato, è qui che il valore del sistema si manifesta concretamente, attraverso l'interazione con gli utenti e l'ambiente operativo. Gli input includono il sistema validato, gli operatori formati, i materiali di supporto e i report di manutenzione, mentre le attività comprendono l'esecuzione dell'operazione, la gestione dei risultati e il supporto al cliente. Gli output sono la strategia operativa, i vincoli operativi, i report di supporto e la documentazione delle attività svolte; questa fase è continua e può durare anni: è qui che si misura la sostenibilità, l'affidabilità e l'efficacia del sistema nel tempo. [1,2]

L'ingegneria dei sistemi non si esaurisce con la messa in servizio del sistema, al contrario, una parte fondamentale del suo valore si manifesta nel tempo, durante l'uso prolungato, la manutenzione e infine la dismissione. Le ultime due fasi del ciclo di vita (manutenzione e smaltimento) sono spesso trascurate, ma rivestono un ruolo cruciale per garantire la continuità operativa, la sicurezza, la sostenibilità e il rispetto delle normative. [1,2]

1.8.13 Maintenance Process

Il *Maintenance Process* ha l'obiettivo di preservare la capacità del sistema di fornire i servizi previsti, anche in presenza di usura, guasti o cambiamenti nel contesto operativo; questa è una fase che richiede pianificazione, risorse e competenze specifiche. Gli input includono il sistema validato, i materiali formativi, i report di validazione e manutenzione, mentre le attività comprendono la preparazione alla manutenzione, l'esecuzione degli interventi, il supporto logistico e la gestione dei risultati; i risultati che si riscontrano sono molteplici: dalla strategia di manutenzione alle procedure operative, fino alla documentazione tecnica e ai report aggiornati. Una manutenzione ben progettata e gestita consente di prolungare la vita utile del sistema, ridurre i costi operativi e garantire la sicurezza e l'affidabilità nel tempo. [1,2]

1.8.14 Disposal Process

Quando un sistema o uno dei suoi elementi ha esaurito la propria funzione, entra in gioco il *Disposal Process*, processo che non riguarda solo la rimozione fisica, ma anche la gestione responsabile degli elementi dismessi, nel rispetto delle normative ambientali, della sicurezza e della tracciabilità. Gli input provengono dalle fasi precedenti: il sistema validato, i report di operazione e manutenzione; le attività includono l'esecuzione dello smaltimento e la sua finalizzazione, con output come la strategia di dismissione, i vincoli e requisiti specifici, le procedure operative e la documentazione finale. Il processo di smaltimento chiude il ciclo di vita del sistema in modo ordinato, sicuro e sostenibile, assicurando che ogni elemento venga trattato in modo appropriato, sia esso riciclato, riutilizzato o eliminato. [1,2]

Con i processi di manutenzione e smaltimento si conclude il ciclo di vita completo di un sistema, così come descritto dalla norma ISO/IEC 15288. Dall'analisi della missione alla dismissione finale, ogni fase è parte di un percorso integrato e tracciabile, che mira a garantire efficacia, efficienza e responsabilità. [2]

L'approccio sistemico non si limita a progettare "cose che funzionano", ma si preoccupa di come funzionano nel tempo, di come vengono mantenute, e infine di come vengono dismesse. È questa visione completa e consapevole che rende l'ingegneria dei sistemi uno strumento fondamentale per affrontare le sfide tecnologiche, organizzative e ambientali del nostro tempo.

CAPITOLO 2 Requirements Engineering

Nel contesto dello sviluppo di sistemi complessi, l'ingegneria dei sistemi si configura come un elemento cardine per garantire l'allineamento tra le esigenze operative degli utenti e le soluzioni tecniche fornite dagli sviluppatori. Questo ruolo può essere efficacemente rappresentato attraverso la metafora del ponte, che collega due sponde apparentemente distanti (fig.10): da un lato gli utenti, portatori di bisogni operativi spesso espressi in termini funzionali e contestuali (requisiti); dall'altro gli sviluppatori di sistema, suddivisi in discipline specialistiche come l'ingegneria hardware, software e delle specialità. [1,8]

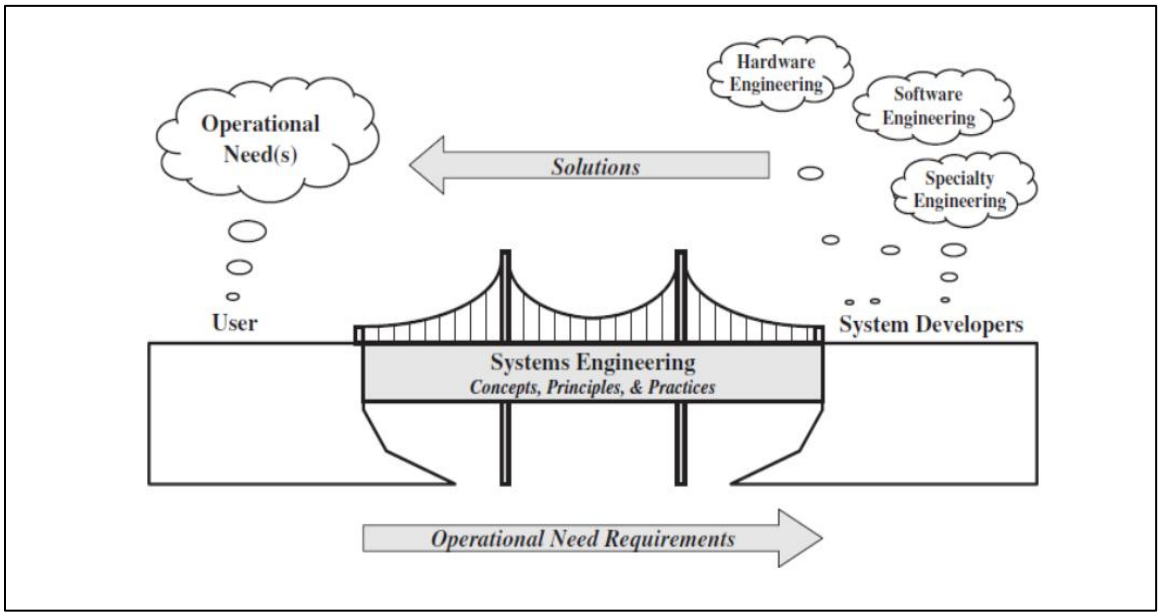

Fig.10 System Engineering: a bridge between users and developers [8,28]

L'ingegneria dei sistemi si colloca esattamente al centro di questo scenario, fungendo da mediatore strutturato tra domanda e offerta, infatti, attraverso l'applicazione di concetti, principi e pratiche consolidate, essa consente di tradurre le esigenze operative in requisiti tecnici chiari e verificabili, che possono poi essere implementati in soluzioni ingegneristiche coerenti e sostenibili. [8]

Il processo non è unidirezionale: oltre a fornire soluzioni, l'ingegneria dei sistemi facilita un flusso continuo di feedback tra le parti, assicurando che le soluzioni sviluppate siano effettivamente rispondenti ai bisogni iniziali e che eventuali modifiche o adattamenti possano essere gestiti in modo tempestivo e controllato. [1, 8]

In sintesi, l'ingegneria dei sistemi non si limita a una funzione tecnica, ma assume un ruolo strategico nella gestione dell'intero ciclo di vita del sistema, promuovendo la comunicazione, la coerenza e l'efficacia tra tutti gli attori coinvolti. [1]

2.1 Trasformazione dei bisogni in requisiti tecnici

Uno degli aspetti più critici e determinanti per il successo di un progetto è la trasformazione strutturata dei bisogni in requisiti; questo processo non è immediato né lineare, ma si articola attraverso una serie di livelli gerarchici e analitici, ciascuno dei quali contribuisce a raffinare e formalizzare le esigenze iniziali in specifiche tecniche implementabili. [1,10]

Il concetto di requisito riveste un ruolo centrale e imprescindibile. Si forniscono di seguito le definizioni di requisito secondo lo standard ISO/IEC/IEEE 29148 [9]:

- "*Una condizione o capacità necessaria per un utente al fine di risolvere un problema o raggiungere un obiettivo*".[9]
- "*Una condizione o capacità che deve essere soddisfatta da un sistema o da un suo componente per rispettare contratti, standard, specifiche o altri documenti formali.*" [9]
- "*Una rappresentazione documentata di una delle due definizioni precedenti.*" [9]

Queste definizioni evidenziano come il requisito non sia semplicemente un desiderio o una preferenza, ma piuttosto un vincolo essenziale che guida la progettazione, lo sviluppo e la verifica di un sistema. In ingegneria, infatti, un requisito è qualsiasi elemento che influenzi il modo in cui un prodotto o sistema viene progettato e costruito, questo include funzionalità, prestazioni, sicurezza, conformità normativa e molto altro.[9]

È in questo contesto che si inserisce l'ingegneria dei requisiti, una disciplina che costituisce una branca fondamentale dell'ingegneria dei sistemi; essa ha il compito di sviluppare e mantenere un insieme adeguato di requisiti per il sistema, il servizio o il software da realizzare. L'ingegneria dei requisiti è una funzione multidisciplinare, che traduce i bisogni del cliente in requisiti privi di significati tecnici durante la fase di elicitazione (ma comunque ben scritti, non ambigui e formalizzati dal punto di vista dello user/stakeholder), fino a trasformarli in requisiti tecnici che costituiscono una base solida e utilizzabile per l'implementazione del sistema. [10]

La Fig.11 propone una visione sistemica di questo processo, suddividendolo in due macro-sezioni: la "Needs view" e la "Requirements view". Nella prima, si parte da elementi di alto livello come le strategie aziendali e i concetti preliminari di ciclo di vita (operazioni, acquisizione, distribuzione, supporto, dismissione), che definiscono il contesto operativo e gestionale del sistema; questi elementi generano i bisogni aziendali, che vengono analizzati attraverso il processo di Business or Mission Analysis, producendo una prima formalizzazione: la Business Requirements Specification (BRS). [1,8]

Successivamente, si passa alla definizione dei bisogni degli stakeholder, che vengono analizzati nel processo di Stakeholder Needs and Requirements Definition, portando alla redazione della Stakeholder Requirements Specification (StRS); si nota che questo passaggio è fondamentale per garantire che le esigenze di tutte le parti interessate siano comprese e integrate nel progetto. Il processo continua con l'identificazione dei requisiti di sistema, che vengono redatti nella System Requirements Specification (SyRS) in cooperazione con il processo di System Architecture

Definition. Infine, si arriva al livello più granulare, quello degli elementi di sistema, i cui requisiti vengono analizzati e formalizzati nella System Element Requirements Specification. [1,8]

Ogni livello di analisi è accompagnato da una riflessione sui concetti di ciclo di vita specifici per quel livello, a sottolineare l'importanza di considerare l'intero arco di esistenza del sistema, dalla concezione alla dismissione. [1,8]

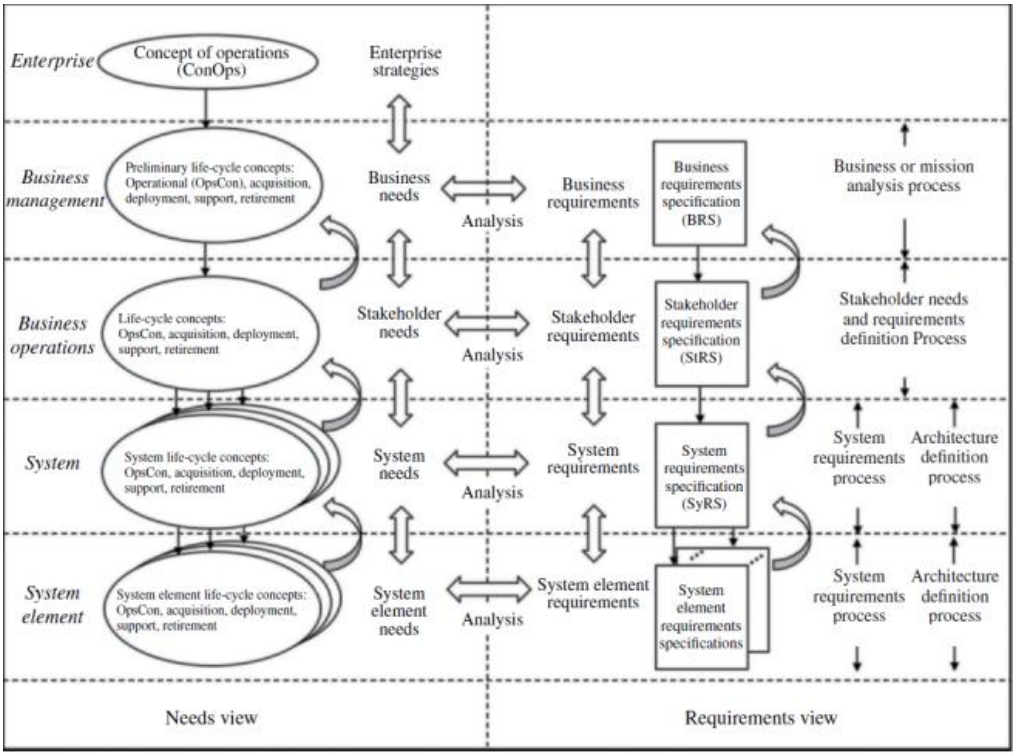

Fig.11 Transformation of needs into requirements [29]

In sintesi, la trasformazione dei bisogni in requisiti, e lo sviluppo dei requisiti di Sistema, Sottosistemi e Componenti è un processo iterativo, multilivello e interdisciplinare, che consente di garantire la tracciabilità, coerenza e completezza delle specifiche di sistema, riducendo il rischio di ambiguità e incomprensioni tra stakeholder e sviluppatori. [1]

A supporto dell'applicazione pratica dello standard ISO/IEC 15288:2015 per la System Engineering di cui si è parlato precedentemente, l'INCOSE (International Council on Systems Engineering) ha pubblicato il Systems Engineering Handbook, versione 5, che ne interpreta e approfondisce i contenuti, offrendo linee guida operative per i professionisti del settore. [1] Sempre in ambito INCOSE, il Requirements Working Group ha sviluppato la Guide for Writing Requirements, un documento che fornisce indicazioni concrete per la redazione di requisiti chiari, verificabili e privi di ambiguità. [10] Un ulteriore riferimento fondamentale è rappresentato dallo standard ISO/IEC/IEEE 29148 [8], che si concentra specificamente sull'ingegneria dei requisiti.

Questo standard descrive in dettaglio i processi necessari per identificare, analizzare, validare e gestire i requisiti lungo tutto il ciclo di vita del sistema.[9]

Secondo la ISO 29148 [9], che evidenzia come i requisiti emergano e si articolino all'interno di un contesto organizzativo e ambientale ben definito, per dettagliare accuratamente un sistema devono essere descritti l'interazione tra il sistema stesso e l'ambiente esterno, l'ambiente organizzativo, le operazioni aziendali e quelle di sistema, fino ad arrivare agli elementi costitutivi del sistema, come il software. In questo flusso, i bisogni degli stakeholder rappresentano il punto di partenza per la definizione dei requisiti, che vengono poi declinati a diversi livelli, in funzione della struttura e delle responsabilità dell'organizzazione. [9]

Questo approccio multilivello consente di garantire che ogni requisito sia tracciabile rispetto a un bisogno reale e che sia coerente con gli obiettivi strategici e operativi dell'organizzazione, quindi, in definitiva, l'integrazione tra standard, linee guida e contesto organizzativo costituisce la base per una gestione efficace e sostenibile dei requisiti nei progetti di ingegneria dei sistemi. [9,10]

Uno dei compiti principali dell'ingegnere di sistema è quello di stabilire i requisiti a livello di sistema e di allocarli correttamente ai sottosistemi, questa attività non solo guida l'intero processo di progettazione, ma garantisce anche che ogni componente del sistema contribuisca in modo coerente al raggiungimento degli obiettivi complessivi. [1]

Il processo di sviluppo dei requisiti viene articolato in sei fasi principali [9]:

- Sviluppare i requisiti, partendo dai bisogni degli stakeholders e dal contesto operativo.
- Scrivere e documentare i requisiti, rendendoli espliciti e condivisibili.
- Verificare la completezza, assicurandosi che tutti gli aspetti rilevanti siano stati considerati.
- Analizzare, affinare e decomporre i requisiti, per renderli più dettagliati e assegnabili ai vari livelli del sistema.
- Validare i requisiti, ovvero verificarne la coerenza con i bisogni iniziali e la fattibilità tecnica.
- Gestire i requisiti, mantenendoli aggiornati e tracciabili lungo tutto il ciclo di vita del sistema.

Prendendo come esempio fig.12, si può vedere la suddivisione dei requisiti di alto livello in requisiti più specifici e dettagliati, assegnabili ai singoli sottosistemi o componenti. Un aspetto particolarmente rilevante è l'importanza di giustificare i requisiti attraverso modelli o analisi; i requisiti più solidi e affidabili sono quelli che non si limitano a enunciare una necessità, ma che sono accompagnati da una motivazione chiara, da un modello di riferimento o da un'analisi tecnica che ne dimostri la validità. Questa giustificazione può essere inserita direttamente nel documento dei requisiti, spesso in corsivo, subito sotto il requisito stesso, per facilitarne la comprensione e la tracciabilità. [8, 10]

2.2 Decomposizione e classificazione dei requisiti

Il diagramma presente in fig.12 illustra visivamente questo processo: i requisiti di sistema vengono derivati e suddivisi in requisiti software, manuali e hardware, che a loro volta vengono allocati a componenti specifici come Component A, Component B, Component C, Component D e anche alle persone, evidenziando come non tutti i requisiti siano necessariamente di natura tecnica. [30]

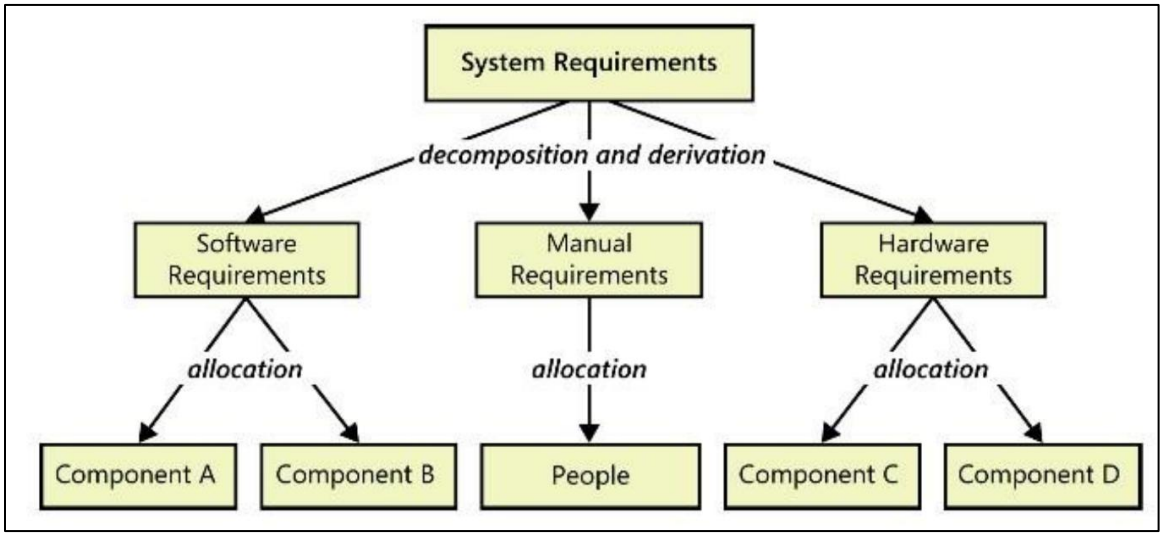

Fig.12 System requirements are decomposed into software, hardware, and manual requirements, then allocated to appropriate components. [30]

Questo approccio consente di mantenere una coerenza verticale tra i livelli del sistema, assicurando che ogni componente contribuisca in modo misurabile al soddisfacimento dei requisiti di alto livello; inoltre, la decomposizione facilita la gestione del progetto, permettendo di assegnare responsabilità chiare e di verificare il rispetto dei requisiti in modo puntuale. [30]

Altri due concetti da tenere in conto parlando di requisito sono il flow-down dei requisiti e i requisiti derivati. Il flow-down rappresenta la naturale suddivisione dei requisiti in livelli di dettaglio progressivamente più specifici, ad esempio in un sistema complesso, come un aeromobile, si parte dai requisiti di livello A, che rappresentano le specifiche di alto livello del sistema (es. “l’aeromobile deve avere un’autonomia di 300 miglia nautiche”) [8]. Questi vengono poi tradotti in requisiti di livello B, che introducono un maggiore dettaglio tecnico (es. “l’aeromobile deve trasportare 389.000 libbre di carburante”), fino ad arrivare ai requisiti di livello C, che definiscono le soluzioni progettuali concrete (es. “il carburante deve essere distribuito in 5 serbatoi, 2 nelle ali e 3 nella fusoliera, per garantire una corretta gestione del baricentro”). Questo approccio consente di mantenere una tracciabilità verticale tra i requisiti e le soluzioni implementative, facilitando la verifica e la validazione del sistema. [8,10]

I requisiti derivati invece, non sono direttamente dichiarati, ma emergono dall'applicazione di principi ingegneristici, analisi matematiche, normative o pratiche consolidate. Le definizioni fornite da INCOSE chiariscono ulteriormente il concetto: "*i requisiti derivati sono spesso identificati durante la sintesi delle soluzioni preliminari, oppure inferiti da vincoli contestuali, come leggi, regolamenti o standard industriali. Pur non avendo un "requisito padre" diretto, sono indispensabili per garantire che il sistema possa svolgere correttamente le sue funzioni*". [1,10]

Inoltre, i requisiti possono essere classificati in funzionali e non funzionali, classificazione che sebbene apparentemente semplice, ha implicazioni profonde sulla progettazione, sull'implementazione e sulla verifica di un sistema.[9]

I requisiti funzionali descrivono ciò che il sistema deve fare: rappresentano le funzionalità attese e le capacità operative che l'utente si aspetta, cioè definiscono il comportamento del sistema in risposta a determinati input o condizioni. Al contrario, i requisiti non funzionali non aggiungono nuove funzionalità, ma definiscono le qualità che il sistema deve possedere; infatti, si tratta di vincoli che influenzano il modo in cui le funzionalità vengono implementate e percepite. Tra questi rientrano aspetti come le prestazioni, l'affidabilità, la sicurezza, la conformità normativa, la disponibilità del sistema. [9]

Può essere utile pensare ai requisiti funzionali come a capacità, e a quelli non funzionali come a vincoli. Questa distinzione aiuta a chiarire che, mentre i primi definiscono cosa il sistema deve fare, i secondi definiscono come deve farlo, entro quali limiti e con quali caratteristiche qualitative; comprendere e gestire correttamente entrambe le categorie è essenziale per garantire che il sistema non solo soddisfi le aspettative funzionali degli utenti, ma lo faccia anche in modo efficiente, sicuro, conforme e sostenibile nel tempo. [10]

2.3 Tracciabilità dei requisiti

In System Engineering, la tracciabilità dei requisiti rappresenta uno degli elementi più critici e strutturanti dell'intero processo di gestione dei requisiti, infatti, costituisce il cuore della requirements management, poiché consente di mantenere una visione coerente e controllata dell'evoluzione dei requisiti lungo tutto il ciclo di vita del sistema [9]. La tracciabilità è definita come la capacità di descrivere e seguire la vita di un requisito sia in direzione forward (dalla sua origine fino all'implementazione e alla verifica), sia in direzione backward (dalla soluzione tecnica o dal test fino al requisito che l'ha generata). Questo doppio flusso è essenziale per garantire che ogni esigenza iniziale sia effettivamente soddisfatta e che ogni componente del sistema abbia una giustificazione chiara e documentata. [10]

In sintesi, la tracciabilità permette di documentare come gli obiettivi di alto livello vengano trasformati in obiettivi di dettaglio, e di comprendere in che modo i bisogni degli stakeholders vengano tradotti in specifiche tecniche, implementati e infine verificati. Essa consente inoltre di valutare l'impatto di eventuali modifiche, facilitando la gestione del cambiamento e riducendo il rischio di errori o incoerenze. [1,10] Le definizioni fornite dagli standard internazionali rafforzano questo concetto. Secondo la norma ISO/IEC/IEEE 29148, la tracciabilità è "*il grado in cui può essere stabilita una relazione tra due o più prodotti del processo di sviluppo, in particolare tra*

prodotti che hanno una relazione di predecessore-successore o di subordinazione" [9]. L'ISO 8402, invece, la definisce come "la capacità di tracciare la storia, l'applicazione o la localizzazione di un'entità attraverso identificazioni registrate". [11]

La Figura 13 rappresenta visivamente il concetto di tracciabilità, mostrando una catena di relazioni che parte dalla dichiarazione del bisogno e arriva fino all'uso operativo del sistema, passando attraverso i vari livelli di requisiti e i relativi test. [10,11]

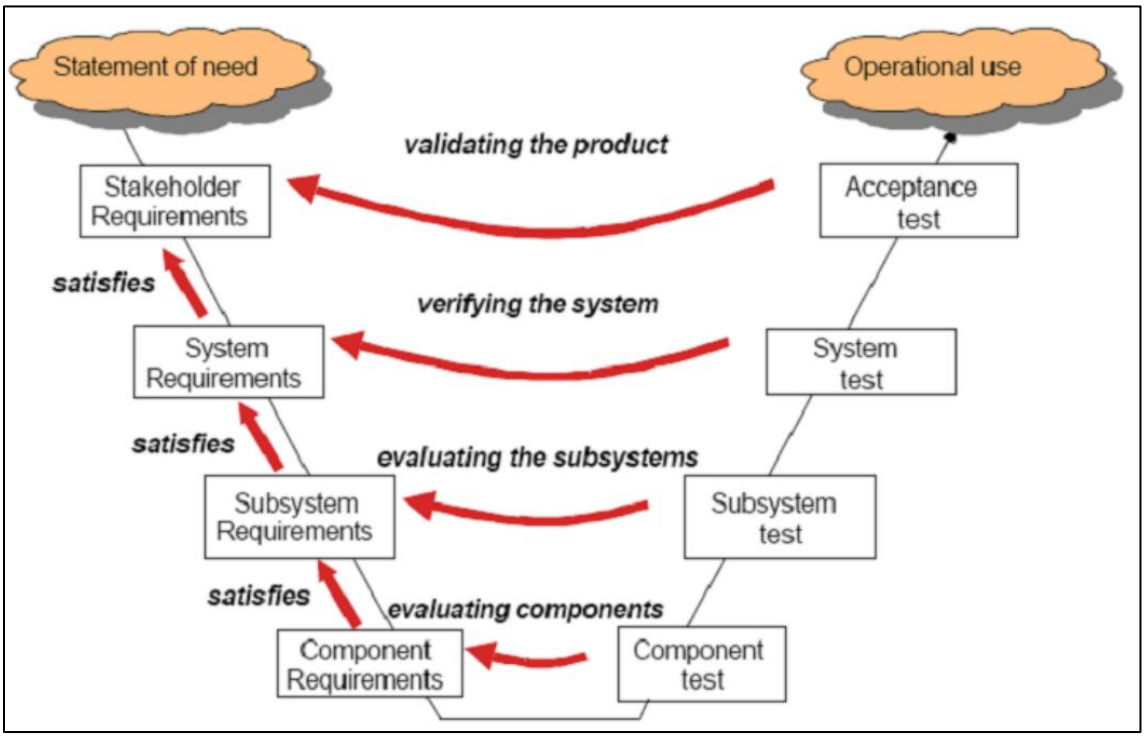

Fig.13 Relationship between the requirement and the source (user reqs or other contract docs) [11,31]

Il diagramma si sviluppa in verticale, con una sequenza ordinata di elementi che riflettono la struttura gerarchica dei requisiti [10,11]:

- Si parte dallo Statement of need, ovvero l'espressione iniziale del bisogno operativo o del problema da risolvere.
- Da qui si derivano gli Stakeholder Requirements, che traducono il bisogno in aspettative e obiettivi misurabili da parte degli utenti o committenti.
- I System Requirements rappresentano la formalizzazione tecnica di tali aspettative, definendo cosa il sistema deve fare nel suo complesso.
- Seguono i Subsystem Requirements, che suddividono le funzionalità tra le varie parti del sistema.

- Infine, i Component Requirements dettagliano le specifiche dei singoli elementi hardware o software.

A ogni livello di requisito è associata una fase di verifica o validazione [10,11]:

1. I Component test verificano che ogni componente soddisfi i requisiti assegnati.
2. I Subsystem test valutano l'integrazione e il corretto funzionamento dei sottosistemi.
3. Il System test verifica che il sistema nel suo insieme risponda ai requisiti di sistema.
4. L'Acceptance test, implicito nel passaggio verso l'Operational use, serve a validare che il prodotto finale soddisfi i bisogni iniziali.

Le frecce che collegano i vari blocchi sono etichettate con verbi chiave come satisfies, verifying, evaluating, validating, che indicano il tipo di relazione tra i requisiti e le attività di test [11]; questo schema evidenzia come la tracciabilità non sia solo un concetto teorico, ma una pratica operativa che consente di:

- garantire la coerenza tra requisiti e implementazione;
- facilitare la verifica e la validazione a ogni livello;
- assicurare che ogni elemento del sistema abbia una giustificazione documentata;
- risalire in modo chiaro alla fonte di ogni requisito, come indicato nella didascalia in basso: "Relationship between the requirement and the source (user reqs or other contract docs)".

In sintesi, la tracciabilità dei requisiti rappresenta il filo conduttore che unisce le intenzioni iniziali del committente con il prodotto finale, passando attraverso ogni fase del ciclo di vita del sistema. Essa garantisce trasparenza, controllo e qualità, rendendo possibile una gestione efficace anche nei progetti più complessi [9,10].

A supporto del processo di definizione e gestione dei requisiti, gli ingegneri di sistema possono applicare anche un approccio di progettazione basato su modelli. Tale approccio verrà descritto nel capitolo successivo.

CAPITOLO 3 Model-Based Systems Engineering 3

"L'ingegneria dei sistemi basata su modelli (MBSE) è l'applicazione formalizzata della modellazione per supportare i requisiti del sistema, le attività di progettazione, analisi, verifica e convalida a partire dalla fase di progettazione concettuale e proseguendo durante le fasi di sviluppo e successive del ciclo di vita." [14]

La precedente definizione di Model-Based Systems Engineering (MBSE) mette in luce alcuni aspetti fondamentali che caratterizzano questo approccio innovativo all'ingegneria dei sistemi, tra cui in primo luogo, l'MBSE si basa sull'uso formalizzato della modellazione, che non si limita a semplici rappresentazioni grafiche, ma impiega strumenti rigorosi e strutturati, costruiti secondo linguaggi standardizzati come il SysML (Systems Modeling Language). Questi modelli non solo descrivono il sistema, ma sono anche coerenti, tracciabili e spesso eseguibili, permettendo così di effettuare simulazioni, analisi e validazioni automatiche già nelle fasi iniziali dello sviluppo. [1,15]

Un altro elemento distintivo dell'MBSE è la sua capacità di coprire l'intero ciclo di vita del sistema, infatti, a differenza dei metodi tradizionali, che spesso si concentrano sulla sola fase di progettazione, l'approccio basato su modelli accompagna il sistema dalla concezione iniziale fino alla dismissione; questo garantisce una continuità digitale tra le diverse fasi del progetto, riducendo la perdita di informazioni e migliorando la coerenza tra i vari artefatti ingegneristici prodotti nel tempo. [1]

Inoltre, fornisce un supporto concreto alle principali attività ingegneristiche, ad esempio, attraverso i modelli è possibile gestire in modo integrato i requisiti, progettare architetture funzionali e fisiche, analizzare il comportamento del sistema e verificarne la conformità rispetto alle specifiche rendendo così possibile anticipare la rilevazione di errori e incongruenze, contribuendo così a ridurre i costi e i tempi di sviluppo. [15]

Quindi l'MBSE rappresenta un vero e proprio cambio di paradigma nell'ingegneria dei sistemi, ovvero si passa da una gestione documentale frammentata a un modello centrale, condiviso e dinamico, che diventa il riferimento principale per tutte le attività di sviluppo. Questo favorisce la collaborazione multidisciplinare, migliora la qualità del sistema e aumenta la capacità di adattamento a cambiamenti e complessità crescenti. [1]

3.1 MBE, MBSE e MBD

Nel contesto dell'ingegneria moderna, l'adozione di approcci basati su modelli ha portato alla definizione di tre concetti chiave: Model-Based Engineering (MBE), Model-Based Systems Engineering (MBSE) e Model-Based Design (MBD) [16]; sebbene questi termini siano spesso utilizzati in modo intercambiabile, è importante comprenderne le differenze e le relazioni gerarchiche per una corretta applicazione metodologica.

Il termine Model-Based Engineering (MBE) rappresenta l'approccio più ampio e generale, ovvero si riferisce all'utilizzo sistematico di modelli come parte integrante della baseline tecnica di un

sistema o prodotto. In questo contesto, i modelli non sono semplici strumenti di supporto, ma elementi centrali che accompagnano tutte le fasi del ciclo di vita ingegneristico: dalla definizione dei requisiti, all'analisi, alla progettazione, fino all'implementazione e alla verifica. L'MBE si applica trasversalmente a diverse discipline e consente una gestione più coerente e integrata delle informazioni tecniche. [16]

All'interno dell'MBE si colloca il concetto di Model-Based Systems Engineering (MBSE), che si focalizza specificamente sull'ingegneria dei sistemi. L'MBSE è definito come l'applicazione formalizzata della modellazione per supportare le attività di definizione dei requisiti, progettazione, analisi, verifica e validazione, a partire dalla fase concettuale e proseguendo lungo tutto il ciclo di vita del sistema. Questo approccio si avvale di modelli operativi, che descrivono il contesto e le missioni del sistema, e di modelli di sistema, che ne rappresentano l'architettura e il comportamento. L'MBSE consente una visione integrata e tracciabile del sistema, migliorando la comunicazione tra stakeholder e riducendo il rischio di incoerenze progettuali. [1,15]

Infine, il concetto di Model-Based Design (MBD) si riferisce all'applicazione dei modelli nelle fasi più tecniche e implementative del progetto. L'MBD è un termine ombrello che include approcci modellistici utilizzati in discipline specifiche come l'ingegneria del software, l'ingegneria hardware, l'ingegneria meccanica e altri ambiti specialistici. Il suo obiettivo principale è quello di supportare l'implementazione e l'analisi tecnica in risposta ai requisiti generati dell' MBSE e tra gli strumenti più comuni associati all'MBD si trovano MATLAB, Simulink, CAD, FEA, strumenti per la progettazione RF e tool di sviluppo software. [13]

3.2 MBSE Methods, tools and languages

Come per l'Ingegneria dei Sistemi anche in MBSE vi sono tre pilastri fondamentali che costituiscono l'ambiente concettuale e operativo dell'approccio Model-Based Systems Engineering (fig.14). [13]

Al centro di fig.14 troviamo il modello, che rappresenta la fonte principale di conoscenza e il riferimento condiviso per tutte le attività ingegneristiche. Un modello è un'approssimazione, rappresentazione o idealizzazione di aspetti selezionati della struttura, del comportamento, delle operazioni o di altre caratteristiche di un sistema o processo del mondo reale, tuttavia, affinché il modello sia efficace, deve essere supportato da tre elementi fondamentali: il metodo, il tool e il linguaggio di modellazione. [1,13,15]

Il *modelling process (metodo)* rappresenta l'approccio strutturato utilizzato per costruire il modello. Può trattarsi di metodologie standardizzate come MagicGrid™ (che verrà utilizzato in questo lavoro di tesi e descritto nel capitolo successivo), oppure di approcci ad hoc sviluppati per specifici contesti industriali o esigenze del cliente. Il metodo guida le attività, definisce le fasi del progetto e stabilisce le regole per garantire coerenza e tracciabilità. [13,17]

Il *modelling tool* è il software che consente di realizzare concretamente il modello. Tra gli strumenti più diffusi si annoverano Capella, Rhapsody, Cameo Systems Modeler (tool utilizzato per lo sviluppo del modello della tesi) ed Enterprise Architect; questi ambienti permettono di creare

diagrammi, tabelle, viste architetturali e altri artefatti modellistici, gestendo la complessità del sistema in modo strutturato. [13,16]

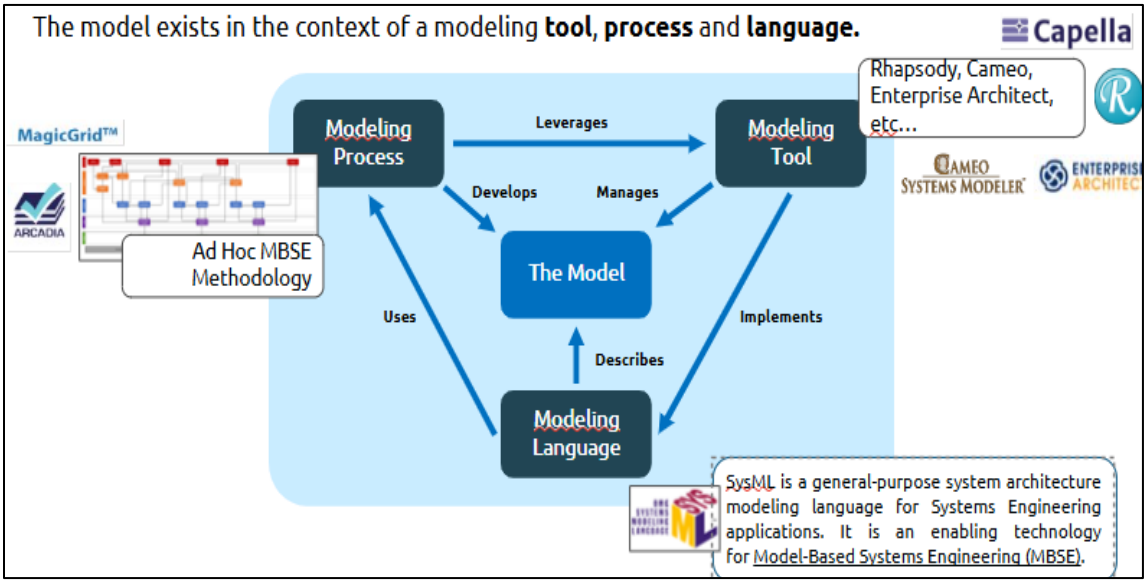

Fig.14 MBSE Environment [13]

Il *modelling language (linguaggio)*, infine, fornisce la sintassi e la semantica necessarie per descrivere formalmente il sistema. Il linguaggio più utilizzato in ambito MBSE è il SysML (Systems Modeling Language), uno standard general-purpose che consente di rappresentare requisiti, architetture, comportamenti e parametri del sistema; tuttavia, l'MBSE può impiegare anche altri linguaggi, a seconda del livello di modellazione e del dominio applicativo. [13,15]

Le relazioni tra questi tre elementi sono fondamentali: il metodo sfrutta lo strumento, lo strumento implementa il linguaggio, e il linguaggio è utilizzato all'interno del metodo e del tool; insieme, questi componenti costituiscono l'ambiente MBSE, che consente di sviluppare modelli coerenti, tracciabili e riutilizzabili lungo tutto il ciclo di vita del sistema. [1]

3.3 Layers of abstraction in MBSE approach

In figura 15 si nota come l'approccio Model-Based Systems Engineering (MBSE) si basa su una struttura multilivello che consente di affrontare la complessità dei sistemi in modo progressivo, tracciabile e coerente. In particolare, si articola attraverso una sequenza logica di livelli analitici che rispondono alle domande fondamentali: *WHY*, *WHAT* e HOW, corrispondenti rispettivamente ai livelli operativo, funzionale e architetturale. [13]

Il livello operazionale (WHY) analizza il contesto in cui il sistema dovrà operare, identificando missioni, attori, scenari e bisogni; è qui che si definiscono gli obiettivi del sistema in risposta alle esigenze del business. In questa fase, il System Of Interest (SOI) viene analizzato in relazione ai

bisogni dell'utente e si identificano gli attori esterni, si descrivono le interazioni tramite diagrammi di casi d'uso (fig.16) e diagrammi di sequenza, e si definiscono le attività operative attraverso diagrammi di attività. [13,15]

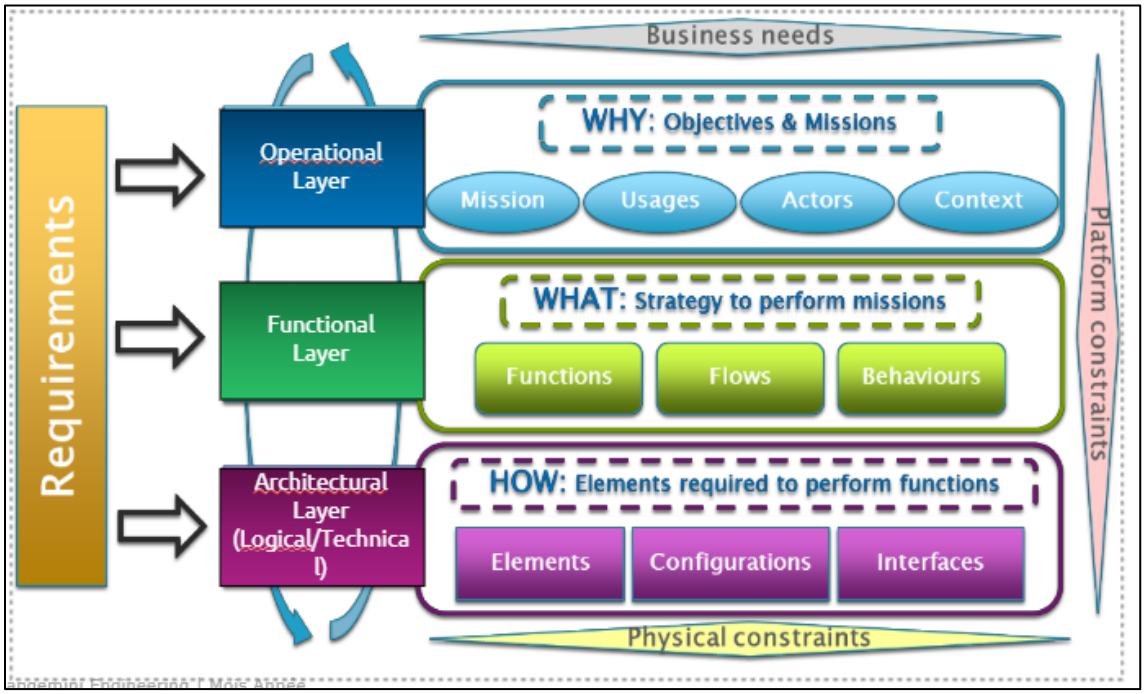

Fig.15 MBSE Aproach [13]

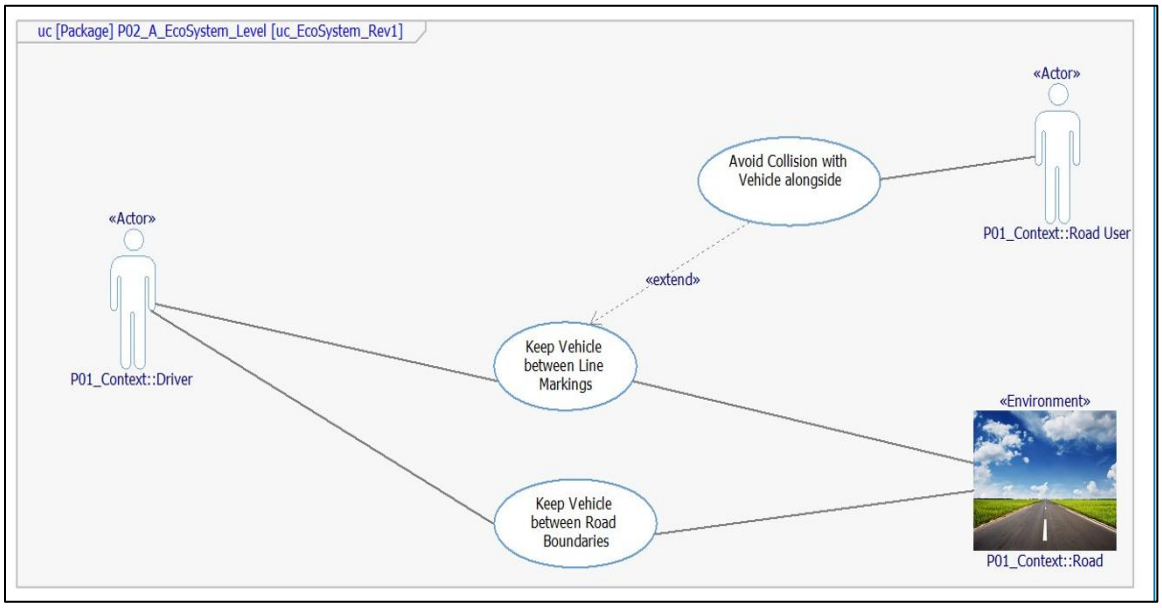

Fig.16 esempio di use case diagram [13]

Il livello funzionale (WHAT) traduce questi obiettivi in funzioni, flussi e comportamenti; si definisce l'architettura funzionale del sistema, modellando le interazioni tra le funzioni e la loro organizzazione logica. Le funzioni vengono rappresentate tramite *Block Definition Diagrams (BDD) (fig.17)*, mentre le interconnessioni funzionali sono modellate con Internal Block Diagrams (IBD). (Nota: tutti i diagrammi citati da qui a breve verranno illustrati nel capitolo dedicato al linguaggio SysML). [13,15]

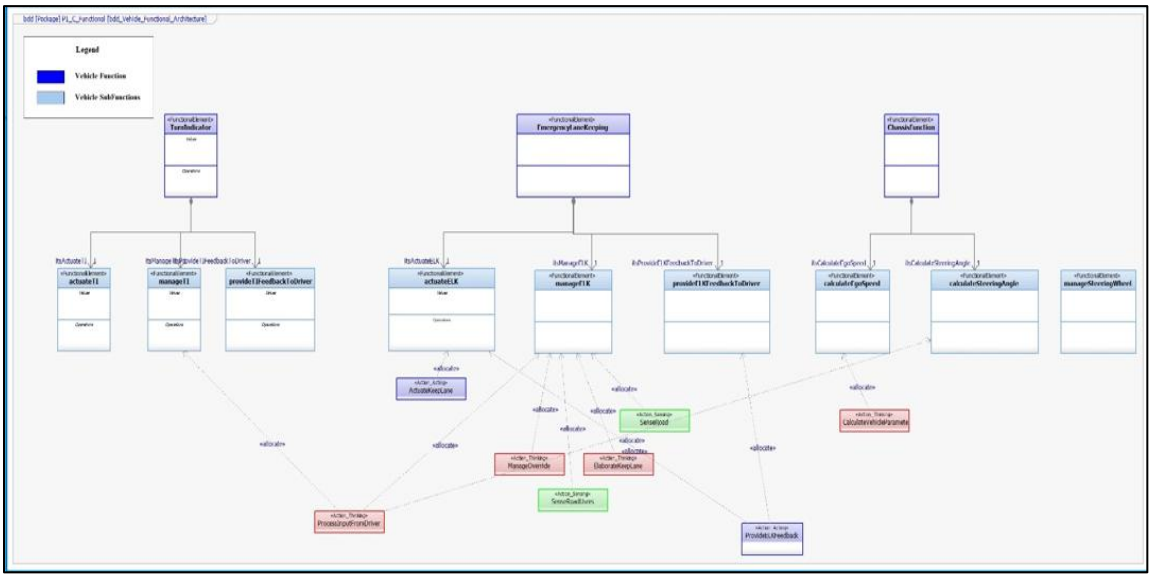

Fig.17 esempio di body block diagram [13]

Il livello logico (HOW) rappresenta la prima parte del livello architetturale. In questa fase, il SOI viene scomposto in elementi logici, ovvero componenti astratti che implementano le funzioni precedentemente identificate. Si definisce l'architettura logica tramite BDD, si allocano le funzioni agli elementi logici, si modellano le interconnessioni logiche con IBD e si descrive il comportamento del sistema con diagrammi di sequenza. [13,15]

Infine, il livello tecnico o fisico (HOW), rappresenta la concretizzazione del sistema; in questo livello, per ciascun elemento logico definito in precedenza, si identificano i corrispondenti elementi fisici e tecnici che compongono il SOI nella realtà, con l'obiettivo di definire l'architettura tecnica del sistema, ovvero la sua realizzazione concreta; quindi, è inclusa l'identificazione dei componenti fisici, delle interfacce reali e delle connessioni tecniche tra gli elementi. I diagrammi utilizzati in questa fase includono Block Definition Diagrams e Internal Block Diagrams, che descrivono la struttura fisica e le interazioni tra i componenti. Questo livello è fondamentale per garantire che le soluzioni progettuali siano realizzabili e coerenti con i vincoli tecnologici e di piattaforma, inoltre, consente di completare la tracciabilità verticale dei requisiti, collegando le esigenze iniziali del business con le soluzioni tecniche implementate. [13,15]

In sintesi, l'MBSE non è solo una tecnica di modellazione, ma un framework metodologico completo che guida lo sviluppo di sistemi complessi attraverso una sequenza logica e coerente di analisi: dal perché al cosa, fino al come. Ogni livello ha un ruolo specifico e strumenti dedicati per garantire coerenza, tracciabilità con le specifiche e qualità lungo tutto il ciclo di vita del sistema. [17]

CAPITOLO 4 Metodo e Linguaggio

4.1 MagicGrid

Nel contesto dell'Ingegneria dei Sistemi Basata su Modelli (MBSE), la teoria acquista valore solo quando è accompagnata da istruzioni chiare e strumenti pratici. Il framework MagicGrid, sviluppato da CATIA® No Magic, nasce proprio con questo obiettivo: fornire un approccio metodologico completo, concreto e adattabile per la modellazione di sistemi complessi tramite il linguaggio SysML.[17]

MagicGrid si distingue per la sua capacità di integrare tre componenti fondamentali dell'MBSE: il metodo, il linguaggio e lo strumento di modellazione. È compatibile con SysML "vanilla", non richiede estensioni e può essere utilizzato con qualsiasi tool che supporti SysML, anche se la sua piena efficacia si realizza attraverso l'uso della suite CATIA Magic, in particolare Cameo Systems Modeler con il plugin Cameo DataHub. [17]

Il framework si presenta come una matrice strutturata in stile Zachman, che guida l'ingegnere lungo tutto il processo di modellazione, rispondendo a domande fondamentali come: "Come organizzare il modello?", "Qual è il workflow?", "Quali artefatti produrre in ogni fase?". Come mostrato nella Figura 69, la matrice è organizzata in domini (Problema, Soluzione e Implementazione) e in aspetti (Requisiti, Struttura, Comportamento, Parametrici e Sicurezza & Affidabilità); ogni cella rappresenta una vista del sistema, associata ad artefatti come diagrammi, tabelle o mappe, che descrivono il sistema da diverse prospettive e livelli di dettaglio. [17]

Nel dominio del Problema (fig.69), la distinzione tra prospettiva black-box e white-box permette di analizzare il sistema sia dal punto di vista esterno (Stakeholder Needs), sia interno, attraverso elementi come il contesto del sistema, i sottosistemi concettuali, i casi d'uso, l'analisi funzionale e le misure di efficacia (MoEs). Viene inoltre introdotta l'analisi dei guasti (FMEA) già a livello concettuale, a dimostrazione dell'approccio sistemico e preventivo del framework. [17]

Nel dominio della Soluzione (fig.69), la tabella mostra come i requisiti vengano declinati a livello di sistema, sottosistema e componente, ciascuno con la propria struttura, comportamento, parametri e aspetti di sicurezza e affidabilità. Questo livello di dettaglio consente di costruire un'architettura multilivello del sistema, mantenendo coerenza e tracciabilità tra le diverse viste.

Infine, il dominio dell'Implementazione (fig.69) si concentra sulla definizione dei requisiti implementativi, che rappresentano il ponte tra la modellazione e la realizzazione fisica del sistema. Sebbene le attività di implementazione non rientrino direttamente nell'ambito dell'MBSE, MagicGrid ne riconosce l'importanza come fase conclusiva del processo ingegneristico.

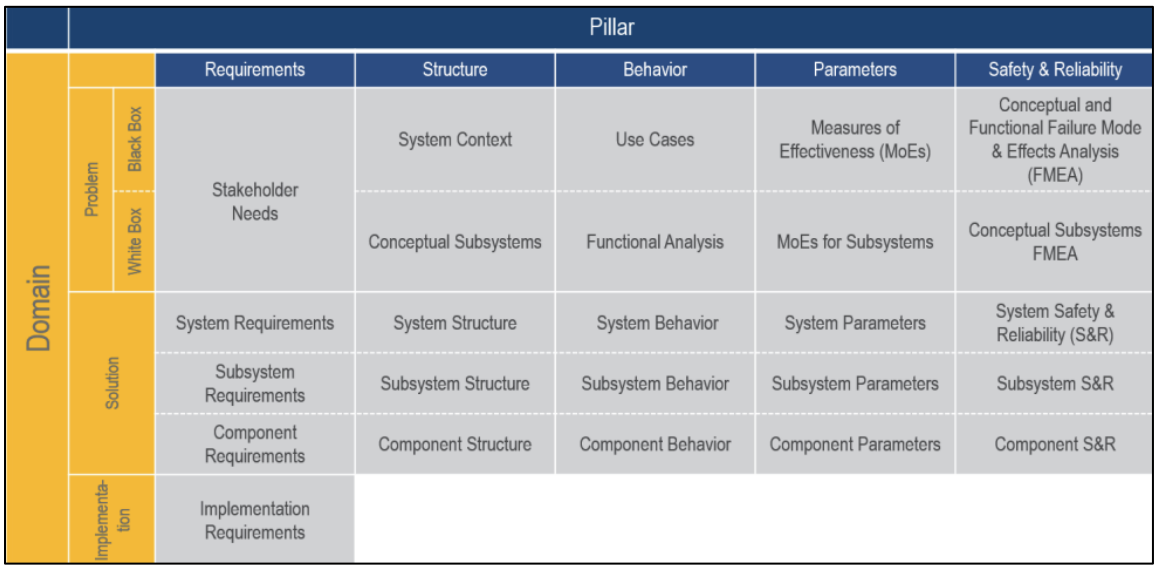

Fig.69 Framework MagicGrid [17]

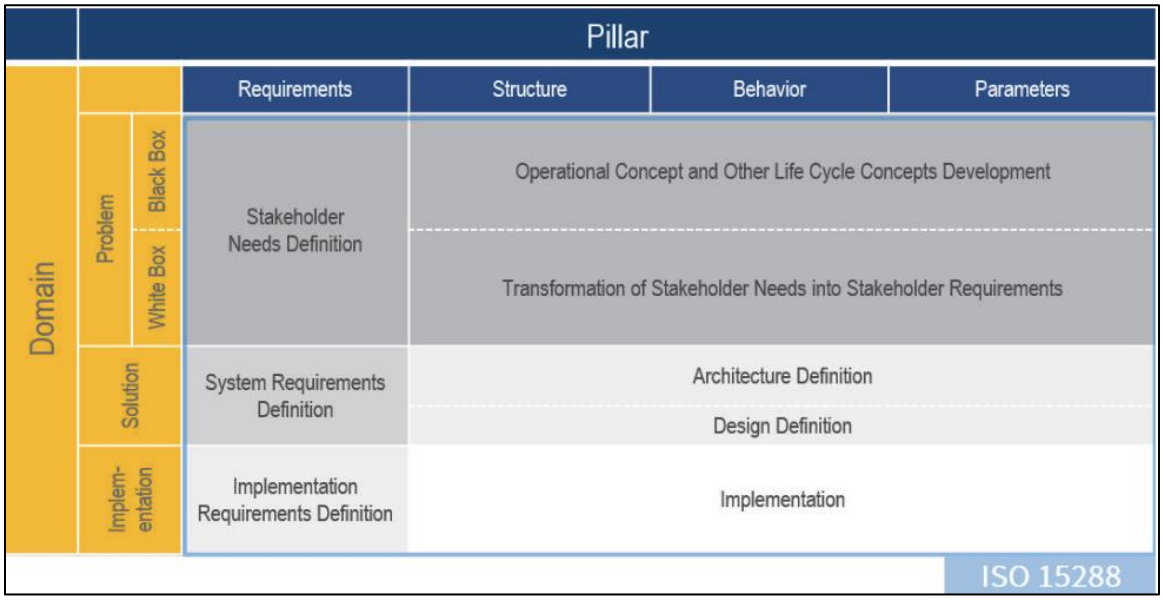

Fig.70 allineamento MagicGrid/ISO 15288

A rafforzare la validità del framework, la Figura 70 mostra l'allineamento di MagicGrid ai processi tecnici definiti dalla norma ISO/IEC/IEEE 15288, che regolano il ciclo di vita dell'ingegneria dei sistemi. Ogni attività (dalla definizione delle esigenze degli stakeholder, alla trasformazione in requisiti, fino alla progettazione architetturale e alla realizzazione) trova corrispondenza diretta nelle celle della matrice MagicGrid. Questo garantisce una piena tracciabilità tra le fasi del processo ingegneristico e gli artefatti prodotti, assicurando coerenza, qualità e conformità agli standard internazionali.[17]

In sintesi, MagicGrid non è solo una guida teorica, ma uno strumento operativo che consente di strutturare, organizzare e documentare in modo rigoroso e sistematico la modellazione di sistemi complessi. La sua applicazione rende l'approccio MBSE realmente efficace, adattabile e pronto per affrontare le sfide dei sistemi moderni.

4.2 Linguaggio

In ambito MBSE esistono linguaggi specifici pensati per rappresentare in modo strutturato e visivo i diversi aspetti di un sistema complesso, due tra i più rilevanti sono UML e SysML. [15]

UML, acronimo di Unified Modeling Language, è un linguaggio orientato agli oggetti, ampiamente utilizzato per modellare sistemi software; si tratta di un linguaggio semi-formale e si basa su diagrammi che permettono di specificare, visualizzare, documentare e modificare gli artefatti software, inoltre è molto utile per rappresentare classi, interazioni, stati e attività all'interno di un sistema software.[15]

Tuttavia, quando si passa dalla modellazione del solo software alla modellazione di sistemi complessi multidisciplinari (che includono hardware, elettronica, meccanica, ecc.), UML da solo non è sufficiente, per questo motivo è stato sviluppato SysML (Systems Modeling Language), un'estensione di UML pensata appositamente per l'ingegneria dei sistemi.[15]

SysML riutilizza una parte di UML, ma introduce anche nuovi concetti fondamentali per la modellazione di sistemi, come:

- Blocchi (Blocks) per rappresentare componenti fisici o logici,
- Flussi di oggetti (Item Flows) per descrivere il trasferimento di materiali o segnali,
- Proprietà di valore e parametriche per modellare aspetti quantitativi e vincoli,
- Requisiti per la gestione e la tracciabilità delle specifiche,
- Allocazioni per definire relazioni tra elementi funzionali e fisici.

Nel mondo della modellazione, UML non è solo un linguaggio, ma un vero e proprio ecosistema di diagrammi pensati per rappresentare in modo visivo e strutturato i diversi aspetti di un sistema software. Con l'evoluzione alla versione UML 2.0, il linguaggio si è arricchito per coprire una porzione ancora più ampia del ciclo di sviluppo, includendo anche pratiche agili e migliorando l'integrazione tra modelli strutturali e comportamentali.[15]

I diagrammi UML sono suddivisi in tre grandi categorie (fig.18) [18]:

- Diagrammi Strutturali, rappresentano la struttura statica del sistema: classi, componenti, pacchetti, oggetti, strutture composite e distribuzione.
- Diagrammi Comportamentali, descrivono il comportamento generale del sistema: casi d'uso, attività e macchine a stati.

- Diagrammi di Interazioni, focalizzati sulle interazioni tra elementi: sequenze, comunicazioni, panoramiche di interazione e temporizzazioni.

Questa varietà di diagrammi consente di modellare un sistema da più prospettive, facilitando la comunicazione tra stakeholder e migliorando la tracciabilità tra requisiti, architettura e comportamento. [18]

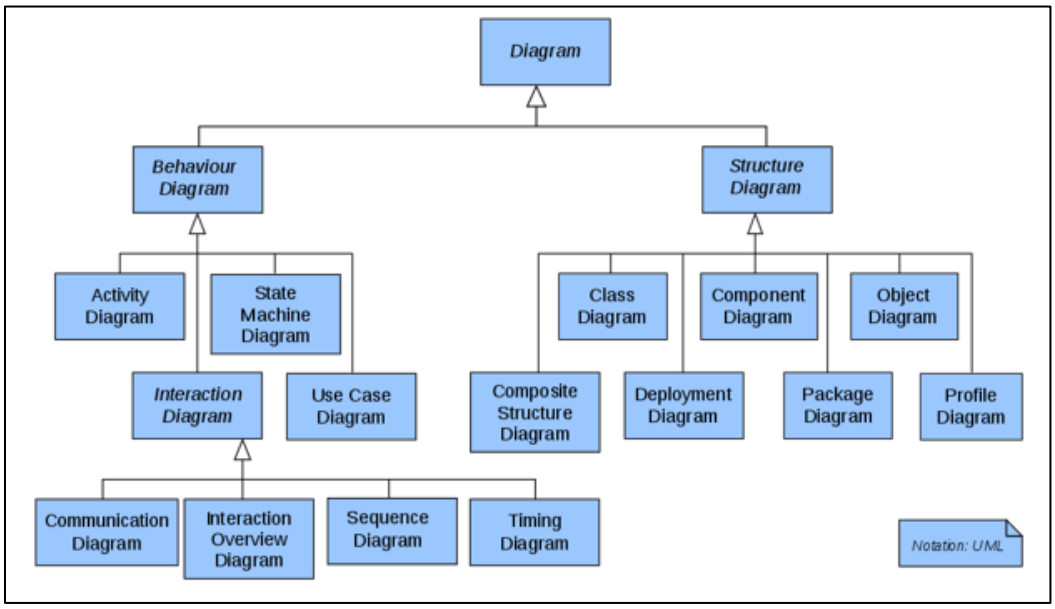

Fig.18 UML Diagrams, modificata da [32]

Quando si passa a SysML, molti di questi diagrammi vengono riutilizzati o adattati per rappresentare sistemi complessi, non solo software, ad esempio, i diagrammi di stato diventano fondamentali per descrivere il comportamento di componenti fisici, mentre i diagrammi di attività possono essere usati per modellare flussi operativi o logiche di controllo. [18]

4.2.1 SYSML

SysML è un linguaggio di modellazione visuale specificamente progettato per l'ingegneria dei sistemi. A differenza di UML, che nasce per il software, SysML è pensato per rappresentare sistemi complessi che possono includere componenti software, hardware, meccanici, elettrici e persino organizzativi. [19]

È importante sottolineare che SysML non è una metodologia, ovvero non prescrive come condurre un progetto o quali fasi seguire, inoltre non è nemmeno uno strumento software ma un linguaggio, ovvero un insieme di semantiche (il significato degli elementi) e notazioni (la loro rappresentazione grafica). Gli strumenti come Cameo, MagicDraw o Enterprise Architect sono quelli che implementano SysML e permettono di usarlo in pratica. [19]

SysML è anche compatibile ed estende UML, riutilizzandone alcune parti e aggiungendo elementi specifici per la modellazione di sistemi, come i blocchi, i requisiti, i parametri e i vincoli.

Questo linguaggio è stato concepito per supportare l'intero ciclo di vita di un sistema, dalla definizione del concetto operativo (OpsCon) fino alla specifica, analisi, progettazione, verifica e validazione; questo lo rende particolarmente adatto per sistemi complessi e system-of-systems, che coinvolgono non solo software e hardware, ma anche processi, persone, strutture e informazioni. [19]

Uno degli obiettivi principali di SysML è quello di integrare i diversi strumenti e linguaggi che gli ingegneri di sistema utilizzano in un progetto, in altre parole, cerca di fare per l'ingegneria dei sistemi ciò che UML ha fatto per il software: unificare e standardizzare la modellazione, riducendo la frammentazione tra strumenti e domini.[19]

Questa capacità di integrazione è particolarmente importante in contesti multidisciplinari, dove team diversi lavorano su aspetti differenti dello stesso sistema. SysML fornisce un linguaggio comune che consente di mantenere coerenza, tracciabilità e comunicazione efficace tra tutte le parti coinvolte. [19]

SysML si fonda su quattro pilastri fondamentali che permettono di modellare un sistema in modo completo, multidisciplinare e coerente; infatti, questi pilastri rappresentano le diverse prospettive da cui è possibile descrivere un sistema, e ciascuno ha un ruolo specifico nel ciclo di vita del progetto.[19]

Il primo pilastro è la struttura, dove si descrive come è fatto il sistema, cioè la sua architettura statica. Si utilizzano diagrammi come il Block Definition Diagram, che mostra la gerarchia e la classificazione dei componenti, e l'Internal Block Diagram, che rappresenta le connessioni interne tra le parti, le porte e i connettori. Questi strumenti permettono di visualizzare chiaramente la composizione del sistema e le sue interfacce (tutti i diagrammi nominati verranno descritti in un sottocapitolo dedicato). [19]

Il secondo pilastro è il comportamento, che tratta cosa fa il sistema, come reagisce agli stimoli e come si evolve nel tempo. SysML offre diversi diagrammi per modellare il comportamento:

- I diagrammi di attività descrivono il flusso delle operazioni.
- I diagrammi di stato rappresentano le transizioni tra stati del sistema.
- I diagrammi di sequenza mostrano le interazioni temporali tra componenti.

Questi strumenti sono fondamentali per analizzare la logica operativa e le dinamiche interne del sistema.[19]

Il terzo pilastro è la gestione dei requisiti; a tal scopo SysML consente di modellare i requisiti in modo esplicito, collegandoli agli elementi strutturali e comportamentali, permettendo così di garantire tracciabilità, verifica e conformità rispetto alle specifiche iniziali. I requisiti possono essere funzionali, prestazionali, normativi, e vengono rappresentati in modo visuale e integrato nel modello. [19]

Il quarto pilastro è la modellazione parametrica, dove si entra nel dettaglio delle relazioni matematiche tra le proprietà dei componenti. I diagrammi parametrici permettono di esprimere vincoli, formule e dipendenze, utili per analisi ingegneristiche, simulazioni e ottimizzazioni. Questo pilastro collega la parte concettuale del modello con quella quantitativa, rendendo possibile una valutazione più precisa del comportamento del sistema sotto certe condizioni.[19]

Insieme, questi quattro pilastri costituiscono la base di SysML e permettono di affrontare la modellazione di sistemi complessi in modo integrato, rigoroso e collaborativo.

4.2.2 SysML Diagrams

Nel linguaggio di modellazione SysML esistono una serie di diagrammi utilizzati per modellare rappresentati in figura 19

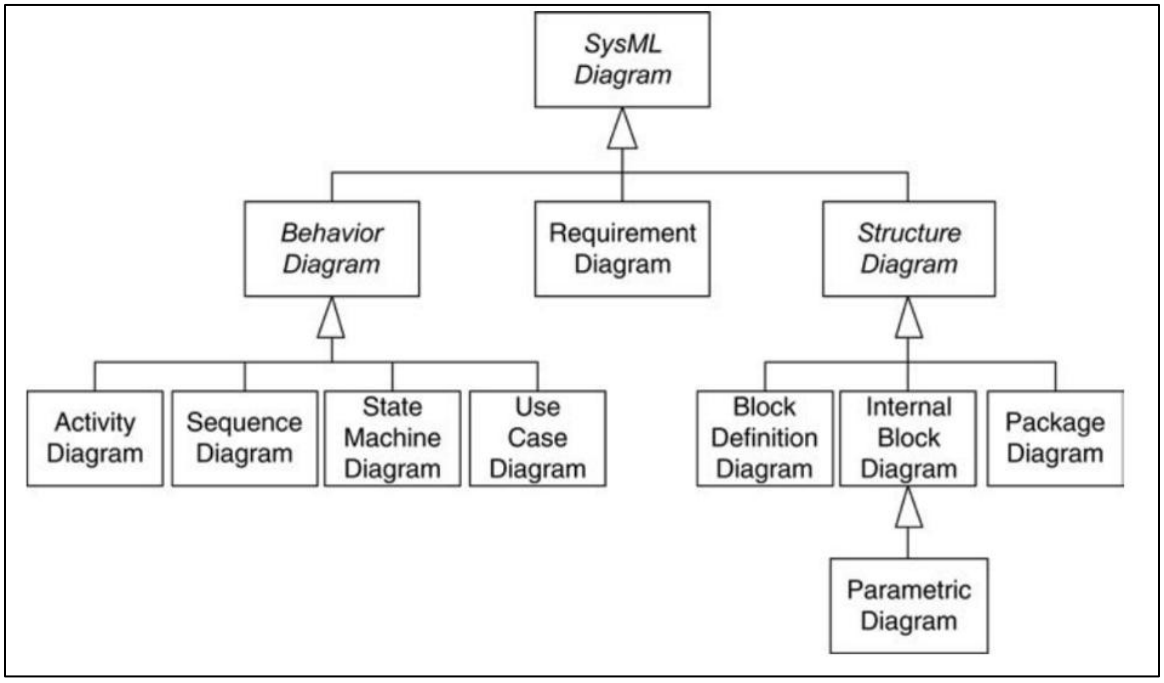

Fig.19 SysML hierarchy [19]

Al vertice della gerarchia troviamo il concetto generale di SysML Diagram, che si suddivide in tre categorie fondamentali:

- Behavior Diagram (Diagrammi comportamentali): Questi diagrammi descrivono il comportamento dinamico del sistema [18]. Includono:
  - Activity Diagram (act): rappresenta i flussi di attività e processi.
  - Sequence Diagram (seq): mostra le interazioni tra elementi nel tempo.
  - State Machine Diagram (stm): descrive gli stati di un oggetto e le transizioni tra essi.
  - Use Case Diagram (uc): evidenzia le funzionalità del sistema dal punto di vista dell'utente.

- Structure Diagram (Diagrammi strutturali): Questi diagrammi rappresentano l'architettura statica del sistema [18]. Comprendono:
  - Block Definition Diagram (bdd): definisce i blocchi e le loro relazioni.
  - Internal Block Diagram (ibd): mostra la struttura interna di un blocco, e da esso deriva il:
  - Parametric Diagram (par): utilizzato per esprimere vincoli matematici e relazioni tra parametri.
  - Package Diagram (pkg): organizza il modello in pacchetti logici.
- Requirement Diagram (req): Questo tipo di diagramma è fondamentale per tracciare, analizzare e verificare i requisiti del sistema, assicurando che ogni parte del modello risponda a specifiche esigenze [18].

Figura 19 ci offre una visione chiara e ordinata della struttura concettuale di SysML, evidenziando come ogni tipo di diagramma contribuisca a una rappresentazione completa e coerente del sistema. È uno strumento essenziale per ingegneri di sistema, progettisti e analisti che lavorano su progetti complessi e multidisciplinari.

4.2.2.1 Block Definition Diagram

l Block Definition Diagram (bdd) rappresenta uno degli strumenti fondamentali offerti dal linguaggio SysML per la modellazione della struttura di un sistema, poiché consente di descrivere in modo chiaro, organizzato e formalmente coerente gli elementi costitutivi del sistema e le relazioni strutturali che intercorrono tra essi, risultando particolarmente efficace nella progettazione di sistemi estensibili e flessibili, in quanto permette di anticipare e gestire con maggiore efficienza le modifiche progettuali che possono emergere nel tempo a seguito dell'evoluzione delle esigenze degli stakeholder [19].

Gli elementi rappresentati all'interno di un bdd, come blocchi, attori e altri componenti, sono definiti elementi di definizione, poiché costituiscono la base concettuale su cui si fonda l'intero modello del sistema; a questi si affiancano le relazioni strutturali, che svolgono un ruolo cruciale nel determinare la decomposizione gerarchica del sistema, partendo dal livello più alto fino ai singoli sottocomponenti [19].

L'elemento centrale del bdd è il blocco, utilizzato per modellare qualsiasi tipo di entità all'interno del sistema, e un concetto chiave in SysML è la distinzione tra definizione di tipo e istanza di utilizzo: i blocchi, i tipi di valore e i blocchi vincolati rappresentano definizioni, mentre le proprietà di parte, di valore e di vincolo rappresentano istanze, e questa distinzione si riflette anche nella notazione grafica, dove gli elementi di definizione sono identificati da un solo nome, mentre quelli di utilizzo presentano un nome seguito da due punti e dal tipo [19].

Graficamente, un blocco è rappresentato da un rettangolo preceduto dallo stereotipo «block» e può includere compartimenti opzionali che descrivono le sue caratteristiche, le quali si suddividono in caratteristiche strutturali (come proprietà di parte, valore, riferimento, vincolo e porte) e

caratteristiche comportamentali (come operazioni e ricezioni), che definiscono i comportamenti associati al blocco e ne arricchiscono la descrizione funzionale [19].

Un altro aspetto fondamentale del bdd riguarda le relazioni tra blocchi, che possono essere di diversi tipi e che contribuiscono a definire la struttura e l'organizzazione del sistema:

- Le associazioni di riferimento indicano una connessione tra istanze di blocchi e sono rappresentate da una linea continua, con direzione dell'accesso unidirezionale (freccia aperta) o bidirezionale (nessuna freccia).[19]
- Le associazioni composite esprimono una relazione di composizione strutturale, dove un blocco composito è costituito da una o più istanze di altri blocchi, rappresentata da una linea continua con un diamante pieno sul lato del composito.[19]
- Le relazioni di generalizzazione stabiliscono un legame di ereditarietà tra un supertipo e un sottotipo, dove il sottotipo eredita le caratteristiche del supertipo e può estenderle, rappresentate da una freccia triangolare cava rivolta verso il supertipo.[19]
- Le relazioni di dipendenza indicano che un elemento (cliente) dipende da un altro (fornitore), e se il fornitore subisse modifiche, anche il cliente potrebbe dover essere aggiornato [19].

Il bdd si configura quindi come uno strumento essenziale per rappresentare sia la struttura interna del sistema sia le sue interazioni con l'ambiente esterno, e grazie alla possibilità di definire gerarchie di tipi attraverso la generalizzazione, favorisce la riusabilità e l'astrazione, rendendo il sistema più modulare e adattabile; questo approccio consente inoltre di separare l'utilizzo di un servizio dalla sua implementazione concreta, facilitando l'evoluzione del sistema con un impatto minimo sulla sua architettura complessiva [19].

4.2.2.2 Internal Block Diagram

L'Internal Block Diagram (ibd) rappresenta uno degli strumenti fondamentali del linguaggio SysML per la modellazione strutturale di un sistema, ed è strettamente collegato al Block Definition Diagram (bdd), da cui si distingue per finalità e livello di dettaglio, poiché mentre il bdd fornisce una visione generale e astratta della struttura del sistema, l'ibd si concentra sulla rappresentazione della struttura interna di un singolo blocco, mostrando come le sue parti interagiscono tra loro e con l'ambiente esterno [19].

A differenza del bdd, che descrive i blocchi come tipi, l'ibd visualizza le istanze di quei blocchi, ovvero le proprietà di parte e di riferimento del blocco specificato nell'intestazione del diagramma, permettendo così di rappresentare in modo più concreto e dettagliato le connessioni tra le parti, i flussi che le attraversano (materia, energia o dati), e i servizi che vengono forniti o richiesti, offrendo una comprensione più operativa di come le parti devono essere assemblate per ottenere un'istanza valida del blocco e come questa si integra nel sistema complessivo [19].

È importante sottolineare che l'ibd, pur offrendo una rappresentazione dettagliata della struttura interna, non è pensato per modellare aspetti geometrici o spaziali, per i quali è necessario ricorrere

a strumenti specifici come i software CAD, mentre l'ibd si concentra su relazioni logiche e funzionali, risultando particolarmente utile nella fase di progettazione concettuale [19].

La costruzione di un ibd parte solitamente dalla definizione di un blocco nel bdd, e una volta stabilite le proprietà strutturali e comportamentali del blocco, si procede con la creazione dell'ibd per rappresentarne una configurazione concreta, comprensiva delle connessioni interne, e sebbene i due diagrammi offrano prospettive diverse, sono complementari e vengono spesso sviluppati in parallelo durante il ciclo di vita del sistema, rispondendo alle esigenze di stakeholder differenti [19].

Nel contesto di un IBD, le proprietà di parte rappresentano componenti interne al blocco e sono visualizzate come rettangoli a bordo pieno, con una notazione del tipo: <nomeParte>: <Tipo> [<molteplicità>], mentre le proprietà di riferimento rappresentano elementi esterni ma necessari al funzionamento del blocco, sono rappresentate con rettangoli a bordo tratteggiato e seguono la stessa notazione, e in entrambi i casi la molteplicità può essere indicata anche nell'angolo superiore destro del rettangolo [19].

Le connessioni tra le proprietà sono rappresentate da connettori (pin), che indicano la possibilità di interazione tra le parti, e ogni connettore può essere etichettato con un nome e un tipo, dove il tipo deve corrispondere a un'associazione già definita nel modello, consentendo così di specificare il mezzo attraverso cui avviene la comunicazione tra le parti [19].

Un ulteriore elemento distintivo dell'ibd è la rappresentazione dei flussi di elementi, che modellano il passaggio di materia, energia o dati tra le strutture del sistema, e tali flussi sono rappresentati da frecce triangolari piene poste sui connettori, accompagnate da un'etichetta che ne indica il tipo, il quale deve essere compatibile con le porte di flusso alle estremità del connettore [19].

In conclusione, l'Internal Block Diagram si configura come uno strumento essenziale per la progettazione di sistemi complessi, in quanto consente di visualizzare in modo dettagliato e concreto le interazioni tra le parti di un blocco, integrando e completando le informazioni fornite dal ibd, e offrendo una prospettiva operativa che facilita la comprensione del funzionamento interno del sistema, rendendolo particolarmente utile per tutti gli stakeholder coinvolti, migliorando la comunicazione tra progettisti, ingegneri e utenti finali, e contribuendo a una progettazione più efficace e condivisa [19].

4.2.2.3 Use Case Diagram

La definizione dei casi d'uso costituisce una fase fondamentale nella progettazione dei sistemi, poiché consente di delineare sin dalle prime fasi del progetto i servizi che il sistema deve offrire e le modalità con cui le entità esterne interagiscono con esso; in questo contesto, il linguaggio SysML fornisce uno strumento efficace, lo Use Case Diagram (uc), che permette di rappresentare in maniera sintetica e intuitiva i comportamenti osservabili del sistema, offrendo una visione a scatola nera focalizzata sulle funzionalità disponibili nel contesto operativo [19].

Questo tipo di diagramma risulta particolarmente utile durante le fasi iniziali del ciclo di vita del sistema, come l'analisi dei requisiti o la definizione del Concetto Operativo (ConOps), poiché consente di descrivere le funzionalità in modo visivo e orientato agli obiettivi, sostituendo in alcune metodologie i requisiti funzionali testuali e facilitando così la comprensione da parte degli stakeholder. Successivamente, i casi d'uso vengono analizzati dagli architetti di sistema per essere scomposti e allocati ai sottosistemi o ai componenti, contribuendo alla definizione dell'architettura [19].

Nel processo di definizione dei casi d'uso, è essenziale considerare che ciascun caso rappresenta un comportamento osservabile del sistema, il cui nome deve essere formulato come una frase verbale che descrive un'azione, coerente con l'obiettivo dell'attore che lo attiva; non tutti i comportamenti del sistema rientrano in questa categoria, ma solo quelli che possono essere influenzati o attivati da attori esterni, i quali possono essere persone o altri sistemi, e si distinguono in attori primari, che avviano il caso d'uso, e attori secondari, che vi partecipano senza avviarlo, con la possibilità che un attore ricopra entrambi i ruoli [19].

È importante che ogni caso d'uso rifletta un obiettivo specifico dell'attore, e che la sua denominazione sia formulata dal punto di vista di quest'ultimo, evitando ambiguità semantiche; ad esempio, se l'operatore deve inviare un comando, il caso d'uso sarà denominato "Invia comando" e non "Ricevi comando", poiché l'azione deve essere descritta dal punto di vista dell'attore che la compie; inoltre, occorre distinguere tra caso d'uso e scenario, dove il primo rappresenta un comportamento generale del sistema, mentre ogni singolo percorso di esecuzione all'interno di quel comportamento costituisce uno scenario distinto, che può includere sia il flusso principale di successo sia eventuali deviazioni, errori o eccezioni [19].

Una prassi consolidata prevede l'analisi di una specifica testuale o di un diagramma di attività relativo a un caso d'uso, da cui si derivano una serie di diagrammi di sequenza, ciascuno dedicato a uno scenario specifico; tali diagrammi non solo facilitano la visualizzazione del comportamento del sistema, ma possono anche essere utilizzati come casi di test, qualora vengano arricchiti con informazioni dettagliate sugli input e sugli output attesi [19].

Dal punto di vista grafico, un caso d'uso è rappresentato da un'ellisse contenente il nome del servizio, e può essere organizzato in gerarchie tramite relazioni di generalizzazione oppure incluso all'interno di altri casi d'uso, al fine di rappresentare funzionalità condivise; il confine del sistema è indicato da un rettangolo che racchiude i casi d'uso, distinguendo chiaramente ciò che è interno al sistema da ciò che è esterno [19].

Gli attori sono rappresentati graficamente e collegati ai casi d'uso tramite associazioni, che indicano la possibilità di invocare o partecipare a un comportamento del sistema; tali associazioni possono specificare la molteplicità, ma devono rispettare alcune regole formali, tra cui il divieto di creare associazioni tra attori, tra casi d'uso, o associazioni compositive tra attori e casi d'uso [19].

Un caso d'uso di base è direttamente collegato a un attore primario e rappresenta l'obiettivo principale dell'interazione, mentre i casi d'uso inclusi rappresentano funzionalità comuni che

vengono richiamate da altri casi d'uso e non sono attivati direttamente dagli attori, contribuendo così alla modularità e alla riusabilità del modello [19].

In sintesi, i diagrammi dei casi d'uso offrono una rappresentazione strutturata e comprensibile delle funzionalità del sistema dal punto di vista degli utenti e delle entità esterne, facilitando la comunicazione tra progettisti e stakeholder e costituendo una base solida per una progettazione coerente, orientata agli obiettivi e centrata sull'utente [19].

4.2.2.4 Activity diagrams

I diagrammi di attività (Activity Diagrams, act) rappresentano uno degli strumenti fondamentali messi a disposizione dal linguaggio SysML per descrivere il comportamento dinamico di un sistema, distinguendosi dai diagrammi di struttura che, al contrario, offrono una visione statica e architetturale; questi diagrammi si concentrano sulla sequenza temporale delle azioni, sulla logica di controllo e sul flusso di oggetti (che possono essere materia, energia o dati) che attraversano i processi del sistema, fornendo così una rappresentazione dettagliata e temporalmente ordinata delle operazioni [19].

Lo scopo principale di un diagramma di attività è quello di modellare comportamenti sia sequenziali sia concorrenti, mettendo in evidenza le modalità con cui le azioni si susseguono o si attivano in parallelo, e illustrando come le risorse vengano trasformate o trasferite tra le diverse fasi di un processo; tale strumento risulta particolarmente utile nelle fasi iniziali dell'analisi, quando è necessario comunicare con le parti interessate per definire in modo condiviso i requisiti funzionali e il comportamento atteso del sistema [19].

Uno degli aspetti più apprezzati dei diagrammi di attività risiede nella loro elevata leggibilità, che consente di rappresentare anche logiche di controllo complesse in maniera chiara e accessibile a un pubblico eterogeneo; l'introduzione dei nodi oggetto permette di modellare esplicitamente il flusso di informazioni o materiali, mentre le partizioni di attività (swimlanes) facilitano l'allocazione delle responsabilità delle azioni alle diverse componenti del sistema, contribuendo così alla chiarezza e alla tracciabilità del modello [19].

Nonostante la loro efficacia comunicativa, i diagrammi di attività presentano alcune limitazioni, tra cui l'impossibilità di rappresentare direttamente quale struttura del sistema invoca una determinata azione, aspetto che viene trattato in modo più esplicito nei diagrammi di sequenza; per tale motivo, i diagrammi di attività vengono spesso impiegati come strumenti di analisi e comunicazione piuttosto che come strumenti per la progettazione dettagliata o per la generazione automatica del codice [19].

Un concetto chiave per comprendere il funzionamento dei diagrammi di attività in SysML è quello di token, un'entità astratta che rappresenta il flusso di controllo o di oggetti attraverso un'attività; sebbene non sia visibile né rappresentato graficamente nel modello, il token costituisce la base semantica su cui si fondano le regole di esecuzione delle attività, in quanto ogni nodo e bordo

(edge) del diagramma è definito in termini di flusso di token, e l'attivazione di un'azione, la transizione tra stati o il trasferimento di dati avviene solo quando un token è presente e può fluire lungo il percorso definito [19].

Per visualizzare intuitivamente il concetto di token, si può immaginare quest'ultimo come una pedina che si muove lungo il diagramma seguendo le regole stabilite dalla logica di controllo; esistono due principali tipologie di token: i token oggetto, che rappresentano istanze di materia, energia o dati e sono associati a tipi definiti nel modello (come blocchi, segnali o tipi di valore), e i token di controllo, che non rappresentano alcun contenuto fisico o informativo ma servono esclusivamente a guidare l'attivazione delle azioni, indicando quando un'azione è abilitata all'esecuzione, permettendo anche l'esecuzione simultanea di più azioni [19].

La comprensione del flusso di token è essenziale per interpretare correttamente il comportamento rappresentato in un diagramma di attività, poiché elementi come i nodi di decisione consentono il passaggio del token solo lungo uno dei rami condizionati, i nodi di unione raccolgono token da percorsi alternativi, mentre i nodi fork e join gestiscono rispettivamente l'esecuzione parallela e la sincronizzazione dei flussi, rendendo possibile la modellazione di comportamenti complessi e distribuiti [19].

All'interno di un diagramma di attività, le azioni costituiscono le unità fondamentali di comportamento, ciascuna delle quali descrive un'attività, una trasformazione o un'elaborazione che il sistema deve eseguire; la notazione prevede l'uso di un rettangolo con angoli arrotondati contenente una descrizione testuale dell'azione, ed è buona prassi utilizzare frasi verbali semplici e dirette, evitando formulazioni complesse o ambigue, e suddividendo comportamenti articolati in azioni atomiche con obiettivi chiari [19].

Oltre al linguaggio naturale, SysML consente l'uso di espressioni opache (opaque expressions), che permettono di descrivere il comportamento di un'azione utilizzando linguaggi formali come C, Java, Verilog, Modelica o MATLAB; queste espressioni risultano particolarmente utili nelle fasi di progettazione tecnica, dove è richiesta una maggiore precisione nella definizione del comportamento, e contribuiscono alla formalizzazione e alla simulabilità del modello [19].

Le azioni sono collegate tra loro da archi direzionali, che definiscono la sequenza o la simultaneità con cui devono essere eseguite, e costituiscono la narrazione comportamentale dell'attività, rappresentando non solo l'ordine delle operazioni ma anche il flusso di oggetti tra le azioni e tra l'attività e il suo contesto esterno, rendendo il diagramma uno strumento potente per la descrizione del comportamento sistemico [19].

Un altro elemento fondamentale nei diagrammi di attività è il nodo oggetto, che consente di modellare il flusso di token oggetto all'interno dell'attività; ogni token oggetto rappresenta un'istanza di materia, energia o dati che transita tra le azioni, e i nodi oggetto sono rappresentati da rettangoli contenenti una stringa nel formato : [<molteplicità>], dove il tipo deve corrispondere a un elemento definito nel modello, e in presenza di entità complesse è possibile visualizzare compartimenti interni per mostrare le proprietà del token oggetto [19].

In alternativa ai nodi oggetto tradizionali, è possibile utilizzare i pin, ovvero nodi oggetto specializzati che vengono associati direttamente ai bordi delle azioni e rappresentati graficamente come piccoli quadrati; questa notazione compatta risulta particolarmente vantaggiosa nei diagrammi complessi, dove la chiarezza visiva è essenziale, e mantiene la stessa sintassi dei nodi oggetto, consentendo anche la definizione di pin opzionali mediante la specificazione di una molteplicità minima pari a zero e l'applicazione dello stereotipo «opzionale» [19].

Un'ulteriore estensione della modellazione del comportamento è costituita dai parametri di attività, che permettono di rappresentare gli input e gli output dell'intera attività e sono graficamente collegati alla cornice del diagramma; tali parametri seguono le stesse convenzioni dei pin e, per convenzione, vengono posizionati nella parte superiore o sinistra della cornice per gli input, e nella parte inferiore o destra per gli output, sebbene la direzione effettiva del flusso sia determinata dagli archi che li collegano [19].

I flussi di oggetti rappresentano i bordi attraverso cui transitano i token oggetto, ovvero istanze di materia, energia o dati, e modellano il trasferimento di informazioni o risorse tra le azioni, risultando particolarmente utili nei processi in cui il contenuto dei dati è determinante per l'esecuzione del comportamento; la notazione standard prevede una linea continua con freccia aperta che collega nodi oggetto, pin o parametri di attività, ma può anche connettersi a nodi di controllo come decisioni, unioni, fork e join, a condizione che i nodi collegati abbiano tipi compatibili, garantiti dall'identità dei tipi o da una relazione di generalizzazione che consente l'uso di sottotipi, favorendo così la riusabilità e l'estensibilità del sistema [19].

I flussi di controllo, invece, trasportano token di controllo che non rappresentano dati ma abilitano l'esecuzione delle azioni, modellando la sequenza logica tra le azioni soprattutto nei casi in cui non vi sia un flusso di oggetti che ne implichi l'ordine; SysML consente due notazioni per rappresentarli: una linea tratteggiata con freccia aperta, consigliata per distinguere visivamente i flussi di controllo, e una linea continua, utilizzata quando la notazione tratteggiata non è supportata dallo strumento; al termine di un'azione, viene offerto un token di controllo lungo il bordo in uscita, che abilita l'azione successiva, permettendo di modellare comportamenti sequenziali, paralleli o condizionati [19].

Tra le azioni disponibili, le Call Behavior Actions rivestono un ruolo centrale nella scomposizione gerarchica del comportamento, poiché permettono di richiamare un comportamento definito altrove nel modello, come un'altra attività, un'interazione o una macchina a stati; la notazione è identica a quella di un'azione base, ma il nome segue il formato : , e i pin associati devono corrispondere ai parametri dell'attività richiamata, garantendo la coerenza del flusso di token tra il comportamento chiamante e quello chiamato, con la possibilità che gli output generati siano immediatamente disponibili per le azioni successive, abilitando l'elaborazione parallela [19].

Le Send Signal Actions sono azioni specializzate per l'invio asincrono di istanze di segnale a una destinazione, risultando particolarmente utili nella modellazione di sistemi distribuiti e concorrenti;

queste azioni sono rappresentate da un pentagono convesso contenente il nome del segnale e, una volta abilitate, ricevono un token oggetto, generano un'istanza del segnale e la inviano, completandosi immediatamente senza attendere risposta, consentendo l'invio ripetuto del segnale ogni volta che l'azione riceve un nuovo token e migliorando così la reattività e la scalabilità del sistema [19].

Complementari alle azioni di invio sono le Accept Event Actions, che modellano il punto in cui un'attività attende l'occorrenza di un evento esterno prima di proseguire; l'evento può essere un segnale o un evento temporale, e la notazione è un pentagono concavo, simile a un rettangolo con una tacca triangolare; quando abilitata, l'azione resta in attesa dell'evento specificato e si completa solo al suo verificarsi, mentre se l'evento è già avvenuto, l'azione si conclude immediatamente; in assenza di bordi in ingresso, l'azione può essere attiva fin dall'inizio dell'attività, pronta a reagire a ogni occorrenza dell'evento, rendendola ideale per modellare comportamenti reattivi continui [19].

Un'estensione delle azioni di ricezione evento è rappresentata dalle Time Event Actions, che attendono il verificarsi di un evento temporale anziché un segnale; la loro notazione è una clessidra stilizzata, accompagnata da un'espressione temporale che può essere assoluta (con la parola chiave at) o relativa (con after), e quando l'azione viene abilitata, inizia a monitorare il tempo e si completa al raggiungimento del momento specificato, consentendo di modellare ritardi temporizzati o comportamenti periodici, come letture cicliche di sensori, e rendendo possibile la riattivazione automatica dell'azione a intervalli regolari [19].

Oltre alle azioni e ai nodi oggetto, i nodi di controllo costituiscono un elemento essenziale nei diagrammi di attività, poiché permettono di indirizzare il flusso di esecuzione lungo percorsi alternativi, paralleli o condizionati; SysML definisce sette tipi principali di nodi di controllo:

- il nodo iniziale (cerchio pieno) che segna l'avvio dell'attività;
- il Flow Final Node (cerchio con una X) che termina un singolo flusso senza interrompere l'intera attività;
- l'Activity Final Node (cerchio con un cerchio pieno interno) che termina l'intera attività anche se altri flussi sono ancora attivi;
- il nodo decisionale (rombo con guardie booleane) che dirige il flusso in base a condizioni logiche;
- il nodo di unione (merge, anch'esso un rombo) che raccoglie token da percorsi alternativi e li inoltra in un unico flusso;
- il nodo fork (barra nera spessa) che duplica un token in più flussi paralleli;
- il nodo join (anch'esso una barra nera) che sincronizza flussi paralleli rilasciando un token solo quando tutti gli ingressi sono attivi [19].

Un ulteriore elemento che arricchisce la semantica dei diagrammi di attività è l'uso delle partizioni di attività, o swimlanes, che permettono di associare le azioni alle strutture del sistema; ogni partizione rappresenta un elemento del modello, come un blocco o una proprietà di parte, ed evidenzia chi o cosa esegue ciascuna azione, aggiungendo una dimensione strutturale alla vista

comportamentale e migliorando la tracciabilità tra comportamento e architettura, soprattutto nei sistemi distribuiti o multi-componente [19].

In conclusione, i diagrammi di attività in SysML offrono una rappresentazione ricca, espressiva e leggibile del comportamento di un sistema, e grazie alla loro capacità di modellare flussi sequenziali e paralleli, comunicazioni asincrone, eventi temporali, logiche di controllo complesse e allocazioni strutturali, si rivelano strumenti estremamente efficaci per:

- analizzare e comunicare i requisiti comportamentali;
- modellare scenari operativi realistici;
- supportare la progettazione di sistemi distribuiti e reattivi;
- favorire la collaborazione tra stakeholder tecnici e non tecnici;

In definitiva, il diagramma di attività rappresenta un ponte tra la logica del comportamento e l'architettura del sistema, contribuendo in modo decisivo alla comprensione, validazione e progettazione di sistemi complessi [19].

4.2.2.5 State machine diagrams

Il diagramma macchina a stati (State Machine Diagram, stm) rappresenta uno degli strumenti fondamentali offerti dal linguaggio SysML per la modellazione del comportamento dinamico di un sistema, distinguendosi dai diagrammi di attività e di sequenza che si focalizzano rispettivamente sul flusso di controllo e sulle interazioni tra entità; il diagramma macchina a stati, invece, si concentra sulle transizioni di stato che una struttura subisce in risposta a eventi che si verificano nel tempo, offrendo così una rappresentazione rigorosa e deterministica del comportamento reattivo del sistema [19].

Questo tipo di diagramma si rivela particolarmente efficace per descrivere il comportamento interno di un blocco, noto come comportamento di classificazione, che ha inizio con la creazione dell'istanza del blocco e termina con la sua eliminazione; sebbene sia tecnicamente possibile associare una macchina a stati anche a una singola operazione o ricezione, nella pratica questa modalità è meno diffusa. I diagrammi macchina a stati possono essere impiegati a qualsiasi livello della gerarchia del sistema, dal sistema complessivo fino ai singoli componenti, e in qualsiasi fase del ciclo di vita, dalla definizione concettuale iniziale alla progettazione dettagliata [19].

In SysML, gli stati rappresentano condizioni operative del sistema e si distinguono in tre categorie principali:

- Stati semplici, rappresentati da rettangoli con angoli arrotondati, che possono includere comportamenti interni suddivisi in entry, exit e do. I primi due sono eseguiti in modo atomico all'ingresso e all'uscita dallo stato, mentre il comportamento do è continuo e può essere interrotto da eventi esterni [19]

- Stati compositi, che estendono gli stati semplici includendo sottostati nidificati. Quando uno stato composito è attivo, è attivo anche uno dei suoi sottostati. Le transizioni possono avvenire sia tra sottostati interni sia tra lo stato composito e altri stati esterni [19]
- Stati finali, che indicano la conclusione del comportamento della macchina a stati. La loro presenza è facoltativa, ma utile per esplicitare la terminazione del ciclo comportamentale. [19]

Le transizioni rappresentano il passaggio da uno stato a un altro in risposta a un evento, e possono anche avvenire all'interno dello stesso stato (auto-transizioni) [19]. Ogni transizione può essere arricchita da tre elementi opzionali:

- Il trigger, ovvero l'evento che attiva la transizione;
- La guardia, un'espressione booleana che deve risultare vera affinché la transizione possa avvenire;
- L'effetto, il comportamento eseguito durante la transizione.

Quando una transizione viene attivata, si verifica una sequenza ordinata e atomica di comportamenti: prima viene eseguito il comportamento di uscita dallo stato di origine, poi l'effetto della transizione, infine il comportamento di ingresso nello stato di destinazione; questa sequenza viene completata interamente prima che la macchina a stati possa elaborare nuovi eventi, garantendo così coerenza e prevedibilità nel modello [19].

Gli eventi costituiscono elementi centrali nella modellazione del comportamento reattivo, e SysML ne distingue quattro tipi principali:

- Eventi di segnale, che rappresentano la ricezione di un segnale da parte di una struttura predisposta [19]
- Eventi di chiamata, che modellano la ricezione di una richiesta di esecuzione di un'operazione [19]
- Eventi temporali, che si attivano in corrispondenza di un momento specifico o dopo un intervallo di tempo [19]
- Eventi di modifica, che si verificano quando un'espressione booleana passa da false a true. [19]

Questi eventi possono attivare transizioni sia interne che esterne e possono essere parametrizzati, consentendo l'uso di argomenti nelle condizioni di guardia o nei comportamenti associati, aumentando così la flessibilità e la precisione del modello [19].

In sintesi, il diagramma macchina a stati si configura come uno strumento potente e rigoroso per rappresentare il comportamento di strutture che rispondono a stimoli esterni in modo deterministico; la sua capacità di modellare stati, transizioni, condizioni e azioni consente una rappresentazione chiara e verificabile del comportamento interno di un sistema, facilitando la comunicazione tra stakeholder, la progettazione consapevole e, nei contesti più avanzati, anche la generazione automatica del codice [19].

CAPITOLO 5 Strumenti e Ambiente di Sviluppo

Nel seguente lavoro di tesi si è scelto di adottare un approccio basato sulla Model-Based Systems Engineering (MBSE), con l'obiettivo di progettare un sistema complesso attraverso l'integrazione e co-simulazione di modelli realizzati con strumenti diversi, ovvero Cameo Systems Modeler e Simulink, la cui integrazione costituisce un elemento centrale del flusso di lavoro implementato.

Cameo Systems Modeler, sviluppato da Dassault Systèmes, è una piattaforma avanzata per la modellazione di sistemi secondo lo standard SysML (Systems Modeling Language). Questo strumento consente di rappresentare in modo strutturato e tracciabile i diversi aspetti di un sistema, tra cui requisiti, architettura, comportamenti e vincoli. La sua architettura modulare e l'ampio supporto agli standard MBSE lo rendono particolarmente adatto per ambienti ingegneristici in cui è richiesta coerenza tra le diverse fasi del ciclo di vita del sistema. [20]

Parallelamente, Simulink, sviluppato da MathWorks, rappresenta uno standard industriale per la modellazione e la simulazione di sistemi dinamici, in particolare nei domini dell'automazione, dell'elettronica e del controllo. La sua interfaccia grafica consente di costruire modelli numerici dettagliati e di eseguire simulazioni temporali per analizzare il comportamento del sistema sotto diverse condizioni operative. [21]

L'integrazione tra Cameo e Simulink, resa possibile tramite specifici connettori e meccanismi di co-simulazione, permette di colmare il divario tra la modellazione concettuale (SysML) e la simulazione numerica (Simulink). In questo contesto, i modelli SysML fungono da riferimento architetturale e funzionale, mentre Simulink consente di validare quantitativamente i comportamenti dinamici del sistema. Questo approccio integrato consente una maggiore coerenza tra progettazione e verifica, riducendo il rischio di incongruenze e migliorando la tracciabilità dei requisiti fino alla simulazione. [22]

Nel corso della tesi, verranno quindi sviluppati modelli SysML in Cameo Systems Modeler che tratteranno l'analisi funzione e che, in seguito saranno successivamente collegati a modelli Simulink per l'analisi dinamica. Tale metodologia consente di sfruttare i punti di forza di entrambi gli strumenti, promuovendo una visione sistemica e multidisciplinare del progetto.

5.1 Cameo System Modeler

Nel contesto della progettazione di sistemi complessi, l'approccio Model-Based Systems Engineering (MBSE) si è affermato come una metodologia fondamentale per garantire coerenza, tracciabilità e qualità lungo tutto il ciclo di vita del sistema. In questo lavoro di tesi, uno degli strumenti principali adottati è Cameo Systems Modeler, una piattaforma avanzata e ampiamente riconosciuta nel settore per la modellazione sistemica secondo lo standard SysML. [20]

Cameo si distingue per la sua capacità di supportare in modo integrato tutte le fasi della progettazione, dalla definizione dei requisiti fino alla verifica e validazione del sistema. L'ambiente offre una gamma di strumenti intelligenti e intuitivi che permettono di costruire modelli formali e visuali, capaci di rappresentare in modo strutturato gli aspetti architetturali, comportamentali e parametrici del sistema. [20]

Uno degli elementi centrali di Cameo è la gestione dei requisiti, che avviene attraverso diagrammi SysML dedicati e una tracciabilità completa tra requisiti, componenti, test case e livelli di astrazione. Questo consente di mantenere una visione chiara e coerente del sistema, facilitando l'analisi delle lacune progettuali e la verifica della copertura dei requisiti.[20]

Il software include inoltre funzionalità per il controllo automatico della consistenza del modello, che aiutano a individuare errori logici o incoerenze strutturali durante lo sviluppo; a ciò si aggiungono strumenti per l'analisi ingegneristica, che permettono di valutare le decisioni progettuali e verificare i requisiti attraverso modelli parametrici e misure di efficacia del sistema (MoEs). [20]

Dal punto di vista collaborativo, Cameo si integra con MagicDraw Teamwork Cloud, una piattaforma che consente lo sviluppo distribuito e simultaneo del modello da parte di più utenti; questo sistema di archiviazione centralizzato garantisce che tutte le modifiche siano tracciate e gestite in modo ordinato, evitando conflitti e facilitando il lavoro in team. [20]

Un altro aspetto rilevante è la capacità di pubblicazione diretta dei modelli in vari formati, come documenti, immagini e visualizzazioni web, rendendo più semplice la comunicazione con stakeholders di diversa natura, sia tecnici che non tecnici. Inoltre, la interoperabilità del software consente l'esportazione dei modelli in formato standard XMI e l'integrazione con altri ambienti di simulazione e analisi, rendendo Cameo altamente adattabile a diversi contesti industriali. [20]

Grazie a queste caratteristiche, Cameo Systems Modeler è oggi utilizzato in settori ad alta complessità come l'aerospazio, la difesa, l'automotive e l'industria manifatturiera avanzata. La sua adozione in questo lavoro di tesi permette di strutturare un processo di modellazione rigoroso e scalabile, che sarà ulteriormente potenziato attraverso l'integrazione con Simulink, ambiente dedicato alla simulazione dinamica. Questa integrazione consentirà di validare i modelli SysML attraverso simulazioni numeriche, offrendo una visione più completa e realistica del comportamento del sistema progettato.

5.2 Matlab Simulink

Nel percorso di progettazione di un sistema complesso, la possibilità di esplorare, testare e validare le soluzioni prima ancora di passare all'implementazione fisica rappresenta un vantaggio strategico fondamentale. In questo contesto, Simulink si rivela uno strumento estremamente potente e versatile. Sviluppato da MathWorks, Simulink è un ambiente di modellazione grafica basato su diagrammi a blocchi, pensato per rappresentare e simulare sistemi dinamici multidominio in modo intuitivo e modulare. [21]

Uno degli aspetti più apprezzati di Simulink è la sua capacità di anticipare la fase di test, permettendo di simulare il comportamento del sistema in condizioni realistiche già nelle prime fasi della progettazione. Questo approccio consente di individuare rapidamente eventuali criticità, esplorare alternative progettuali e ottimizzare le prestazioni, riducendo tempi e costi legati alla prototipazione fisica. [21]

Simulink si inserisce perfettamente nella filosofia della Model-Based Systems Engineering (MBSE), offrendo un ambiente integrato in cui è possibile passare in modo fluido dalla definizione dei requisiti alla progettazione architetturale, fino alla generazione automatica del codice e alla validazione. [21] I modelli creati in Simulink non sono semplici rappresentazioni grafiche, ma veri e propri artefatti eseguibili, che possono essere utilizzati per:

- Simulare e testare virtualmente il sistema, anche in configurazioni avanzate come la prototipazione rapida o l'Hardware-In-the-Loop (HIL).
- Generare codice di produzione in diversi linguaggi (C, C++, CUDA, PLC, Verilog, VHDL), pronto per essere distribuito su sistemi embedded.
- Mantenere la tracciabilità tra requisiti, architettura, componenti, codice e test, creando un collegamento continuo tra le diverse fasi dello sviluppo.
- Eseguire simulazioni su larga scala, sfruttando risorse locali o distribuite, come cluster o ambienti cloud.

Un altro punto di forza di Simulink è la sua capacità di adattarsi a metodologie di sviluppo moderne, come l'approccio Agile. Grazie alla possibilità di eseguire test automatizzati e simulazioni continue, i team di sviluppo possono iterare rapidamente sulle soluzioni, rispondere in modo flessibile ai cambiamenti nei requisiti e fornire risultati valutabili in tempi brevi. [21]

Nel contesto di questa tesi, Simulink verrà utilizzato in integrazione con Cameo Systems Modeler, per realizzare una co-simulazione che consenta di validare dinamicamente i modelli SysML sviluppati. Questa integrazione rappresenta un ponte tra la modellazione concettuale e la simulazione numerica, offrendo una visione completa e coerente del sistema, dalla sua architettura logica fino al comportamento dinamico. [21]

5.3 Integrazione

Come anticipato nelle sezioni precedenti, questo elaborato ha come obiettivo principale l'integrazione funzionale tra due ambienti di sviluppo distinti ma complementari: Cameo Systems Modeler e Simulink. L'idea alla base di questa integrazione è quella di sfruttare le potenzialità di ciascuno strumento per ottenere una rappresentazione completa e dinamica del sistema oggetto di studio.

Da un lato, Cameo Systems Modeler consente di costruire un modello concettuale e architetturale del sistema nella sua totalità. Attraverso l'utilizzo dello standard SysML, è possibile suddividere il sistema in sottosistemi e componenti, descrivere le interconnessioni tra di essi, rappresentare i flussi

di informazioni, gli stati operativi e le attività svolte. Questo tipo di modellazione permette di avere una visione strutturata e tracciabile del sistema, utile sia in fase di progettazione che di analisi.

Dall'altro lato, Simulink viene impiegato come motore di calcolo per la simulazione dei comportamenti dinamici del sistema. In pratica, Simulink si occupa di eseguire le elaborazioni numeriche e le simulazioni associate ai modelli definiti in Cameo. Questa suddivisione dei ruoli consente di mantenere la modellazione concettuale e architetturale all'interno di Cameo, mentre la parte computazionale e simulativa viene delegata a Simulink, che offre strumenti avanzati per la simulazione multidominio e la generazione automatica del codice.

L'integrazione tra i due ambienti permette quindi di realizzare una co-simulazione, in cui i modelli funzionali Cameo (in linguaggio SysML) vengono arricchiti da modelli dinamici simulati in Simulink. Questo approccio consente di validare il comportamento del sistema in modo realistico, verificando che le specifiche architetturali siano coerenti con le prestazioni attese e facilitando l'analisi di scenari complessi.

Per abilitare questa integrazione, è necessario disporre di una versione a 64 bit di MATLAB (R2016b o successiva), in modo da garantire la compatibilità con l'architettura a 64 bit degli strumenti di modellazione come MagicDraw o Cameo. Una volta configurato correttamente l'ambiente, è possibile utilizzare espressioni scritte in sintassi MATLAB direttamente all'interno del Simulation Toolkit di Cameo (vedremo in seguito come utilizzare le opaque actions), rendendo possibile l'esecuzione di simulazioni parametriche e l'analisi di scenari complessi.[22]

Affinché l'integrazione funzioni correttamente, è importante:

- Impostare la prospettiva corretta all'interno di Cameo (ad esempio "Full Featured", "System Engineer" o "Software Architect"), in modo da abilitare il menu delle integrazioni. [22]
- Eseguire un riavvio del sistema dopo la prima configurazione o in seguito a un cambio di versione di MATLAB, per assicurarsi che l'ambiente riconosca correttamente il collegamento. [22]
- Avviare Cameo con privilegi di amministratore, specialmente su sistemi Windows, per evitare problemi di accesso o di comunicazione tra i due ambienti. [22]

Questa configurazione consente di sfruttare appieno le potenzialità della co-simulazione, in cui i modelli SysML definiti in Cameo possono essere arricchiti da componenti dinamici simulati in Simulink. In questo modo, è possibile validare il comportamento del sistema in modo più realistico, verificando che le specifiche architetturali siano coerenti con le prestazioni attese. [22]

L'integrazione tra modellazione e simulazione non solo migliora la qualità del progetto, ma consente anche una maggiore tracciabilità tra requisiti, architettura e risultati di test, contribuendo a una progettazione più solida, iterativa e orientata alla verifica continua.

Una volta accertatosi di rispettare i requisiti di integrazione, è sufficiente andare alla voce integrations, presente nel menu tools di Cameo System Modeler è selezionare l'integrazione con Matlab. (fig.20)

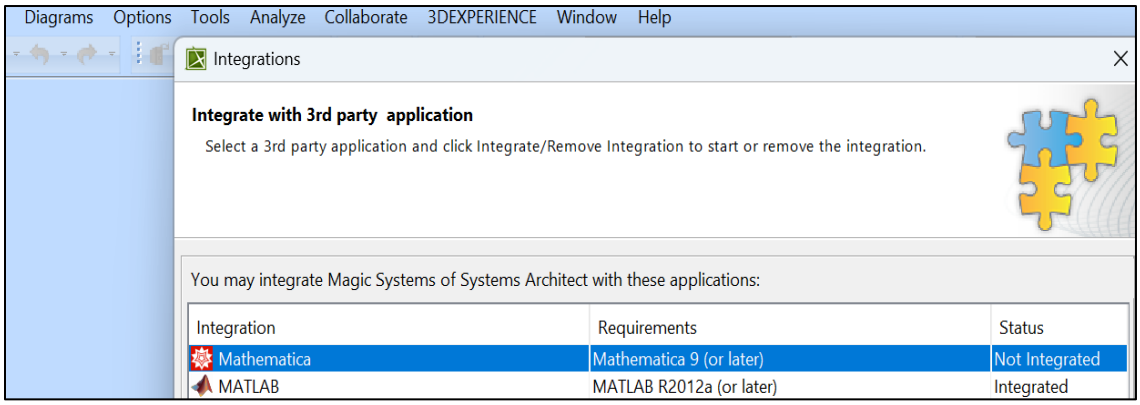

Fig.20, realizzata a fine dimostrativo tramite Cameo

Da questo momento in poi l'integrazione tra i due tool è pienamente funzionante, e non resta che realizzare i modelli Simulink da utilizzare all'interno dell'ambiente Cameo a fine simulativo.

Una volta completata la modellazione del sistema in Simulink, è stato necessario definire un meccanismo efficace per integrare tali modelli all'interno dell'ambiente Cameo Systems Modeler, al fine di sfruttarne le capacità di simulazione e analisi all'interno di un contesto MBSE. A tal proposito, sono state esplorate due principali modalità operative, entrambe valide ma differenti per approccio e livello di astrazione.

La prima modalità prevede l'utilizzo di Opaque Action all'interno di un Activity Diagram per orchestrare l'interazione tra Cameo e Simulink. Questo approccio si basa sulla scrittura di comandi in linguaggio MATLAB, che vengono interpretati ed eseguiti durante la simulazione (fig.22). Il processo si articola in più fasi:

- Richiamo del modello Simulink: si utilizza una Opaque Action in cui si specifica matlab come linguaggio e si richiama il modello semplicemente in forma testuale all'interno dell'azione (fig.21)
- Definizione delle variabili di input: attraverso ulteriori Opaque Action, si assegnano i valori alle variabili di input del modello (ad esempio vx, vy, vz), che devono essere coerentemente dichiarate come Value Property all'interno del blocco (all' interno del body block diagram) proprietario dell'Activity Diagram. (Fig.20)
- Esecuzione della simulazione: una nuova Opaque Action contenente il comando "sim('modello')" avvia la simulazione vera e propria. (fig.21)
- Visualizzazione degli output: i risultati della simulazione possono essere visualizzati nella console o utilizzati all'interno del modello, a condizione che siano anch'essi definiti come Value Property e gestiti tramite Opaque Action.

Questo approccio offre un elevato grado di controllo e flessibilità, ma richiede una buona padronanza della sintassi MATLAB e una gestione esplicita delle variabili e dei flussi di dati.

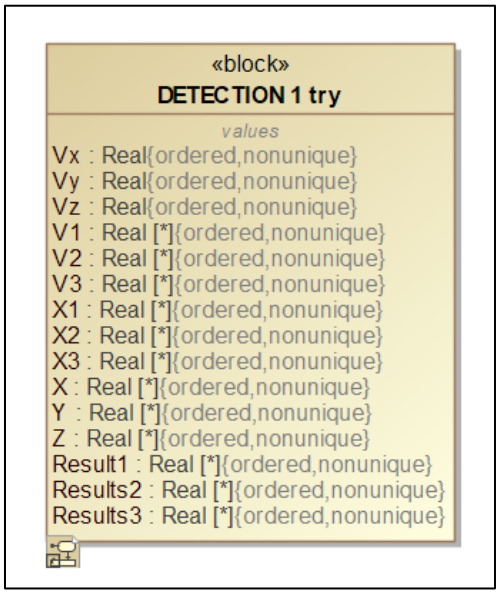

Fig.21 esempio di blocco con value property definite in modo da poter essere impostati come parametri nell'activity diagram

La seconda modalità, più diretta e visuale, consiste nel trascinare direttamente il file ".slx" all'interno di un Activity Diagram in Cameo. In questo caso, il modello Simulink viene rappresentato come un blocco con pin di input e output visibili, che possono essere collegati ad altre azioni o variabili nel diagramma.

Anche in questo scenario, è necessario definire le variabili di input e output come Value Property all'interno del blocco che possiede l'Activity Diagram. Le assegnazioni dei valori in ingresso e la gestione dei risultati in uscita vengono comunque effettuate tramite Opaque Action, ma l'interfaccia grafica semplifica notevolmente il processo di integrazione e rende più immediata la comprensione del flusso di simulazione. (fig.23)

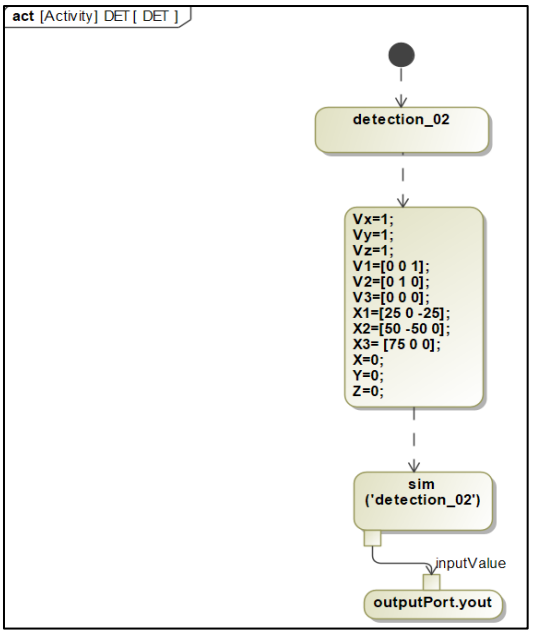

Fig.22 esempio activity diagram sviluppato per attuare la co-simulazione secondo il primo metodo

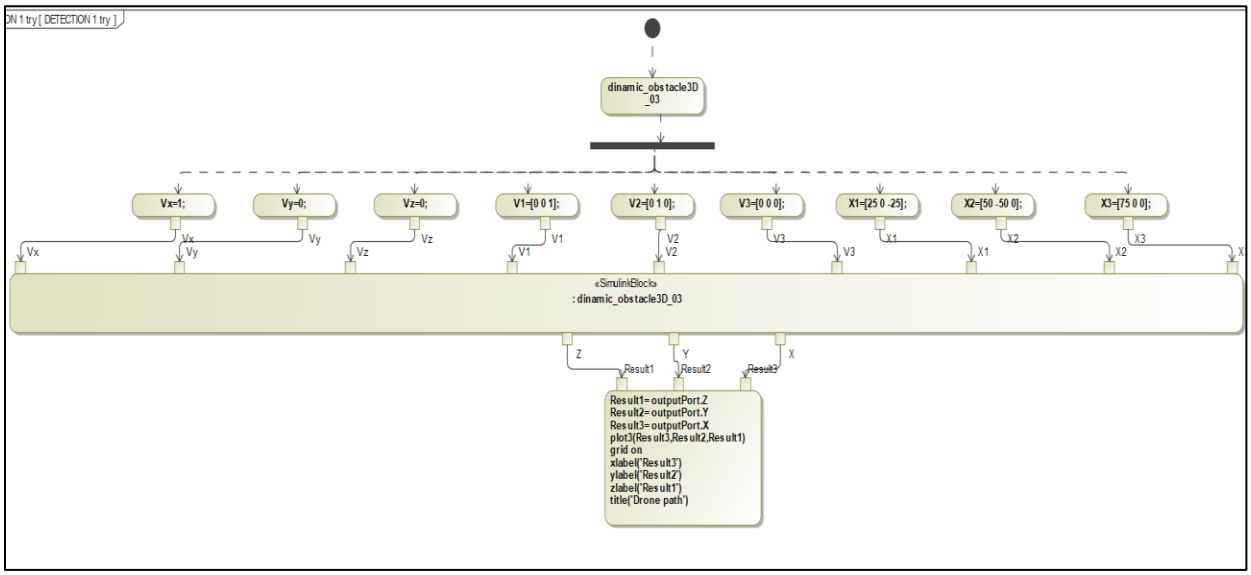

Fig. 23 esempio di activity diagram realizzato trascinando il modello.slx all'interno del diagramma, sono chiaramente distinguibili i pin di input e output

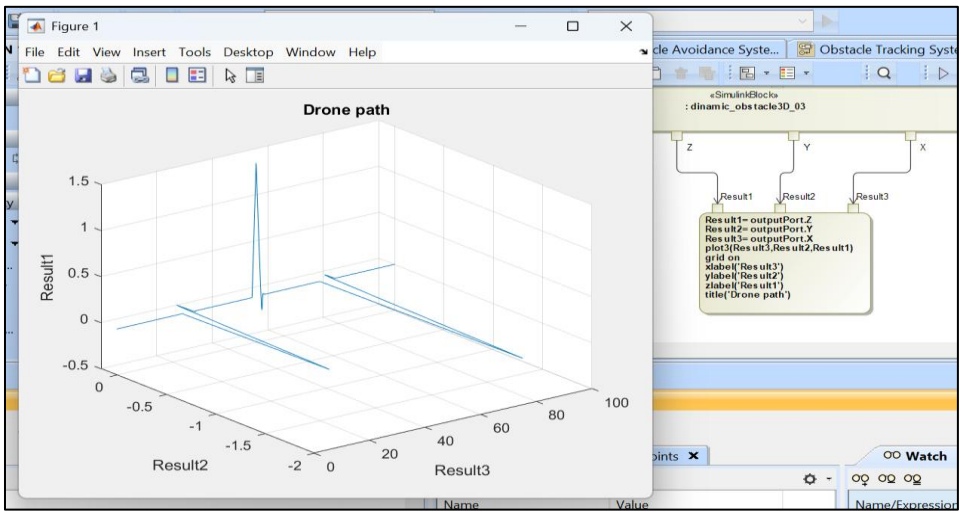

Fig.24, plot matlab richiamato tramite codice contenuto in un opaque action

Entrambi gli approcci permettono di sfruttare Simulink come motore di calcolo all'interno di un flusso MBSE orchestrato da Cameo, consentendo di validare dinamicamente i modelli SysML attraverso simulazioni numeriche. La scelta tra le due modalità dipende dal livello di controllo richiesto, dalla complessità del modello e dalle preferenze metodologiche del progettista. In generale comunque è possibile in entrambi i casi visualizzare gli output in console, e richiamare plot matlab direttamente in cameo tramite righe di codice nelle opaque action (fig.24).

CAPITOLO 6 Drone System

Il presente elaborato si colloca all'interno del progetto DRONE (fig.68), iniziativa interna di Capgemini concepita per dimostrare le capacità avanzate di Model-Based Systems Engineering (MBSE) dei team di Ricerca e Sviluppo. DRONE rappresenta un dimostratore tecnologico multidisciplinare, in grado di integrare concetti chiave come Digital Twin, Digital Continuity, simulazioni fisiche, e tecnologie immersive quali Realtà Virtuale e Aumentata (VR/AR). [23]

Fig.68 progetto DRONE [23]

Il sistema è incarnato in una piattaforma eVTOL (electric Vertical Take-Off and Landing), progettata per affrontare il problema emergente del trasporto autonomo di carichi ad alta priorità in modo efficiente e sostenibile. Il caso d'uso principale riguarda la consegna di forniture mediche e organi in contesti di emergenza o tra istituzioni sanitarie. DRONE ha debuttato in occasione di eventi dimostrativi rivolti a un pubblico eterogeneo, che ha incluso studenti, esperti del settore e stakeholder industriali. [23]

Alla base del progetto vi è una visione sistemica e collaborativa, in cui l'MBSE funge da fonte unica di verità per la progettazione e lo sviluppo. Il modello è stato costruito secondo la metodologia MagicGrid, garantendo tracciabilità, verificabilità e co-simulazione tra i diversi domini ingegneristici. Le attività di verifica e validazione precoce hanno incluso [23]:

- Modellazione CAD e assemblaggio virtuale dei componenti prima della stampa 3D;
- Sviluppo e test dei protocolli di comunicazione e del software;
- Test fisici dei componenti elettronici prima dell'assemblaggio finale.

Il progetto ha seguito una storyline strutturata, che ha previsto l'identificazione di problemi industriali complessi ("wicked problems"), la definizione dell'architettura e delle specifiche per la fornitura di assistenza medica e interventi di emergenza, e la transizione del concetto dalla fase di design a quella di implementazione. Tale transizione ha coinvolto attività di progettazione CAD,

stampa 3D, assemblaggio del drone, sviluppo di script di comunicazione, progettazione elettronica e test, fino alla configurazione dei protocolli tra sistemi interdipendenti. [23]

La prima generazione di DRONE ha dimostrato con successo le seguenti capacità:

- Modellazione MBSE completa e co-simulativa;
- Progettazione CAD, stampa 3D e assemblaggio del drone;
- Selezione, configurazione e test dei componenti elettronici;
- Sviluppo di API e connettività;
- Integrazione di un sistema VR per l'interazione con il Digital Twin;
- Modellazione MBSE dei processi industriali legati alla produzione e assemblaggio.

Il progetto ha richiesto competenze trasversali in ambiti quali CAD, stampa 3D, elettronica, scripting Python, motori fisici (CoppeliaSim), e VR. Tra i software utilizzati figurano Cameo Systems Modeler per l'MBSE, CATIA/3DX per il CAD, Arduino per l'elettronica, e l'app VR "Drone" per la visualizzazione immersiva. [23]

È importante quindi sottolineare che DRONE non è stato progettato per il volo reale, ma come piattaforma dimostrativa. Alcuni vincoli e rischi tecnici, come la gestione termica delle batterie e dei motori, la sicurezza dei servomeccanismi e la stabilità dell'app VR, sono stati affrontati con misure specifiche. Inoltre, la complessità e le dimensioni del sistema impongono limitazioni logistiche in fase di trasporto e allestimento.[23]

In linea con questa visione sistemica e multi-dominio, il presente lavoro ha contribuito allo sviluppo e alla modellazione di una delle funzionalità fondamentali per l'autonomia e la sicurezza del drone: la capacità di rilevare ed evitare ostacoli. Tale funzionalità, che sarà approfondita successivamente, rappresenta un elemento abilitante per l'operatività del sistema in scenari complessi e dinamici, e costituisce un caso d'uso rappresentativo dell'integrazione tra modellazione architetturale e simulazione comportamentale.

6.1 Detect and Avoid Obstacles

Nel contesto delle missioni autonome ad alta complessità, la capacità di un sistema aereo di rilevare, classificare e reagire in tempo reale alla presenza di ostacoli rappresenta una delle funzionalità più critiche per garantire la sicurezza operativa e la continuità della missione. Il caso d'uso "Detect & Avoid Obstacles" si inserisce all'interno di questa esigenza, proponendo un comportamento intelligente e adattivo che consente al drone di operare in ambienti dinamici, non strutturati e potenzialmente ostili, sia in scenari di volo libero che in contesti tattici come le operazioni di convoglio. [24] La funzionalità si basa su una catena di elaborazione articolata (fig.25), che inizia con la rilevazione dell'ostacolo attraverso un insieme eterogeneo di sensori, come LIDAR, camere RGB e sensori infrarossi (necessari per garantire operatività in condizioni di scarsa visibilità o banalmente di notte). Una volta acquisiti i dati ambientali, inoltre, il sistema deve essere in grado, in base alla situazione e al tipo di ostacolo, di elaborare manovre evasive o di evitamento a seconda dei casi (sarà quindi necessario stimare diversi parametri dell'ostacolo al fine di evitarli). L'intero

processo è gestito in tempo reale da un algoritmo software dedicato, che tiene conto delle condizioni ambientali, dei vincoli di volo e degli obiettivi di missione. [24]

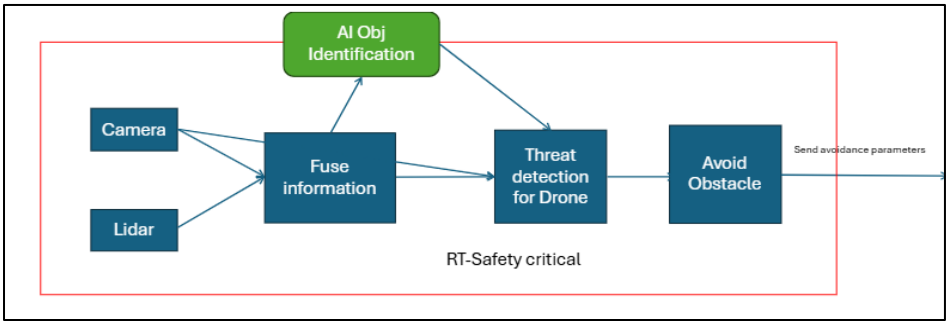

Fig.25, Logica dietro il caso d'uso "Detect and Avoid obstacle" [24]

Questa capacità è stata pensata considerando che il drone viene inviato da un punto A a un punto B, operando in autonomia sia in condizioni diurne che notturne, infatti, durante il volo, il sistema deve rilevare un ostacolo lungo la traiettoria, identificarne la tipologia e predirne il comportamento, adottando una manovra di evitamento che consente di proseguire la missione in sicurezza. In caso di situazioni critiche o impreviste, il sistema è inoltre in grado di attivare procedure di emergenza, come l'atterraggio controllato o la comunicazione prioritaria con la stazione di terra. [24]

Il sistema deve essere in grado di operare a diverse altitudini, in condizioni ambientali variabili, e per elaborare autonomamente il percorso sulla base della destinazione impostata dall'operatore; inoltre, dovrebbe acquisire immagini e video dell'ambiente circostante, archiviare le informazioni rilevanti in un database interno e mantenere una comunicazione continua con il centro di comando.[24]

La capacità di rilevare, identificare e reagire agli ostacoli non si limita alla semplice evitamento, ma si configura come un comportamento cognitivo distribuito, che contribuisce alla costruzione di una conoscenza situazionale condivisa e all'adattamento dinamico della missione. Tale funzionalità rappresenta un elemento abilitante per l'impiego del sistema Drone in contesti operativi ad alta variabilità, dove l'autonomia decisionale e la resilienza del sistema costituiscono fattori determinanti per il successo della missione.[24]

6.1.1 Detection Subsystem

Per garantire la sicurezza operativa del drone durante missioni autonome in ambienti complessi, è stato pensato di sviluppare un sistema avanzato di rilevamento e gestione degli ostacoli denominato AI-SODIA (il nome deriva dal fatto che il sistema deve essere in grado di notare, "identificare" tramite AI ed evitare ostacoli). Questo sistema deve operare in tempo reale, integrando dati provenienti da più sensori e applicando logiche decisionali intelligenti per identificare, classificare e reagire a potenziali minacce lungo la traiettoria di volo. (fig.26, sarà possibile vedere i componenti citati in questo sotto capitolo e le loro interazioni) [24]

Il processo inizia con la raccolta di dati da tre fonti sensoriali principali: una camera RGB, che fornisce immagini visive ad alta risoluzione; un sensore a infrarossi, utile per la rilevazione termica in condizioni di scarsa visibilità o durante operazioni notturne; e un sensore LIDAR, che genera una nuvola di punti tridimensionale dell'ambiente circostante. Ogni dato acquisito è associato a un timestamp, permettendo una sincronizzazione temporale accurata tra le diverse fonti. Successivamente, la nuvola di punti generata dal LIDAR viene sottoposta a una serie di elaborazioni: viene prima filtrata per rimuovere il rumore, poi segmentata per isolare le superfici rilevanti, e infine suddivisa in cluster che rappresentano potenziali ostacoli. [24]

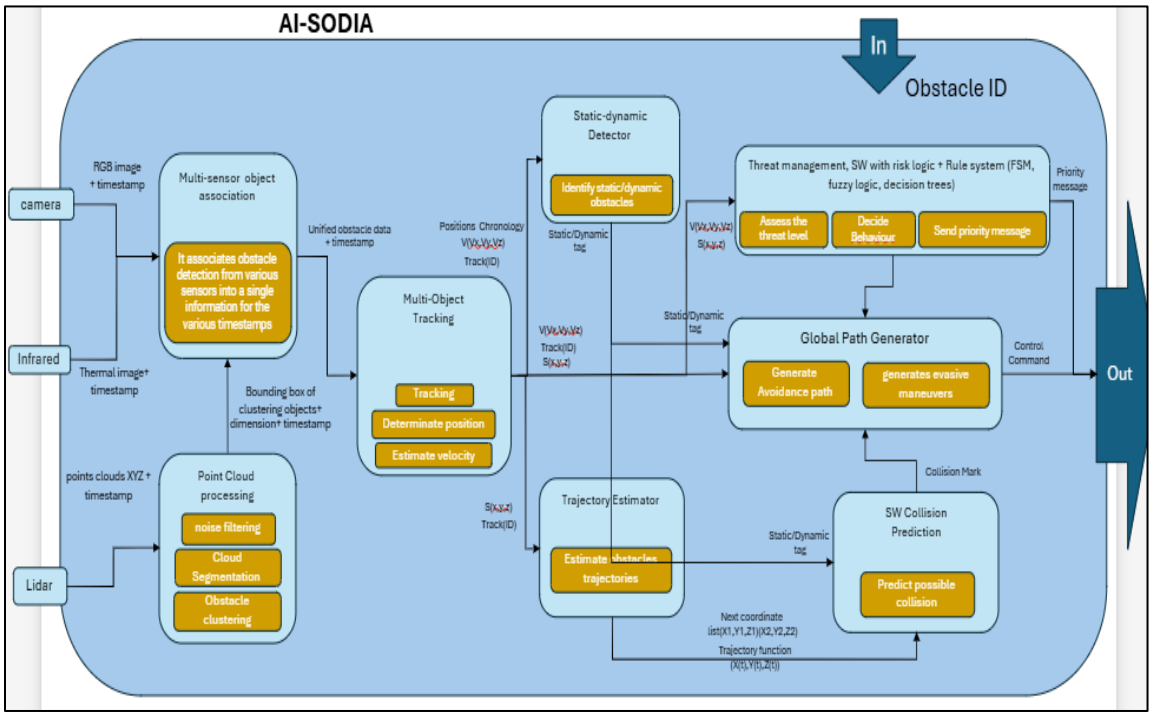

Fig.26, AI-SODIA, Detection Subsystem [24]

Una volta acquisiti, i dati vengono elaborati in un modulo di associazione multi-sensoriale, che ha il compito di integrare le informazioni provenienti dai diversi sensori in una rappresentazione unificata e coerente dell'ambiente; questo passaggio è fondamentale per ridurre l'ambiguità e migliorare l'affidabilità della percezione. A questo punto, il sistema attiva un modulo di tracciamento multi-oggetto, che consente di monitorare nel tempo la posizione e la velocità relativa di ciascun ostacolo rilevato. [24]

Un elemento chiave dell'architettura è la capacità di distinguere tra ostacoli statici e dinamici; attraverso l'analisi delle variazioni spaziali e temporali, il sistema deve essere in grado di classificare ogni oggetto in base al suo comportamento, stimando anche la traiettoria futura degli ostacoli in movimento. Questa informazione è essenziale per anticipare situazioni di rischio e pianificare manovre evasive.

È previsto inoltre un componente del sistema focalizzato alla gestione dei tipi di ostacolo, rappresentato dal modulo di identificazione e gestione della minaccia, che utilizza logiche basate su macchine a stati finiti, fuzzy logic e alberi decisionali per valutare il livello di rischio associato a ciascun ostacolo. In base alla valutazione effettuata, il sistema decide il comportamento più appropriato da adottare e, se necessario, invia messaggi prioritari al sistema di controllo (questo componente dovrebbe ricevere informazioni riguardo all'ID dell'ostacolo da parte di un ipotetico identification subsystem). [24]

Infine, un modulo dedicato alla predizione delle collisioni valuta la probabilità di impatto e invia segnali al generatore di percorsi, che elabora un possibile percorso evasivo da inviare ed elaborare nell'avoidance subsystem. [24]

L'intero processo è concepito per essere altamente modulare, scalabile e adattabile a diversi scenari operativi, garantendo al drone la capacità di navigare in modo sicuro e autonomo anche in ambienti non strutturati e ad alta variabilità.

6.1.2 Avoidance Subsystem

Per garantire la sicurezza operativa del drone durante missioni autonome in ambienti complessi e dinamici, è stato pensato un sottosistema di *Avoidance* in grado di gestire in modo efficace la presenza di ostacoli lungo la traiettoria di volo. Questo sistema dovrebbe funzionare in due modalità distinte: una modalità manuale, che consente l'intervento diretto dell'operatore, e una modalità autonoma, che permette al drone di reagire in tempo reale in completa autonomia. [24]

Nella modalità manuale, il drone mantiene un collegamento continuo con la stazione di terra, dalla quale l'operatore può monitorare l'ambiente circostante attraverso i dati visivi e termici acquisiti dai sensori di bordo. Qualora il sistema autonomo non sia in grado di gestire una situazione complessa o ambigua, l'operatore può intervenire direttamente, impartendo comandi di evitamento tramite l'interfaccia di controllo. Questa modalità rappresenta un'importante misura di sicurezza, soprattutto in scenari ad alta incertezza o in presenza di ostacoli non classificabili automaticamente.[24]

La modalità autonoma (fig.27), invece, si basa su un'architettura distribuita e reattiva, articolata in più moduli funzionali che collaborano per garantire una navigazione sicura e adattiva. Il processo inizia con la generazione di una mappa dinamica locale (Dynamic Costmap), costruita in tempo reale a partire dai dati della nuvola di punti acquisita dai sensori. Ogni cella della mappa rappresenta una porzione dello spazio e viene associata a un valore di "costo" che riflette la difficoltà o il rischio di attraversarla: le aree libere hanno un costo basso, mentre le zone occupate da ostacoli o considerate pericolose presentano un costo elevato. Per aumentare la sicurezza, le celle adiacenti agli ostacoli vengono "gonfiate", creando un margine di sicurezza attorno agli oggetti rilevati. [24]

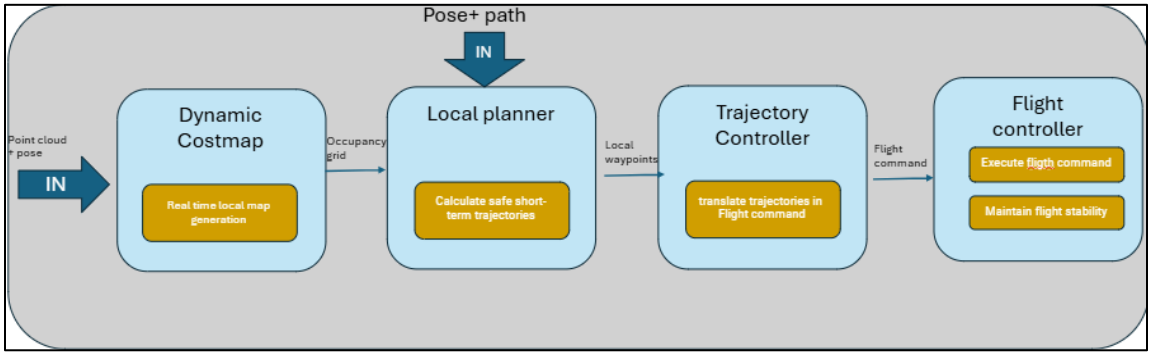

Fig.27 Autonomous Avoidance Subsystem [24]

Sulla base di questa rappresentazione, il pianificatore locale (Local Planner) genera una serie di traiettorie candidate a breve termine, che tengono conto della posizione attuale del drone, del percorso globale previsto e delle informazioni contenute nella costmap. Le traiettorie vengono simulate considerando le dinamiche del drone, come velocità, accelerazione e capacità di virata, e successivamente valutate secondo un criterio di costo complessivo. Tale costo tiene conto di diversi fattori, tra cui la distanza dagli ostacoli, la fluidità del movimento, il consumo energetico e il rispetto dei vincoli dinamici. La traiettoria con il costo minimo, che soddisfa tutti i vincoli, viene selezionata come ottimale. [24]

Una volta scelta la traiettoria, il controllore di traiettoria (Trajectory Controller) la traduce in comandi di volo, calcolando la differenza tra lo stato attuale del drone e quello desiderato; viene quindi applicata una legge di controllo (ad esempio un regolatore PID) per minimizzare l'errore e garantire una manovra fluida e precisa, mentre i comandi generati vengono infine inviati al Flight Controller, che li esegue mantenendo la stabilità del volo.

L'intero processo è gestito in ciclo chiuso, con aggiornamenti continui basati sui dati sensoriali in ingresso, permettendo al drone di adattarsi dinamicamente all'ambiente circostante e di reagire prontamente a ostacoli imprevisti. Questa architettura consente al sistema di operare in modo robusto anche in ambienti non strutturati, garantendo un elevato livello di autonomia e sicurezza.

6.1.3 Sviluppi futuri, Identification Subsystem e SLAM Subsystem

In una fase successiva dell'evoluzione del sistema, è previsto lo sviluppo di un sottosistema dedicato all'identificazione degli ostacoli, basato su tecniche di Intelligenza Artificiale, che sarà in grado di riconoscere e classificare oggetti rilevati nell'ambiente circostante. Questo modulo sfrutterà l'input proveniente da sensori visivi, come camere RGB e LiDAR, per rilevare la presenza di ostacoli e sarà integrato con un sistema SLAM (Simultaneous Localization and Mapping), capace di generare e aggiornare in tempo reale una mappa tridimensionale dell'ambiente operativo, fornendo all'utente una rappresentazione dinamica e dettagliata dello scenario. L'integrazione di questi componenti permetterà di migliorare significativamente le capacità di percezione, navigazione autonoma e consapevolezza ambientale del sistema, rendendolo più robusto e affidabile anche in contesti complessi e non strutturati.

Tuttavia, nel presente lavoro di tesi verrà trattato esclusivamente il caso d'uso "Detect and Avoid Obstacles", che rappresenta il nucleo funzionale attualmente implementato. Per completezza metodologica e coerenza con l'architettura generale del sistema, nei capitoli successivi e nell'analisi funzionale verranno comunque considerati anche lo SLAM Subsystem e l'Identification Subsystem, pur non essendo oggetto di sviluppo diretto in questo elaborato.

Per semplicità espositiva e coerenza terminologica, da questo punto in avanti il sistema relativo al caso d'uso "*Detect & Avoid Obstacles*" verrà indicato semplicemente con il nome "SODIA", includendo implicitamente le componenti attuali, mentre il sistema completo mantiene in nome AI-SODIA (comprendente sia i sottosistemi trattati che quelli che verranno sviluppati in futuro).

CAPITOLO 7 Dal drone al Modello 7

7.1 Definizione di Architettura Logica e Funzionale in Cameo System Modeler

Nel contesto dell'Ingegneria dei Sistemi Basata su Modelli (MBSE), la comprensione e la rappresentazione coerente del comportamento e della struttura di un sistema complesso sono fondamentali per garantire una progettazione efficace e tracciabile. Il framework MagicGrid, sviluppato da CATIA® No Magic, offre un approccio metodologico rigoroso che consente di modellare un sistema attraverso una progressiva decomposizione funzionale e strutturale, mantenendo sempre l'allineamento tra ciò che il sistema deve fare e come è organizzato per farlo. [17]

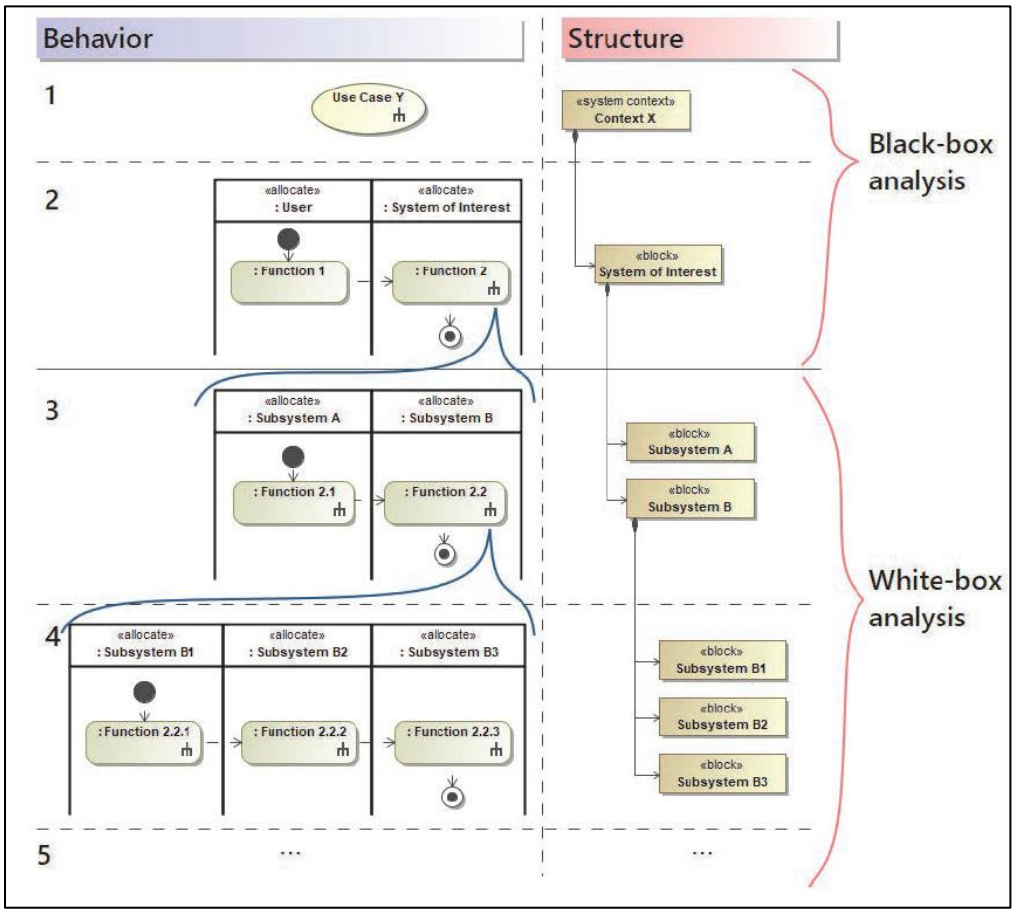

Fig.71 Il comportamento e la struttura decomposti su diversi livelli [17]

Come illustrato nella Figura 71 , il processo di decomposizione si articola su più livelli, distinguendo tra analisi black-box e white-box. Nei livelli superiori, l'attenzione è rivolta alle interazioni tra il sistema e l'ambiente esterno: si parte da un caso d'uso generale ("Use Case Y") associato all'utente, e si definisce il contesto del sistema ("Context X"). Successivamente, si introducono le prime funzioni ("Function 1" e "Function 2"), allocate rispettivamente all'utente e al sistema di interesse, e si rappresenta il sistema come un blocco logico. [17]

Scendendo nei livelli inferiori, l'analisi si approfondisce: il sistema viene suddiviso in sottosistemi ("Subsystem A" e "Subsystem B"), ciascuno con funzioni specifiche, e ulteriormente scomposto in componenti più granulari ("Subsystem B1", "B2", "B3"), con funzioni dettagliate come "Function 2.2.1", "2.2.2", ecc. La struttura segue questa stessa logica, con eblocks che rappresentano le entità fisiche o logiche corrispondenti. [17]

Le frecce di allocazione presenti nel diagramma evidenziano il legame diretto tra comportamento e struttura, garantendo che ogni funzione sia associata a un elemento architetturale preciso. Questo principio di allocazione funzionale è alla base dell'approccio MBSE, poiché consente di mantenere coerenza, tracciabilità e validazione lungo tutto il ciclo di vita del sistema.[17]

Quindi, dopo aver definito la struttura che AI-SODIA dovrà assumere per supportare il caso d'uso "Detect & Avoid Obstacles", è stata modellata l'architettura logica e funzionale del sistema all'interno di Cameo Systems Modeler, in linea con quanto concordato negli incontri con i vari team e illustrato nel precedentemente capitolo. Come mostrato nella figura 28, è stato realizzato il Block Definition Diagram del sistema, rappresentante l'architettura logica suddivisa nei quattro sottosistemi principali:

- Obstacle Avoidance System
- Identify Object System (citato per completezza)
- SLAM System (citato per completezza)
- Detection System

Ognuno di questi sottosistemi è composto da vari componenti funzionali, il cui scopo è stato ampiamente spiegato nel capitolo dedicato (CAPITOLO 6). Si nota che l'Obstacle Avoidance System interagisce direttamente con il Flight Controller per eseguire le manovre di evasione, pertanto vi è associazione tra SODIA (sistema in studio) e il Flight Controller, uno dei sistemi del SOI ovvero il drone.

Dopo aver realizzato il Block Definition Diagram (figura 28), con l'obiettivo di rappresentare l'architettura logica del sistema e la sua decomposizione in sottosistemi principali e componenti funzionali (in particolare evidenziando la struttura interna del *Detection System* e dell'*Obstacle Avoidance System*, come già illustrato nelle figure 26 e 27) si è proseguito con l'analisi funzionale del sistema AI-SODIA, attraverso la modellazione di un Activity Diagram.

In figura 29 è possibile osservare le principali funzioni considerate per il sistema, organizzate secondo una suddivisione in swimlanes che rappresentano le diverse parti coinvolte nel processo operativo. Le swimlanes includono i tre sensori principali, i sottosistemi di *Detection* e *Avoidance*, e due moduli futuri previsti per l'evoluzione del sistema: lo *SLAM System* e l'*Identify Obstacle System*. Le funzioni assegnate a ciascun elemento risultano sufficientemente esplicative: i sensori si occupano dell'acquisizione dei dati ambientali, associati a un riferimento temporale, mentre i quattro sottosistemi principali svolgono le rispettive funzioni operative, ovvero rilevare, identificare, evitare e mappare l'ambiente.

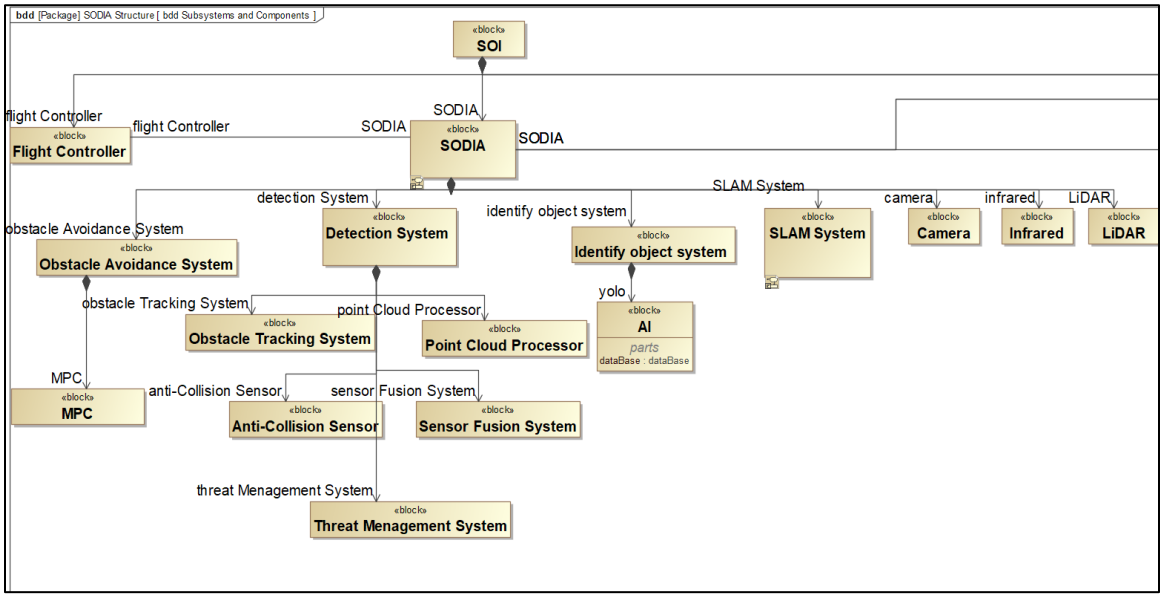

Fig.28 architettura logica di SODIA

Questa rappresentazione funzionale consente di visualizzare chiaramente il flusso delle informazioni e delle azioni all'interno del sistema, evidenziando sia le interazioni tra i componenti attualmente implementati, sia quelle previste per i moduli futuri, che pur non essendo sviluppati in questo lavoro di tesi, vengono comunque considerati nell'analisi per garantire coerenza con l'architettura complessiva.

Analogamente a quanto avviene nella decomposizione strutturale del sistema AI-SODIA, rappresentata nel Block Definition Diagram (figura 28), si è proceduto con una decomposizione funzionale coerente, focalizzandosi sulle funzioni principali relative al caso d'uso "*Detect & Avoid Obstacles*". In particolare, sono state considerate le funzioni *Detect Obstacles* e *Avoid Obstacles*, per le quali sono stati realizzati due ulteriori Activity Diagrams con l'obiettivo di approfondire l'analisi funzionale e descrivere nel dettaglio il comportamento interno dei sottosistemi corrispondenti. Questa scelta è giustificata dal fatto che, anche a livello strutturale, i sottosistemi di Detection e Avoidance sono stati già scomposti nei rispettivi componenti funzionali.

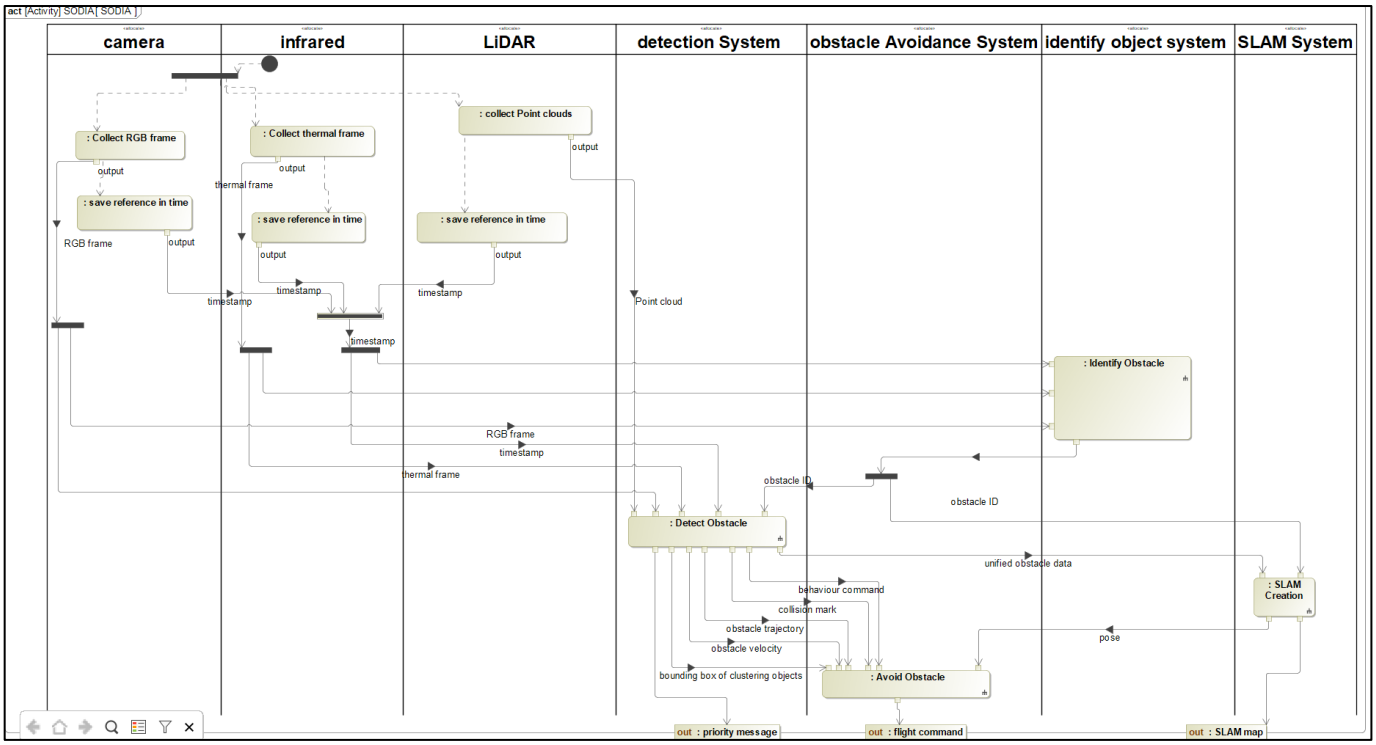

Fig.29 Functional Analysis SODIA

In figura 30 è riportata la decomposizione funzionale della funzione Detect Obstacles (functional analysis a livello componente del sottostistema Detection System), dove sono evidenziate le attività svolte dai diversi componenti che costituiscono il Detection System, come il Point Cloud Processor, il Sensor Fusion System, l'Anti-Collision Sensor, l'Obstacle Tracking System e il Threat Management System, già introdotti nella figura 28. Osservando il diagramma, si nota come le funzioni assegnate a ciascun componente siano coerenti con quanto descritto nel sottocapitolo 6.1.1, in cui è stato illustrato il comportamento del sottosistema durante l'esecuzione del suo compito. Il diagramma evidenzia inoltre il flusso delle informazioni tra i vari elementi, permettendo di comprendere in modo chiaro e sistematico la sequenza operativa e le interazioni funzionali che caratterizzano il processo di rilevamento degli ostacoli.

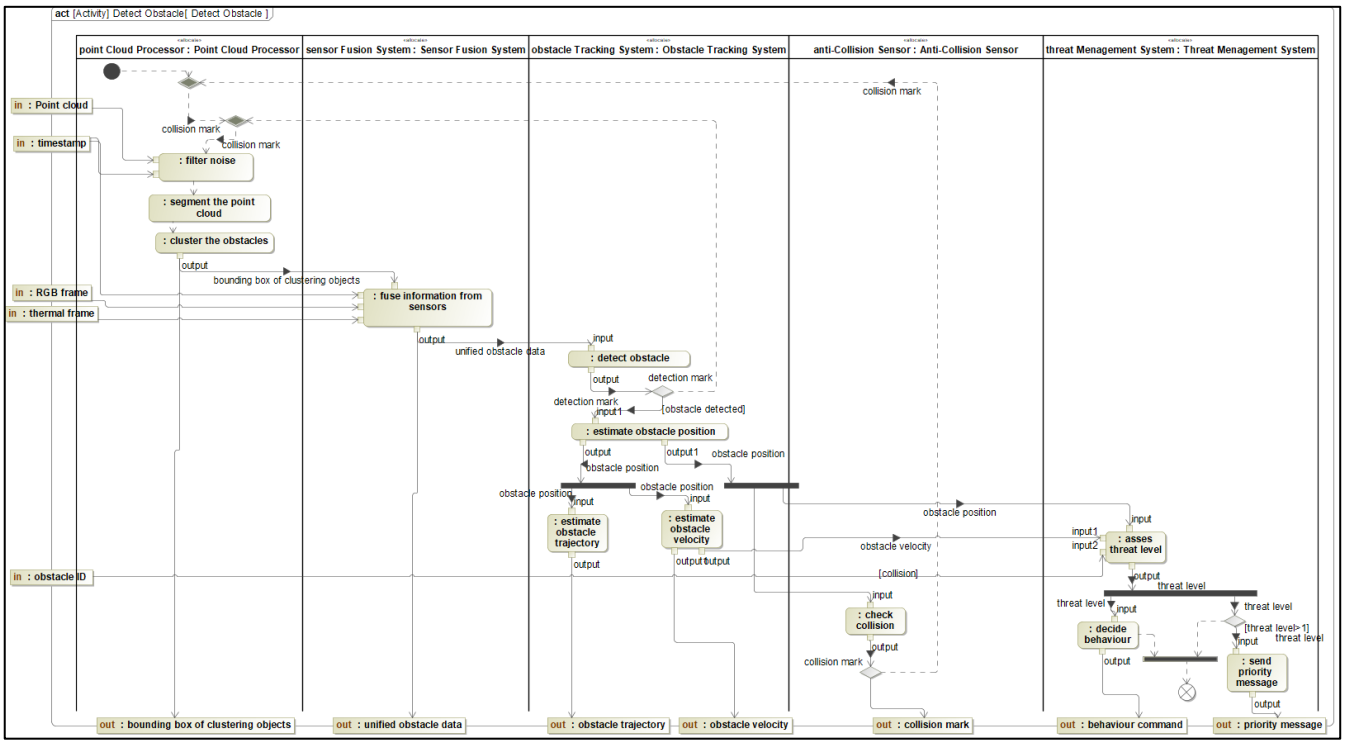

Fig.30 Decomposizione funzionale funzione *Detect Obstacles*

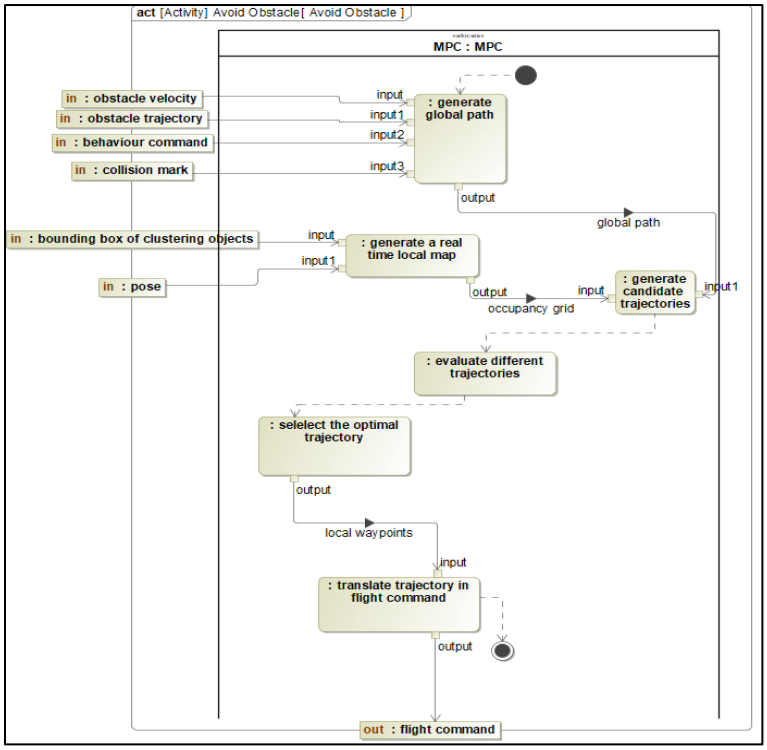

Fig.31 Decomposizione Funzionale Funzione *Avoid Obstacles*

Un discorso analogo si puo fare per figura 31, dove si osserva la decomposizione funzionale della funzione della funzione *Avoid Obstacles,* dove anche in questo caso le funzioni e i flussi di informazioni sono coerenti con quanto descritto nel sottocapitolo 6.1.2.

7.2 Semplificazioni considerate

Una volta definita l'architettura logica e l'analisi funzionale del sistema AI-SODIA, si è deciso di adottare alcune semplificazioni, finalizzate a rendere più agevole la modellazione e l'integrazione tra Cameo Systems Modeler e Simulink (trattandosi comunque di un primo approccio all'integrazione tra i due sistemi).

In particolare, è stato deciso di concentrare lo sviluppo esclusivamente sulla componente di rilevamento ed evitamento degli ostacoli (*Detect & Avoid*), tralasciando la fase di identificazione degli stessi. Tale scelta è motivata dal fatto che l'identificazione degli ostacoli, sebbene parte integrante del comportamento completo del sistema, risulta non essenziale ai fini dell'integrazione tra i due ambienti di modellazione. Inoltre, tale funzionalità è generalmente implementata tramite tecniche di intelligenza artificiale, che esulano dal perimetro di modellazione strutturale e comportamentale trattato in questa fase. (Il sottosistema di identification, come il sottosistema relativo allo SLAM, non verranno più trattati a partire da questo punto in poi nel seguente lavoro).

La semplificazione adottata consente di focalizzarsi sugli aspetti più rilevanti per la simulazione e la verifica del comportamento del sistema, in particolare sulla gestione delle traiettorie, sulla rilevazione della presenza di ostacoli e sull'attuazione delle manovre di evitamento. Questo approccio ha permesso di ridurre la complessità del modello, facilitando l'interoperabilità tra Cameo e Simulink e rendendo più efficiente il processo di validazione.

Inoltre, si è scelto di non modellare i sensori fisici, ma di inserire manualmente i dati utili ai modelli per poter garantire la co-simulazione. Questa scelta è stata dettata dalla volontà di testare in modo diretto ed efficace il comportamento del modello e, soprattutto, di valutare la robustezza dell'integrazione tra Cameo e Simulink. L'inserimento manuale dei dati ha permesso di simulare scenari specifici e controllati, facilitando l'analisi del sistema e l'individuazione di eventuali criticità nella trasmissione e gestione delle informazioni tra i due ambienti.

In questo modo, è stato possibile concentrarsi esclusivamente sulla logica funzionale del comportamento *Detect & Avoid*, evitando la complessità aggiuntiva derivante dalla simulazione di un sottosistema sensoriale, e garantendo al contempo un controllo completo sui dati in ingresso alla simulazione.

A supporto di questa visione, e in linea con le semplificazioni adottate per rendere il sistema più gestibile e focalizzato sul caso d'uso oggetto di studio, è stato realizzato un nuovo Block Definition Diagram semplificato del sistema SODIA, che riflette la struttura attuale del modello e ne evidenzia i sottosistemi effettivamente considerati in questa fase (fig.32). Questa rappresentazione consente

di mantenere coerenza tra la modellazione strutturale e quella funzionale, facilitando l'integrazione con i modelli Simulink e ponendo le basi per una futura estensione del sistema.

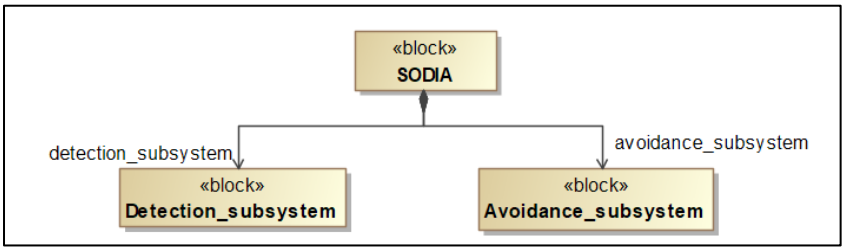

Fig.32 bdd SODIA semplificato

CAPITOLO 8 I Modelli – System & Design

Nel presente capitolo viene illustrato il modello sviluppato in Cameo Systems Modeler rappresentante il progetto di SODIA (*Detect & Avoid Obstacle System*), con particolare attenzione alla sua architettura logica e funzionale. In questa configurazione, Simulink viene impiegato esclusivamente come motore di calcolo, integrato al modello Cameo tramite il motore di simulazione MATLAB, con l'obiettivo di mantenere all'interno di Cameo la gestione della logica sistemica, mentre l'elaborazione numerica e la simulazione dinamica vengono demandate a Simulink.

L'integrazione tra i due ambienti consente di realizzare un flusso di simulazione distribuito, in cui i modelli Simulink, derivati dalla scomposizione del sistema in sottosistemi autonomi, verranno richiamati e simulati singolarmente, producendo i risultati di una singola iterazione per ciascun passo di simulazione. Cameo, in questo contesto, assumerà il ruolo di esoscheletro del sistema simulato, gestendo in modo centralizzato il flusso delle informazioni, il controllo temporale e la logica di interazione tra i sottosistemi, ricostruendo il loop dinamico in maniera orchestrata e coerente con la struttura architetturale definita. Gli obiettivi principali di questo modello sono:

- realizzare l'integrazione e la co-simulazione tra Cameo Systems Modeler e MATLAB/Simulink;
- descrivere in modo completo l'architettura del sistema all'interno di Cameo, inclusa la gestione delle variabili;
- utilizzare i modelli Simulink per calcolare nuove variabili da impiegare durante la simulazione (loop dinamico);

A partire da questa sezione, vengono presentati i modelli sviluppati nel corso del lavoro, illustrando le assunzioni adottate, le semplificazioni introdotte e i risultati ottenuti, con l'intento di fornire una panoramica chiara e progressiva del processo di modellazione, dalla fase iniziale fino all'integrazione tra gli strumenti utilizzati. In una prima fase, si è scelto di procedere con la realizzazione del modello in Simulink, adottando alcune accortezze legate al contesto operativo e semplificando il sistema per concentrarsi sulle funzionalità di rilevamento e manovra di evitamento degli ostacoli, in quanto rappresentano il nucleo del caso d'uso trattato.

Lo sviluppo iniziale è stato condotto utilizzando la licenza MATLAB Student fornita dal Politecnico di Torino, il che ha comportato alcune limitazioni in termini di funzionalità e interoperabilità, ma ha comunque permesso di costruire un modello capace di rappresentare efficacemente il comportamento del drone in ambienti bidimensionali e tridimensionali. Una volta completato il modello Simulink, è stata avviata una prima prova di integrazione con Cameo, con l'obiettivo di verificare la fattibilità tecnica dell'operazione e osservare come il sistema MBSE potesse gestire lo scambio di informazioni con il simulatore dinamico.

Questa fase ha permesso di identificare i primi vincoli e le potenzialità dell'integrazione, fornendo indicazioni utili per le successive iterazioni e per una futura estensione del framework. Come verrà illustrato nelle sezioni seguenti, si è cercato di approfondire progressivamente il livello di dettaglio del modello, al fine di comprendere i limiti effettivi dell'interoperabilità tra Cameo e Simulink e valutare il grado di applicabilità di tale integrazione in un contesto di progettazione sistemica.

8.1 Modellazione su Simulink

Nonostante aver definito l'architettura logica del modello completo ed essere passati a un modello semplificato in modo da poter approcciare in modo più semplice l'integrazione tra i modelli, prima di continuare la modellazione del sistema su Cameo System Modeler è stato necessario comprendere come fosse modellabile funzionalmente il caso d'uso in studio, quindi si è passati alla modellazione di "*Detect & Avoid Obstacles*" su Simulink in modo da comprendere cosa dovrà essere implementato in Cameo per garantire l'integrazione (che tipo di variabili mi servono? Quante sono? Quanti modelli devo intengrare?).

Innanzitutto, bisogna soffermarsi sulla definizione del caso d'uso preso in esame, ovvero la capacità del sistema di rilevare ed evitare ostacoli (Detect & Avoid Obstacles). A partire da questa esigenza funzionale, è stato necessario progettare un modello in grado di simulare tale comportamento, ossia un sistema che, durante il volo, sia in grado di rilevare la presenza di ostacoli lungo la propria traiettoria e, qualora necessario, attuare manovre correttive per evitarli in modo sicuro ed efficiente. Per poter simulare in modo efficace il comportamento di rilevamento degli ostacoli, è stato necessario definire una logica semplice ma funzionale, capace di rappresentare le dinamiche essenziali dell'interazione tra il drone e l'ambiente circostante. L'idea di base è che il drone, durante il volo, debba essere in grado di percepire la presenza di ostacoli lungo la propria traiettoria e, in funzione della distanza che lo separa da essi, decidere se attuare o meno una manovra di evitamento. [25]

Il primo passo per costruire questa logica è stato quello di rappresentare la posizione del drone e quella dell'ostacolo come funzioni del tempo; in questo modo è possibile calcolare, in ogni istante, la distanza relativa tra i due, utilizzando una semplice formula euclidea.

$$d = \sqrt{(xdrone} - xobs)\text{^}2 + (ydrone - yobs)\text{^}2$$

Tale distanza, costituisce il parametro chiave per determinare se il drone si trova in una situazione di potenziale pericolo.[25]

A partire da questa distanza, viene introdotta una soglia di rilevamento, ovvero un valore limite al di sotto del quale si considera che l'ostacolo sia sufficientemente vicino da richiedere un'azione correttiva. Se la distanza calcolata scende al di sotto di questa soglia, il sistema interpreta la situazione come critica e attiva la logica di evitamento. In caso contrario, il drone prosegue il proprio volo senza modificare la traiettoria. [25] (fig.33)

Questa impostazione, pur nella sua semplicità, consente di simulare un comportamento reattivo e realistico, in cui il drone è in grado di adattarsi dinamicamente all'ambiente, riconoscendo la presenza di ostacoli e reagendo in modo coerente.

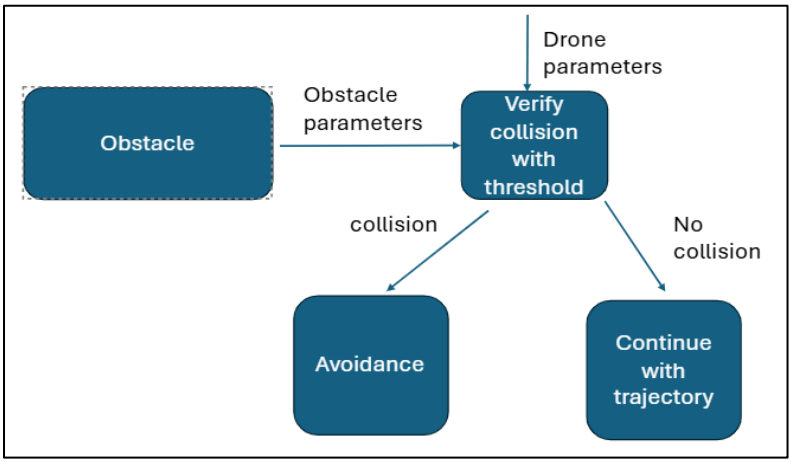

Fig.33 Logica adottata [25]

La logica di rilevamento e gestione delle collisioni è stata strutturata a partire da un modello semplificato (fig.34) in cui si assume che sia il drone che l'ostacolo seguano un moto lineare uniforme. In questo contesto, il sistema necessita informazioni relative alla posizione iniziale e alla velocità di entrambi gli oggetti. Una volta definiti questi parametri, la simulazione procede iterando nel tempo, secondo un intervallo temporale e una durata totale, per verificare in ogni istante la distanza tra drone e ostacolo. [25]

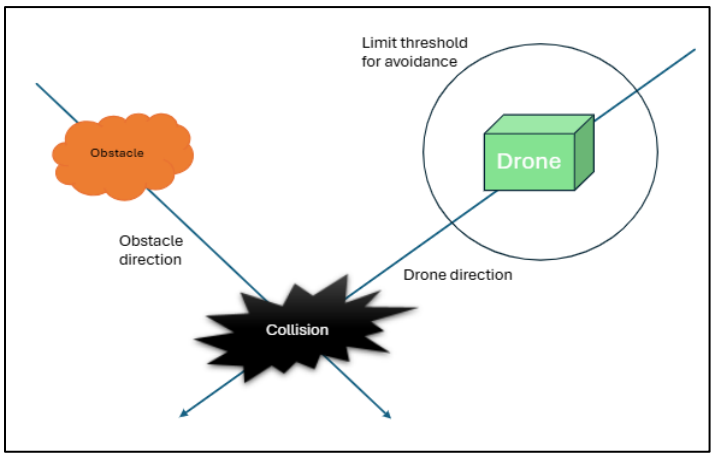

Fig.34 Fisica dell'evento [25]

8.1.1 PRIMO APROCCIO SU SIMULINK, Modello statico 2D

Per la realizzazione del modello Simulink si è adottato un approccio incrementale, partendo dal caso più semplice: la presenza di ostacoli statici in un ambiente bidimensionale. Questo consente di costruire progressivamente il modello fino a giungere alla configurazione finale, che simulerà il caso d'uso "*Detect & Avoid Obstacles*" in ambiente tridimensionale.

In questa prima fase, sono stati definiti gli input necessari, quindi, come anticipato nel paragrafo precedente, è richiesta la conoscenza della posizione del drone in ogni istante temporale; tale informazione è ottenuta mediante l'utilizzo di blocchi Integrator di Simulink, collegati direttamente alle velocità lungo gli assi X e Y, assunte costanti nel presente caso studio. Inoltre, sono state introdotte le posizioni degli ostacoli statici (coordinate X e Y), al fine di poterli allocare nello spazio e consentire l'analisi della loro interazione con il drone.

Successivamente, si è scelto di implementare le funzioni di *Detect & Avoid* mediante l'utilizzo di due blocchi MATLAB Function all'interno del modello Simulink, scrivendo codice in grado di rappresentare la logica progettata per ciascuna funzione.

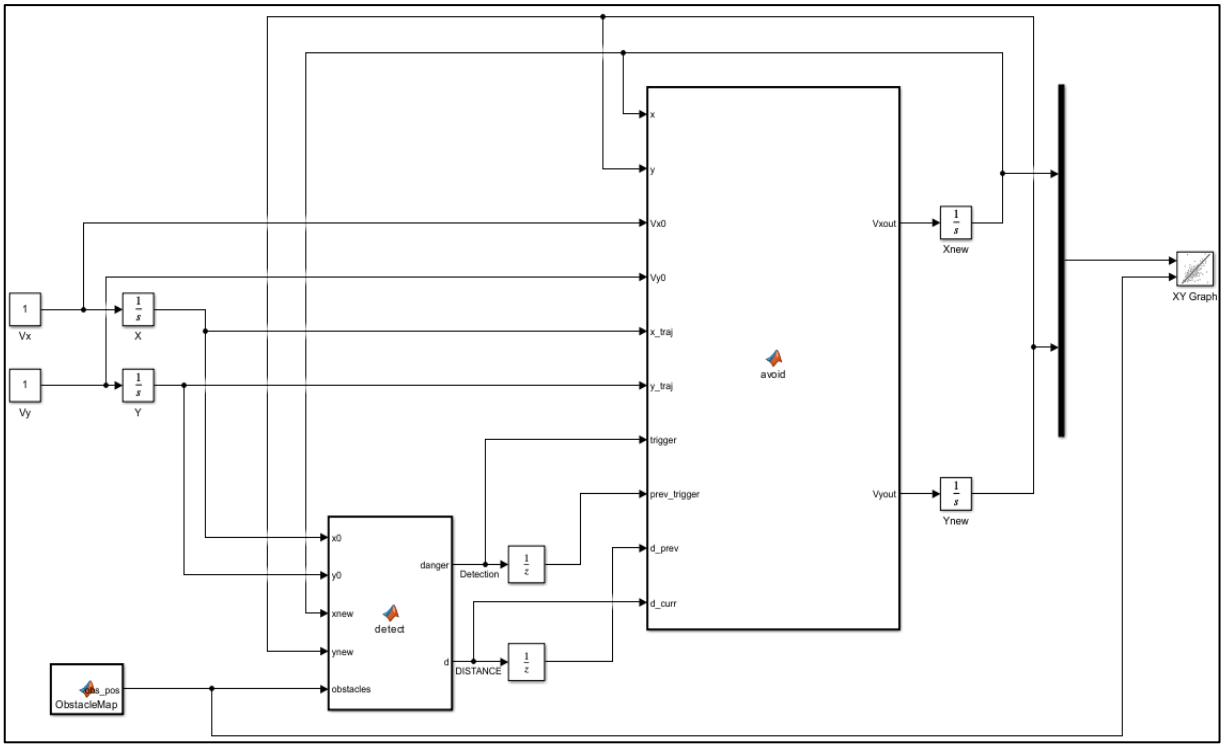

Fig.35 detect and avoid obstacle, 2D static model

La funzione di *Detect* si basa sul rilevamento del pericolo attraverso la distanza tra il drone e una serie di ostacoli statici, come anticipato nei paragrafi precedenti; essa riceve in ingresso le coordinate attuali e quelle nuove del drone (x0, y0, xnew, ynew), oltre a una matrice contenente le posizioni degli ostacoli (obstacles). In figura 35 è possibile osservare il blocco corrispondente alla funzione detect, con i relativi ingressi e uscite.

L'utilizzo sia delle posizioni iniziali che di quelle aggiornate del drone è motivato dal fatto che, una volta attivata la funzione Avoid, il drone devia dalla traiettoria originaria prevista dai blocchi Integrator, assumendo quindi nuove posizioni. In uscita, la funzione restituisce un indicatore booleano che segnala la presenza di un ostacolo entro il raggio di rilevamento, e la distanza effettiva tra il drone e l'ostacolo più vicino, con quest'ultima che viene calcolata tramite la formula euclidea e confrontata con una soglia prefissata, definita in fase di test; quando la distanza risulta inferiore alla soglia, il marker booleano viene attivato, rappresentando l'evento di Detection.

La funzione avoid è stata progettata per gestire la fase di evitamento di ostacoli e il successivo rientro del drone nella traiettoria originaria. Essa riceve in ingresso le coordinate attuali del drone (x, y), le velocità iniziali lungo gli assi X e Y (Vx0, Vy0), la traiettoria desiderata (x_traj, y_traj), due segnali booleani di attivazione (trigger, prev_trigger) e le distanze dal più vicino ostacolo nei due istanti consecutivi (d_prev, d_curr).

La logica della funzione si articola in tre fasi principali:

- Evitamento: Quando il segnale di trigger passa da falso a vero, indicando l'ingresso in una zona di pericolo, viene modificata la velocità del drone per deviare dalla traiettoria. Se la distanza corrente dall'ostacolo è inferiore rispetto a quella precedente, viene applicata una correzione alla velocità lungo l'asse Y, mentre la velocità lungo X viene ridotta. In caso contrario, le velocità rimangono invariate.
- Inizio del rientro: Quando il segnale di trigger torna a falso, viene attivata la fase di rientro alla traiettoria originaria.

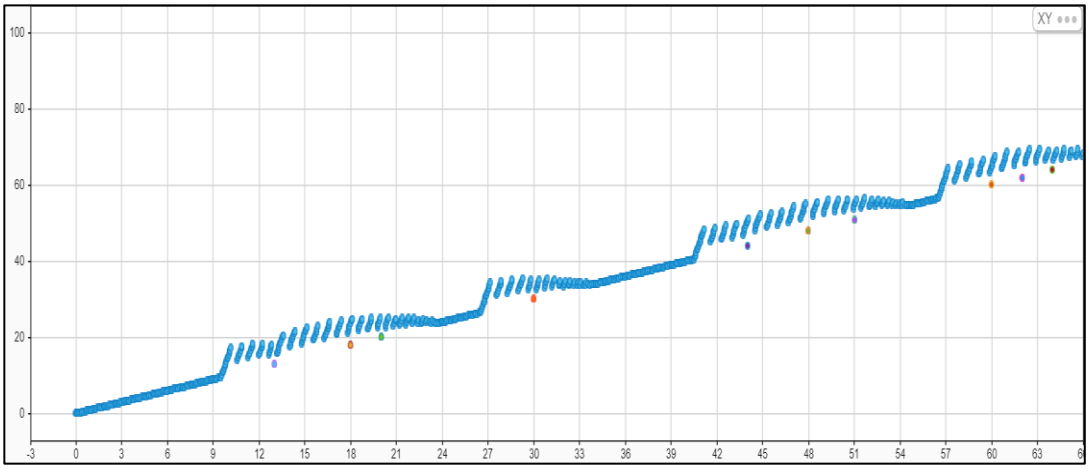

Fig.36 plot risultante della simulazione di detect and avoid obstacle, 2D static model

- Rientro alla traiettoria: Durante questa fase, viene calcolato l'errore tra la posizione attuale del drone e la traiettoria desiderata. Se tale errore è inferiore a una soglia (0.01), il rientro è considerato completato e le velocità vengono ripristinate ai valori iniziali. Altrimenti, viene applicata una correzione proporzionale per guidare il drone verso la traiettoria.

La funzione restituisce in uscita le velocità corrette lungo X e Y (Vxout, Vyout), che vengono poi integrate per ricavare le nuove posizioni del drone.

Figura 36 mostra il risultato della simulazione del primo modello "detect and avoid obstacle" in un contesto bidimensionale statico; gli ostacoli sono stati posizionati intenzionalmente lungo la traiettoria del drone, la quale è determinata dall'integrazione nel tempo delle velocità costanti lungo gli assi X e Y (entrambe pari a 1 m/s). Quando il drone rileva la presenza di un ostacolo, ovvero quando la distanza tra il drone e l'ostacolo scende al di sotto di una soglia prestabilita, viene attivata la procedura di evitamento che consiste in un movimento laterale, con un incremento della velocità lungo l'asse Y, seguito da un ritorno graduale alla traiettoria originaria.

8.1.2 Modello Statico 3D

Il passo successivo, una volta ottenuto questo primo risultato, è stato trasformare l'ambiente di simulazione in un ambiente tridimensionale. Quindi è stato aggiunto un nuovo input per il drone, ovvero la velocita lungo Z, e conseguentemente la posizione lungo Z; inoltre, anche agli ostacoli è stata aggiunta la nuova coordinata lungo Z.

Per quanto riguarda le Matlab function invece, sono stati effettuati degli adattamenti per considerare i nuovi input, ad esempio per quanto riguarda il blocco detection la formula euclidea è stata modificata in modo da considerare la terza coordinata:

```
D =sqrt ((x - obstacles(i,1)) ^2 + (y - obstacles(i,2)) ^2 + (z - obstacles(i,3)) ^2)
```

Per quanto riguarda la fase di avoid, la logica di evitamento è rimasta invariata rispetto al modello precedente: si continua a utilizzare un movimento lungo l'asse Y per aggirare ostacoli statici; tuttavia, nella fase di ritorno alla traiettoria originaria, si introduce una novità significativa: viene ora considerata anche la componente lungo l'asse Z, rendendo il rientro tridimensionale e più aderente alla dinamica reale del drone. (fig.37)

Figura 38 mostra il grafico relativo al nuovo modello tridimensionale, in cui sono rappresentate le variazioni delle posizioni nel tempo lungo i tre assi spaziali. In questa simulazione, le velocità iniziali lungo gli assi X e Y sono rimaste invariate rispetto al modello precedente, mentre la nuova componente di velocità lungo l'asse Z è stata impostata a zero. Anche la disposizione degli ostacoli non è stata modificata, con le coordinate lungo Z fissate anch'esse a zero.

Dal grafico si osserva chiaramente come la posizione lungo l'asse Y (linea blu) vari ogni volta che il drone incontra uno o più ostacoli, mentre non si rilevano deviazioni lungo gli assi X (linea gialla) e Z (linea rossa) rispetto alla traiettoria originale, poiché la logica implementata non prevede alcuna manovra di evitamento in queste direzioni.

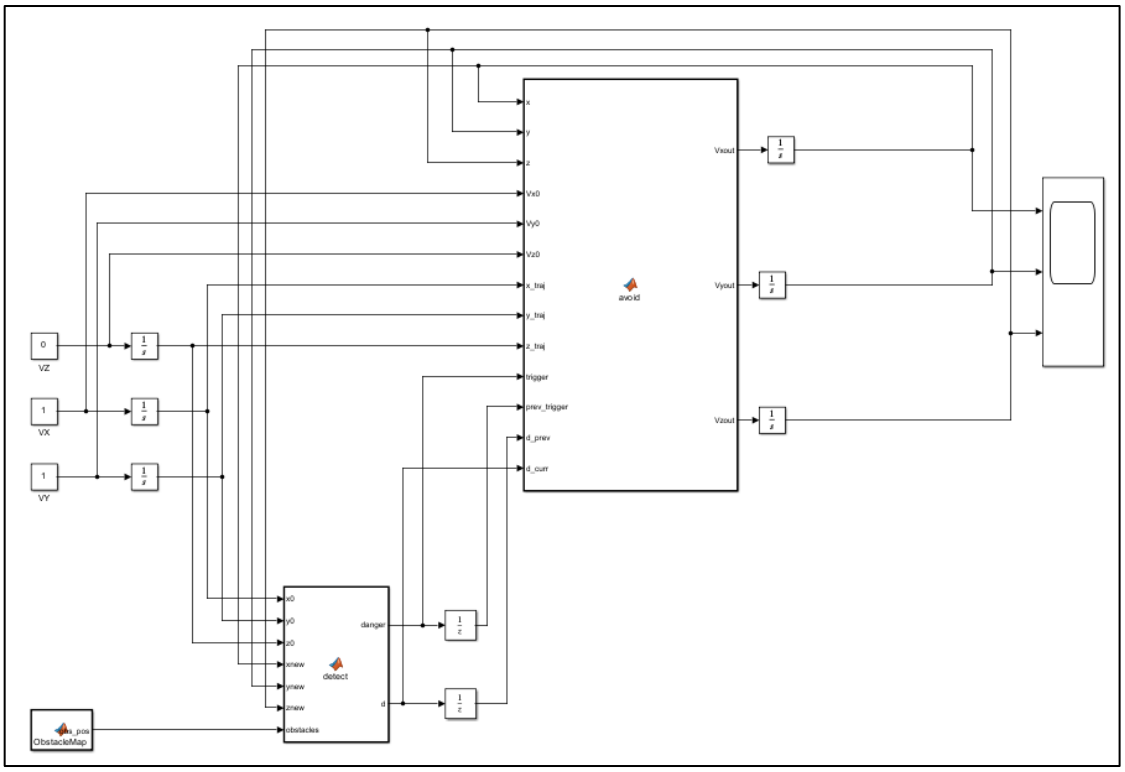

Fig.37 detect and avoid obstacle, 3D static model

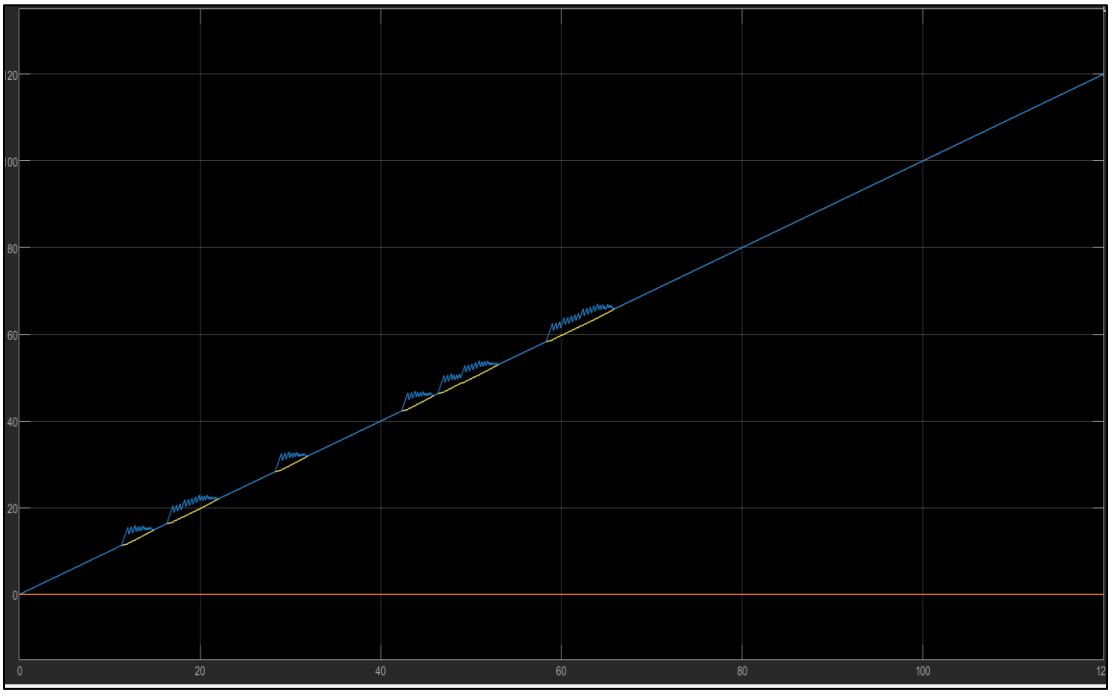

Fig.38 plot delle posizioni nel tempo del drone lungo i 3 assi, detect and avoid obstacle, 3D static model

8.1.3 Modello detect and avoid obstacle, con ostacoli mobili e fissi in ambiente tridimensionale

Il passo successivo ha riguardato lo sviluppo di un modello avanzato, capace di rilevare sia ostacoli statici che dinamici in un ambiente tridimensionale, e di eseguire manovre di evitamento differenziate in base alla natura dell'ostacolo rilevato.

Per la progettazione di questo modello, si è fatto riferimento all'architettura strutturale del sottosistema di detection illustrata in fig.26, nella quale sono identificati specifici componenti la cui logica può essere riprodotta e implementata in Simulink. Questo approccio consente di ampliare le funzionalità del sistema, rendendolo capace di gestire anche ostacoli dinamici, attraverso l'introduzione di nuove variabili e comportamenti che tengano conto di parametri aggiuntivi, come la velocità e la traiettoria degli oggetti rilevati (si nota che queste variabili vengono stimante, simulando il comportamento del sistema il piu realisticamente possibile). L'integrazione di ostacoli mobili, infatti, impone una revisione del modello esistente, affinché sia in grado di distinguere tra una semplice rilevazione entro un certo raggio e una reale possibilità di impatto, aspetto che si rivela cruciale per garantire una risposta efficace e contestualizzata.

Per affrontare questa esigenza, si è resa necessaria l'introduzione di un sensore anticollisione, progettato per valutare non solo la presenza di un ostacolo, ma anche il rischio effettivo di collisione, superando la logica binaria di rilevamento e introducendo una valutazione più sofisticata basata sulla dinamica relativa tra drone e ostacolo. La sola presenza di un oggetto nel campo di rilevamento, infatti, non implica automaticamente una situazione di pericolo, ed è proprio questa distinzione che consente al sistema di adottare comportamenti più intelligenti e selettivi.

Sulla base delle considerazioni precedenti, sono state sviluppate tre nuove MATLAB Function, ciascuna con uno scopo specifico all'interno del modello tridimensionale avanzato. Le prime due funzioni sono dedicate alla stima della traiettoria e della velocità degli ostacoli, parametri fondamentali per la gestione di oggetti dinamici in movimento, mentre la terza funzione è stata progettata per simulare il comportamento di un sensore di collisione, ispirandosi alla logica già implementata nel blocco detect.

In particolare, questa funzione di collisione confronta la distanza tra drone e ostacolo, calcolata tramite la formula euclidea, con una soglia di sicurezza; a differenza del blocco detect, che si attiva quando l'ostacolo entra in un raggio di rilevamento più ampio, il sensore di collisione utilizza una soglia più restrittiva, certificando la probabilità effettiva di impatto. Questo approccio consente di distinguere tra una semplice rilevazione e una situazione di pericolo reale, migliorando la precisione e l'efficacia delle manovre di evitamento.

Infine, per integrare correttamente queste nuove funzionalità, è stato necessario modificare i blocchi esistenti del modello, adattandoli alle nuove logiche e ai parametri introdotti; questo processo ha permesso di ottenere un sistema più robusto e flessibile, capace di gestire scenari complessi con ostacoli sia statici che dinamici.

In figura 39 è rappresentato il modello di detect and avoid obstacle, per ostacoli statici e dinamici in ambiente tridimensionale.

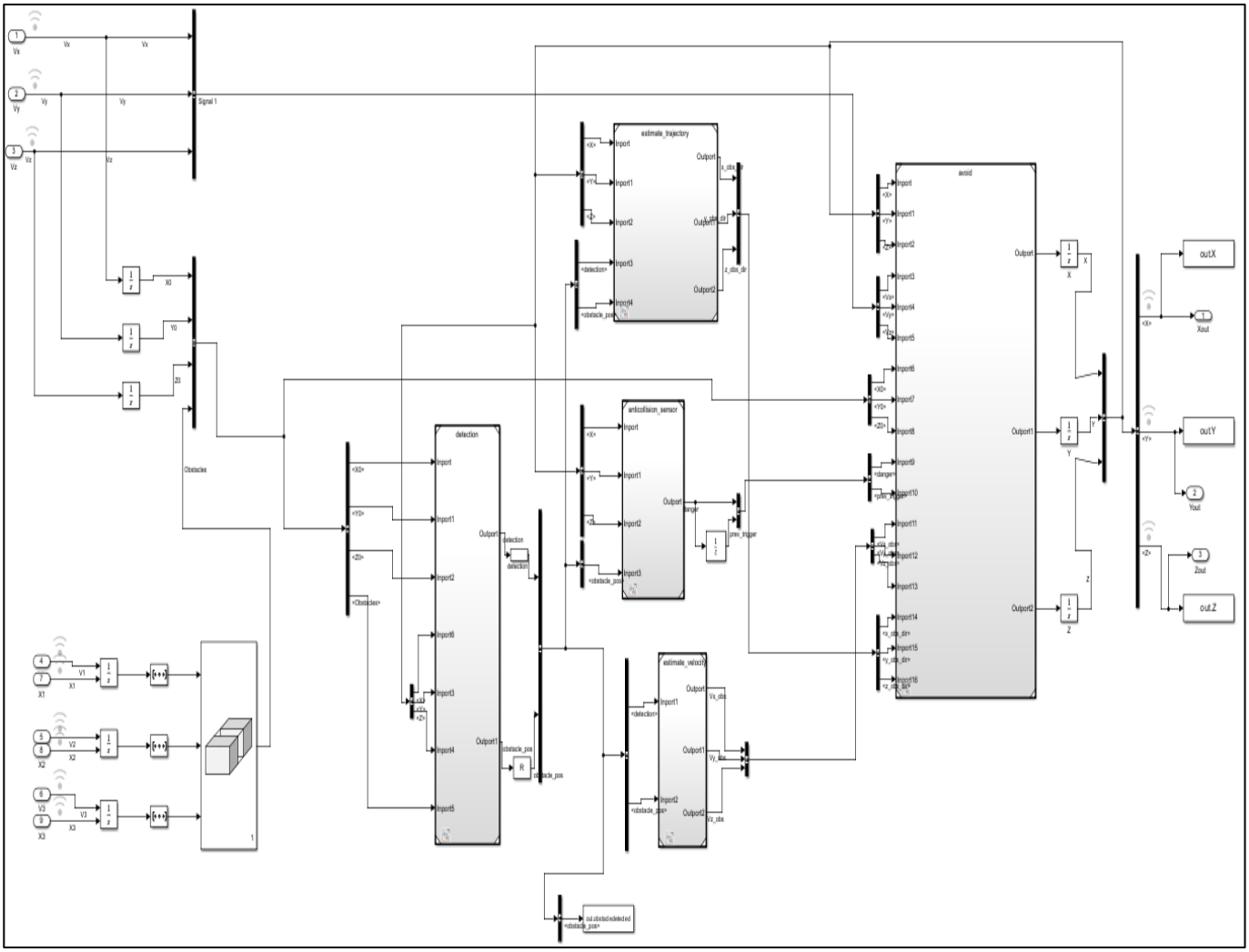

Fig.39 detect and avoid obstacle model, ostacoli di vario tipo in ambiente tridimensionale

Nel nuovo modello sviluppato, dal lato drone, vengono forniti gli stessi input già definiti nei modelli precedenti, ovvero le tre componenti di velocità lungo gli assi X, Y e Z. Questi input vengono trasmessi ai blocchi Integrator, i quali, tramite l'integrazione nel tempo, restituiscono la traiettoria desiderata del drone in ogni istante della simulazione.

Per la gestione degli input, sono stati utilizzati blocchi Inport, che permettono di introdurre i dati esternamente, in previsione della futura integrazione con Cameo Systems Modeler, il quale sarà responsabile della fornitura dei dati necessari alla simulazione.

Per quanto riguarda la modellazione degli ostacoli (come illustrato in figura 40), anche in questo caso la loro configurazione è stata definita manualmente, con l'obiettivo di testare la funzionalità del sistema. Considerando che gli ostacoli possono essere dinamici, per ciascuno di essi è stato necessario specificare:

- le velocità dei tre ostacoli (V_1,V_2, V_3),
- la posizione iniziale al momento dell'avvio della simulazione dei tre ostacoli (X_1, X_2, X_3).

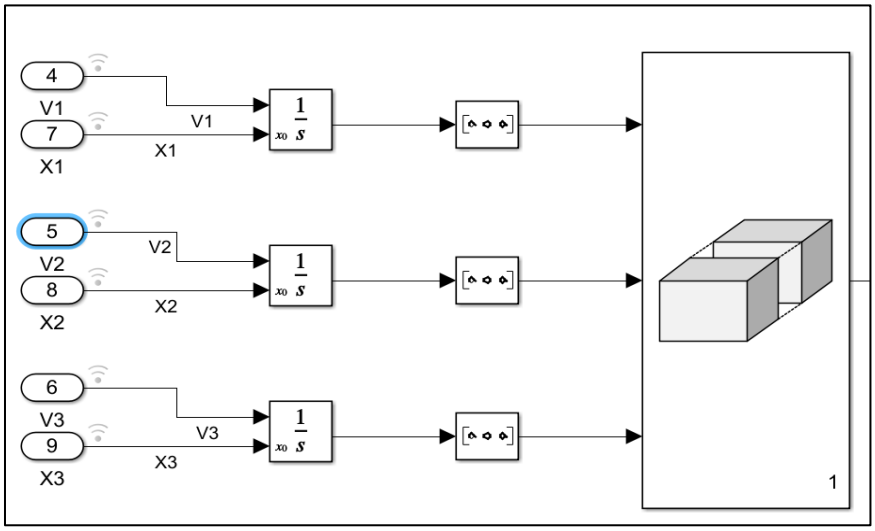

Fig.40 introduzione degli ostacoli nel Simulink model

Questi dati vengono elaborati dai rispettivi blocchi Integrator, che calcolano la posizione degli ostacoli istante per istante; i vettori risultanti (di dimensione 1×3) rappresentano le coordinate spaziali di ciascun ostacolo. Per garantire la compatibilità dimensionale con i blocchi successivi, ogni vettore viene trasformato in un vettore riga tramite il blocco Reshape.

Successivamente, tutti i vettori vengono aggregati nel blocco Concatenate Matrix, generando una matrice di dimensione N×3, dove N rappresenta il numero totale di ostacoli, che contiene le posizioni di tutti gli ostacoli in ogni istante temporale, ed è utilizzata per le successive elaborazioni nel modello.

Un ulteriore miglioramento introdotto in questo nuovo modello, rispetto alle versioni precedenti, riguarda l'utilizzo dei blocchi Bus Creator e Bus Selector, come illustrato in figura 39. Questa scelta è stata motivata dalla crescente complessità del sistema e dal numero sempre maggiore di variabili in gioco, che rendevano difficile mantenere una struttura ordinata e leggibile del modello.

Il blocco Bus Creator consente di raggruppare più segnali in un'unica linea di connessione, creando un bus virtuale che semplifica la gestione e la visualizzazione dei dati; in questo modo, è possibile trasmettere simultaneamente diverse variabili correlate senza doverle collegare singolarmente, riducendo notevolmente il disordine visivo nel modello Simulink.

Il blocco Bus Selector, invece, permette di estrarre specifici segnali da un bus precedentemente creato, funzionaltità particolarmente utile quando si desidera accedere solo ad alcune variabili all'interno di un gruppo, mantenendo comunque la struttura modulare e compatta del modello. L'adozione di questi blocchi ha contribuito in modo significativo a migliorare la leggibilità, la manutenibilità e la scalabilità del modello, rendendolo più adatto a gestire scenari complessi e facilitando l'integrazione con altri strumenti, come Cameo Systems Modeler.

In linea con le pratiche aziendali, tutte le funzioni MATLAB sono state inserite all'interno di Referenced Models. Ogni modello referenziato contiene un blocco MATLAB Function che implementa la logica desiderata, insieme a blocchi Inport e Outport dedicati alla gestione degli ingressi e delle uscite; questo approccio consente di mantenere una struttura ordinata e modulare, facilitando la leggibilità, la manutenibilità e la scalabilità del progetto, inoltre, l'uso dei Referenced Models permette una più agevole integrazione con ambienti esterni come Cameo, e rispecchia il metodo di lavoro adottato in azienda, dove la separazione funzionale tra i componenti è considerata una buona pratica consolidata.

La funzione detect, come precedentemente detto, è stata sviluppata per identificare la presenza di ostacoli nelle vicinanze di un oggetto mobile nello spazio tridimensionale; infatti, riceve in ingresso le coordinate iniziali e aggiornate del drone, insieme a una matrice contenente le posizioni degli ostacoli e determina se uno o più di questi ostacoli si trova entro una distanza predefinita (soglia) rispetto alla posizione corrente del drone.

La funzione utilizza una variabile persistent chiamata detection_triggered, che consente di mantenere lo stato tra chiamate successive. In fase di inizializzazione, se questa variabile non è ancora definita, viene impostata a false; questo meccanismo permette alla funzione di distinguere tra la prima fase di rilevamento (basata sulle coordinate iniziali) e le successive (basate sulle coordinate aggiornate).

La soglia di rilevamento è stata fissata a dieci unità a fine di test; per ogni ostacolo presente nella matrice, viene calcolata la distanza euclidea rispetto alla posizione corrente dell'oggetto e se tale distanza è inferiore alla soglia e minore rispetto a tutte le distanze precedentemente calcolate, l'ostacolo viene considerato rilevato. In tal caso, la funzione restituisce:

- detection = true, indicando che un ostacolo è stato rilevato;
- obstacle_pos, contenente le coordinate dell'ostacolo/i che ha fatto cambiare di stato detection.

Una volta che un ostacolo è stato rilevato, detection_triggered viene impostata a true, e le chiamate successive alla funzione utilizzeranno le coordinate aggiornate dell'oggetto per continuare il monitoraggio (questo perchè l'ostacolo esce dalla traiettoria desiderata per evitare l'ostacolo, quindi, assume una posizione diversa rispetto all'output del blocco integrator).

La funzione estimate_trajectory è stata sviluppata con l'obiettivo di calcolare la direzione dell'ostacolo più vicino rispetto alla posizione attuale del drone, all'interno di uno spazio tridimensionale. Questo tipo di informazione è fondamentale per la stima della traiettoria dell'ostacolo e per la successiva fase di pianificazione del movimento del drone.

La funzione viene attivata solo nel caso in cui sia stata confermata la presenza di almeno un ostacolo, ovvero quando la variabile booleana detection è impostata su true, in caso contrario, la funzione termina immediatamente, restituendo una direzione nulla lungo tutti e tre gli assi spaziali.

Gli input della funzione sono:

- Le coordinate attuali del drone (x, y, z);
- la variabile booleana detection, che indica se è stato rilevato un ostacolo;
- Una matrice obstacles, contenente le coordinate spaziali degli ostacoli rilevati (ogni riga rappresenta un ostacolo con coordinate [x, y, z]).

Nel caso in cui il rilevamento sia attivo, la funzione procede a calcolare la distanza euclidea tra il drone e ciascun ostacolo presente nella matrice, viene quindi identificato l'ostacolo più vicino e, rispetto ad esso, viene calcolato un vettore direzionale normalizzato che punta dal drone verso l'ostacolo. Questo vettore rappresenta la direzione dell'ostacolo nello spazio tridimensionale e può essere utilizzato per stimare la sua traiettoria o per analizzare la sua posizione relativa.

Il risultato della funzione è espresso tramite tre componenti:

- x_obs_dir: direzione lungo l'asse X;
- y_obs_dir: direzione lungo l'asse Y;
- z_obs_dir: direzione lungo l'asse Z.

Questi valori costituiscono un vettore unitario che descrive la direzione dell'ostacolo più vicino rispetto alla posizione corrente del drone.

La funzione collision invece, ha lo scopo di verificare la presenza di una situazione di pericolo imminente tra il drone e uno o più ostacoli nello spazio tridimensionale; a differenza della funzione di rilevamento (detect), questa funzione è pensata per attivarsi in una fase successiva, quando il drone si trova molto vicino a un ostacolo e l'evitamento deve essere immediato.

Gli input della funzione sono:

- le coordinate attuali dell'oggetto (x, y, z);
- una matrice obstacle_pos contenente le posizioni degli ostacoli rilevati (se è solo un ostacolo la matrice sarà un vettore riga), ciascuna espressa come una riga nel formato [x, y, z].

La funzione utilizza una soglia di distanza pari a 2.0 m, quindi per ciascun ostacolo, viene calcolata la distanza euclidea rispetto alla posizione dell'oggetto e se almeno un ostacolo si trova entro questa soglia, la funzione considera la situazione come pericolosa e restituisce:

- danger = 1, indicando una potenziale collisione.
- In caso contrario, danger = 0 segnalando l'assenza di pericolo immediato.

Il segnale danger, prodotto dalla funzione collision, viene passato a un blocco Unit Delay (fig. 41) in Simulink. Questo ritardo è necessario per garantire che la funzione di evitamento (avoid) venga attivata solo dopo che la condizione di pericolo è stata correttamente rilevata e stabilizzata nel tempo (altrimenti tenderebbe a rincominciare la funzione di avoid ogni tal volta entri un segnale danger pari a 1).

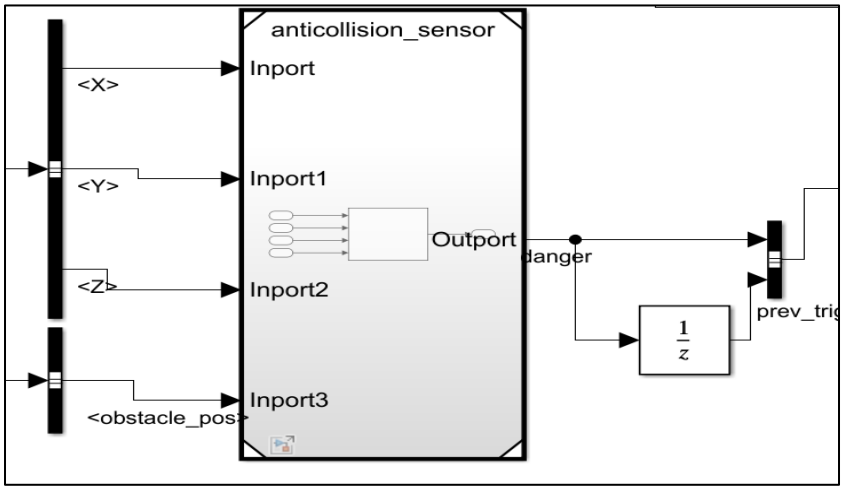

Fig.41 Anticollision_sensor Matlab function, con relativi input output e unit delay block

La funzione estimate_obstacle_velocities è progettata per stimare la velocità degli ostacoli rilevati nello spazio tridimensionale. Essa si attiva solo quando il segnale di rilevamento (detection) è attivo, e utilizza una variabile persistent chiamata prev_obstacles per memorizzare la posizione degli ostacoli al passo precedente della simulazione.

Gli input della funzione infatti sono:

- detection: variabile booleana che indica se è stato rilevato almeno un ostacolo;
- obstacle_pos: matrice contenente le posizioni attuali degli ostacoli, ciascuna espressa come una riga nel formato [x, y, z].

Se prev_obstacles non è ancora inizializzata, viene impostata uguale a obstacle_pos, mentre quando detection è attivo, la funzione calcola la differenza tra le posizioni attuali e quelle precedenti degli ostacoli, e divide tale differenza per un intervallo temporale fisso dt = 0.1 (impostato a questo valore a scopo di test), ottenendo così una stima della velocità lungo ciascun asse spaziale:

- Vx_obs velocità stimata lungo X.
- Vy_obs velocità stimata lungo Y.
- Vz_obs velocità stimata lungo Z.

La funzione avoid, come per le versioni del modello precedenti, rappresenta il nucleo operativo del comportamento di evitamento nel sistema di navigazione simulato; essa è progettata per modificare la traiettoria di un oggetto mobile in risposta alla rilevazione di ostacoli, garantendo un movimento fluido e sicuro. La funzione riceve in ingresso la posizione e la velocità originariamente impostata dell'drone, la traiettoria desiderata (derivata dai blocchi integrator), lo stato del rilevamento (tramite l'output dell'anticollision sensor e del unit delay block, si è deciso di attuare questa logica in modo che l'azione di evitamento inizi unicamente la prima volta che il trigger di collisione diventa positivo e quindi il prev_trigger fornito dall'unit delay quindi riferito all'iterazione precedente sia negativo, evitando così di rincominciare l'evitamento ad ogni iterazione consecutiva in cui il trigger fornito risulta positivo; da notare che si poteva ottenere lo stesso risultato con la logica delle variabili persistent), la velocità stimata degli ostacoli, e la direzione di evitamento calcolata.

Il comportamento della funzione si articola in tre fasi principali:

- Nella prima fase, quando viene rilevato un ostacolo (trigger attivo (anticollision_sensor output) e prev_trigger disattivo (unit delay output)), la funzione decide il tipo di manovra da eseguire. Se l'ostacolo è statico o si muove prevalentemente sul piano orizzontale, viene attivato un evitamento verticale, altrimenti viene eseguito un evitamento laterale. La direzione di manovra viene calcolata in modo da risultare perpendicolare alla direzione dell'ostacolo, e successivamente normalizzata. La velocità desiderata viene ottenuta moltiplicando questa direzione per la velocità attuale del drone, e viene poi interpolata con la velocità precedente per garantire una transizione graduale.
- Nella seconda fase, quando il segnale di rilevamento si disattiva (trigger falso e prev_trigger ancora attivo), la funzione entra in modalità di ritorno alla traiettoria (returning = true). Questo stato indica che l'ostacolo è stato superato e che l'oggetto può iniziare a riprendere il percorso originario.
- La terza fase gestisce il rientro alla traiettoria desiderata, con la funzione calcola l'errore tra la posizione attuale e quella prevista dalla traiettoria (x_traj, y_traj, z_traj); se l'errore è trascurabile, la velocità viene ripristinata ai valori originali, in caso contrario, viene applicato un controllo proporzionale per guidare l'oggetto verso la traiettoria, sempre con interpolazione fluida della velocità.

Per mantenere la coerenza tra i vari cicli di simulazione, la funzione utilizza variabili persistent che conservano lo stato interno, come la velocità modificata (Vx_rot, Vy_rot, Vz_rot) e il flag di ritorno (returning); il risultato finale è una nuova velocità (Vxout, Vyout, Vzout) che tiene conto sia della necessità di evitare ostacoli sia del ripristino della traiettoria originale.

Questa funzione è pensata per essere integrata in un sistema dinamico simulato in Simulink, dove il comportamento dell'oggetto deve adattarsi in tempo reale alla presenza di ostacoli. La logica implementata consente di gestire sia ostacoli statici che dinamici, con manovre fluide e controllate, migliorando la robustezza e la sicurezza del sistema.

Infine, i tre output ottenuti (Vxout, Vyout, Vzout) vengono mandati all'interno di blocchi integrator in modo da ottenere ad ogni iterazione la posizione aggiornata del drone da visualizzare come output di simulazione e da mandare indietro nei vari blocchi precedentemente spiegati in modo da ottenere questo loop dinamico di simulazione (si può vedere in fig.39).

Per verificare il corretto funzionamento del modello e ottenere un riscontro visivo dinamico del comportamento simulato, è stato sviluppato uno script MATLAB dedicato; questo script consente di eseguire la simulazione del modello utilizzando dati esterni, definiti come *structure with time* o *array*, e di visualizzare in tempo reale la traiettoria del drone insieme alla posizione degli ostacoli rilevati.

L'animazione 3D generata dallo script permette di osservare l'evoluzione del sistema iterazione per iterazione, offrendo un supporto visivo utile sia per la validazione del modello che per l'analisi qualitativa del comportamento del sistema. In particolare, è possibile monitorare la traiettoria seguita dal drone, il punto attuale e la posizione degli ostacoli, facilitando l'individuazione di eventuali anomalie o comportamenti non desiderati.

Questa visualizzazione si è rivelata particolarmente utile durante le fasi di test e ottimizzazione, poiché consente di confrontare direttamente i risultati della simulazione con le aspettative progettuali, migliorando l'efficienza del processo di sviluppo e integrazione con Cameo Systems Modeler con i quali verranno effettuati anche paragoni.

L'immagine riportata in figura 42 mostra il comportamento del drone durante una manovra di evitamento; si può vedere come il drone modifica la propria traiettoria per evitare un ostacolo mobile che si muove lungo l'asse Z. Lo scenario è stato scelto con l'obiettivo di verificare la capacità del sistema di gestire ostacoli dinamici con moto prevalentemente verticale (evitando lateralmente), simulando condizioni operative realistiche; l'ostacolo in questione presenta una velocità pari a [0 0 10], che ne determina il movimento lungo la direzione verticale.

La visualizzazione consente di osservare in tempo reale la traiettoria percorsa dal drone, la sua posizione istantanea e quella dell'ostacolo rilevato. Questo approccio ha facilitato l'analisi qualitativa del comportamento del modello, rendendo immediatamente evidenti eventuali deviazioni dalla traiettoria nominale, ritardi nella risposta o errori nel processo di rilevamento.

L'animazione ha inoltre svolto un ruolo fondamentale nella validazione delle principali funzioni implementate nel modello, tra cui la rilevazione degli ostacoli (detect), la stima della loro velocità (estimate_obstacle_velocities) e la logica di manovra di evitamento (avoid). La rappresentazione visiva ha permesso di confrontare la traiettoria simulata con quella prevista, confermando la coerenza tra le logiche implementate e il comportamento atteso del sistema.

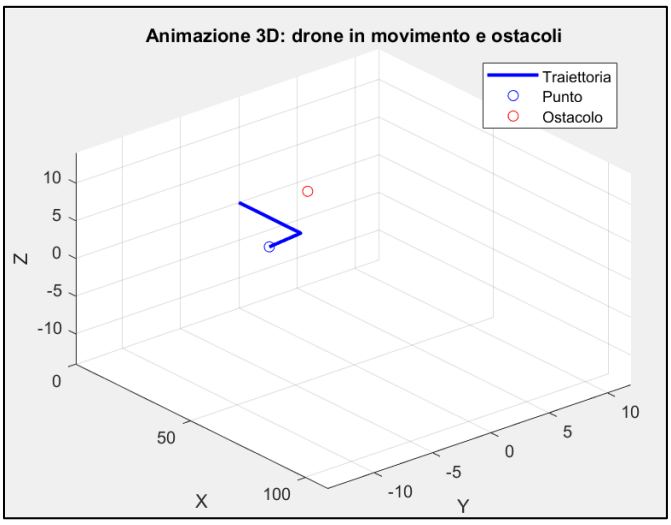

Fig.42 Drone che evita un ostacolo mobile con velocita lungo l'asse Z

La Figura 43 mostra un caso in cui l'ostacolo si muove sul piano, con una velocità lungo l'asse Y; come previsto dalla logica implementata nella funzione MATLAB avoid, il drone reagisce eseguendo una manovra di evitamento verticale, aumentando la propria quota per aggirare l'ostacolo. Questo comportamento conferma il corretto funzionamento della strategia di evitamento programmata, che distingue tra ostacoli statici, verticali e orizzontali, adattando di conseguenza la direzione della manovra.

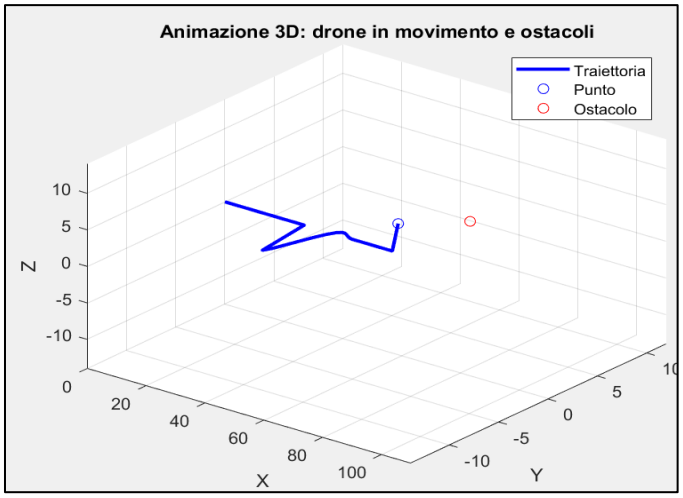

Fig.43 Drone che evita un ostacolo mobile con velocita lungo l'asse Y

La figura 44 mostra un esempio di manovra di evitamento eseguita dal drone in presenza di un ostacolo fisso. In questo caso, il comportamento del sistema rispecchia pienamente quanto previsto dalla logica implementata nella funzione avoid: il drone rileva l'ostacolo statico e modifica la propria traiettoria per evitarlo in modo fluido e sicuro lateralmente; la rappresentazione grafica evidenzia chiaramente la traiettoria del drone (linea blu), la sua posizione attuale (cerchio blu) e quella dell'ostacolo (cerchio rosso), confermando il corretto funzionamento del modello.

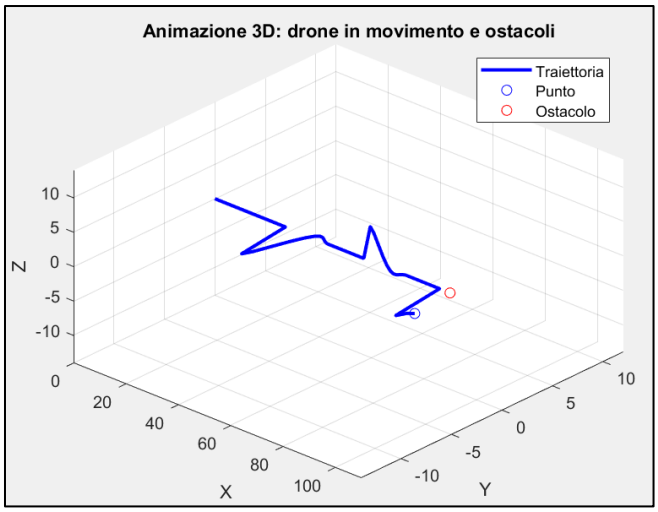

Fig.44 Drone che evita un ostacolo fisso

L'insieme delle simulazioni e delle animazioni analizzate dimostra che tutte le funzioni principali del sistema (rilevamento, stima dei parametri degli ostacoli e manovra di evitamento) operano come previsto; il comportamento osservato è coerente con le logiche progettuali, validando l'efficacia del modello e la sua capacità di gestire scenari realistici e diversificati.

8.2 Test di integrazione

Dopo aver sviluppato il modello Simulink riguardo il caso d'uso *Detect & Avoid Obstacles*, si è ritenuto opportuno avere un primo approccio all'integrazione tra i due ambienti, seguendo il procedimento descritto nel capitolo riguardante il processo di integrazione e presente nelle fonti ufficiali del produttore [22]. Quindi a scopo di test è stato realizzato un modello comprendente un blocco rappresentante il sistema SODIA (fig.45) all'interno di un Block Definition Diagram (bdd) in Cameo Systems Modeler, incaricato dell'esecuzione del caso d'uso Detect and Avoid Obstacle. Il modello SODIA ha quindi assunto il ruolo di riferimento per la definizione delle interfacce, dei flussi informativi e delle variabili condivise tra i due ambienti.

Come illustrato in figura 45, al blocco rappresentante il sistema SODIA sono state associate diverse value property, corrispondenti alle variabili di input e output coinvolte nel modello Simulink Detect and Avoid Obstacle (azione fondamentale ai fini di simulazione, altrimenti quando si simula un determinato diagramma appartenente al blocco in questione, non ne riconosce le variabili come tali). A ciascuna di queste variabili è stato assegnato il tipo Real, in quanto si tratta di valori numerici continui, coerenti con la tipologia di dati utilizzata nel modello Simulink.

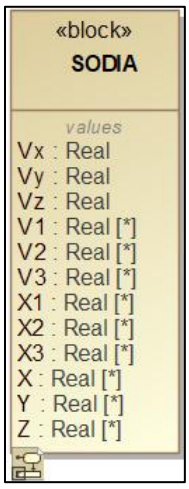

Fig.45 Blocco SODIA

Per quanto riguarda la molteplicità, alle variabili che rappresentano vettori è stata attribuita la molteplicità *, indicando una dimensione non specificata ma potenzialmente variabile. Alle variabili scalari, invece, non è stata assegnata esplicitamente alcuna molteplicità, sebbene sarebbe stato possibile indicare il valore 1; tale omissione non compromette il funzionamento del modello, risultando comunque compatibile con la semantica del sistema.

Inoltre, è possibile osservare che al blocco SODIA è stato associato un Activity Diagram, il quale risulta responsabile dell'esecuzione della simulazione; questo diagramma ha il compito di descrivere il flusso di controllo e le operazioni che il sistema deve compiere durante la simulazione del caso d'uso Detect and Avoid Obstacle. L'associazione tra il blocco e l'Activity Diagram consente di definire in modo esplicito il comportamento del sistema, facilitando l'interazione con il modello Simulink e permettendo l'attivazione delle operazioni necessarie per la simulazione continua.

Attraverso questa struttura, è stato possibile modellare la sequenza delle attività, l'elaborazione dei dati in ingresso e la generazione dei risultati in uscita, mantenendo una coerenza funzionale con quanto già implementato nel modello Simulink. L'Activity Diagram svolge quindi un ruolo centrale nell'integrazione, fungendo da ponte tra la rappresentazione architetturale in SysML e l'esecuzione simulata del sistema.

In figura 46 è rappresentata la struttura dell'Activity Diagram associato al blocco SODIA, il quale svolge un ruolo centrale nell'esecuzione della simulazione. Il diagramma è caratterizzato da una Call Behavior Action centrale, che rappresenta il modello Simulink precedentemente sviluppato. Tale elemento è stato inserito trascinando direttamente il file del modello Simulink all'interno dell'Activity Diagram, come descritto nel capitolo dedicato all'integrazione.

All'interno del diagramma sono state inoltre definite tutte le variabili di input, precedentemente modellate come value property del blocco SODIA, assegnando loro i valori iniziali utilizzati nella simulazione condotta in Simulink. Le variabili in questione includono:

- Vx, Vy, Vz (componenti della velocità del drone)
- V1, V2, V3 (velocità degli ostacoli)
- X1, X2, X3 (posizioni iniziali degli ostacoli)

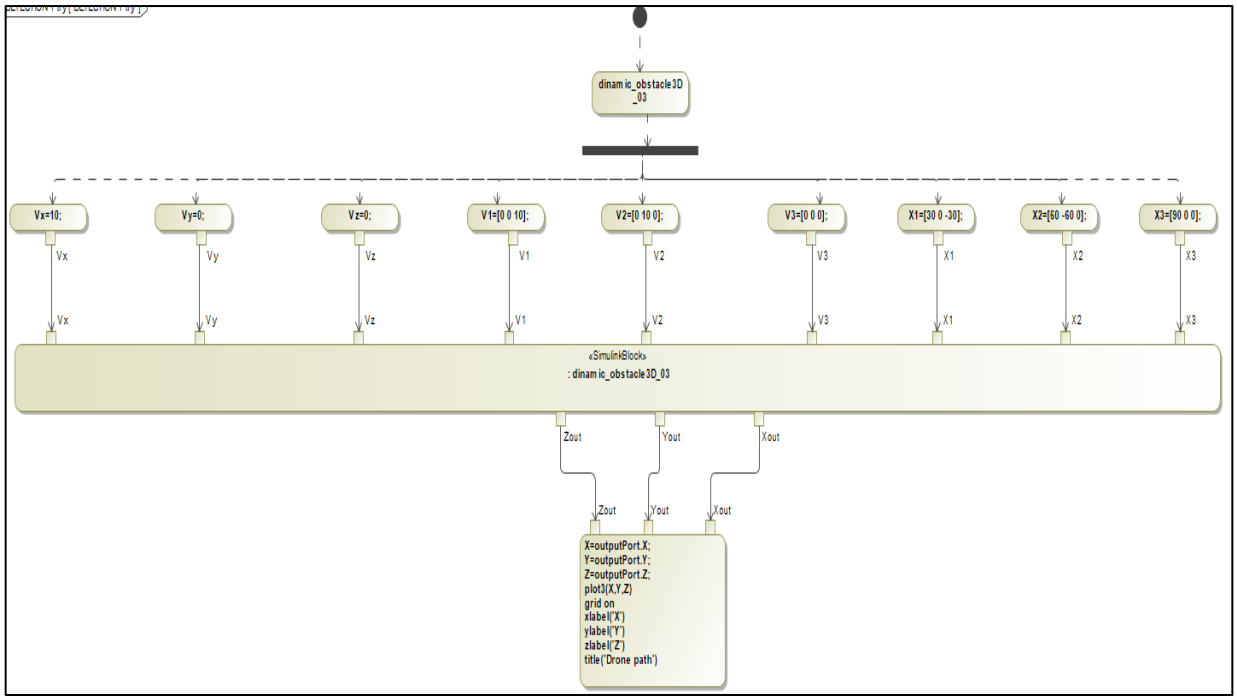

Fig.46 Activity Diagram

Queste variabili sono state implementate utilizzando il linguaggio MATLAB, sfruttando la possibilità offerta da Cameo di scrivere codice MATLAB all'interno di Opaque Action.

Per ciascuna variabile sono stati definiti i relativi pin, mantenendo la stessa nomenclatura, tipologia (Real) e molteplicità già assegnate alle value property, al fine di garantire la coerenza semantica e il corretto funzionamento del modello integrato.

Infine, l'ultima parte del diagramma contiene un'ulteriore Opaque Action, anch'essa scritta in linguaggio MATLAB, che ha il compito di generare un plot visivo direttamente all'interno di Cameo. Questo grafico consente di confrontare visivamente i risultati ottenuti dalla simulazione integrata con quelli prodotti dal modello Simulink originale.

Durante l'esecuzione della simulazione in Cameo, il flusso inizia con l'attivazione dell'Opaque Action iniziale, che richiama il modello Simulink, successivamente, il controllo passa attraverso un Fori Note (spiegato nei capitoli precedenti), che consente l'esecuzione parallela delle Opaque Action contenenti le variabili di input; una volta che tutte le variabili sono state inizializzate, il modello Simulink viene eseguito tramite la Call Behavior Action centrale (fig.46). Gli output

generati vengono infine trasmessi all'ultima Opaque Action, che elabora i dati e produce il grafico MATLAB, completando così il ciclo di simulazione.

Come evidenziato in figura 47, è stato effettuato un confronto tra il plot generato all'interno di Cameo Systems Modeler e quello prodotto direttamente in MATLAB/Simulink. I risultati mostrano una corrispondenza coerente tra i due grafici, confermando la corretta trasmissione delle variabili e l'efficacia dell'integrazione tra i due ambienti.

È importante sottolineare che questa simulazione ha avuto lo scopo principale di verificare la fattibilità tecnica dell'integrazione e di accertare che il flusso di dati tra SysML e Simulink potesse essere gestito correttamente. Si tratta quindi di una prima validazione preliminare, utile per confermare che il modello architetturale e quello funzionale siano in grado di comunicare e produrre risultati coerenti.

Nei capitoli successivi, si procederà con un approfondimento più dettagliato della simulazione, scendendo al livello dei singoli sottosistemi che compongono il sistema SODIA (come da scopo dichiarato inizialmente); l'obiettivo sarà quello di riprodurre, direttamente in Cameo, ogni passo della simulazione, analizzando in modo esplicito i flussi informativi tra i vari componenti interni. Questo tipo di analisi, finora gestita interamente all'interno del modello Simulink, verrà progressivamente trasferita e rappresentata in SysML, permettendo una visione più strutturata e modulare del comportamento del sistema.

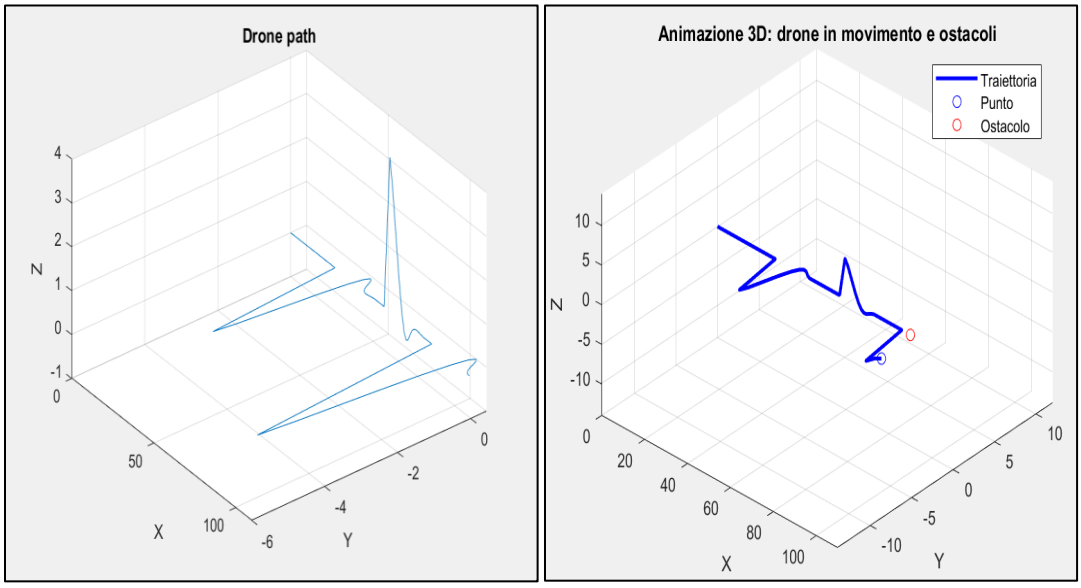

Fig.47 plot realizzato con la opaque action di Cameo System Modeler a confronto con il plot realizzato solo con Matlab/Simulink

8.3 Realizzazione dei modelli per l'integrazione

Dopo aver realizzato il modello iniziale in Simulink, compreso il funzionamento generale del sistema SODIA e verificata l'integrazione tra i due ambienti, si è scelto di approfondire ulteriormente l'integrazione attraverso una decomposizione funzionale del sistema nei suoi due sottosistemi principali: il Detection System e l'Avoidance System. Durante la modellazione della logica funzionale in Simulink, però è emerso che il comportamento del sistema può essere descritto attraverso tre funzioni distinte:

- Detection: individuazione di ostacoli entro un determinato raggio d'azione;
- Obstacle Parameter Estimation: valutazione delle caratteristiche dell'ostacolo rilevato, come velocità, direzione e rischio di collisione, al fine di determinare la strategia di evitamento più appropriata;
- Avoidance: modifica della traiettoria e della velocità del drone per aggirare l'ostacolo in sicurezza.

Sulla base di questa distinzione funzionale, si è deciso di completare il modello architetturale in Cameo (figura 48) introducendo un nuovo sottosistema dedicato alla stima dei parametri dell'ostacolo, denominato VECTRA, che si affianca ai due sottosistemi già esistenti. Il Block Definition Diagram risultante mostra il blocco centrale SODIA, che funge da nodo di interconnessione tra i tre sottosistemi, facilitando lo scambio di dati e la sincronizzazione delle operazioni.

Il primo blocco, Detect_Obstacle_system, rappresenta il sottosistema responsabile della rilevazione degli ostacoli. Al suo interno sono definite le value property che fungono da input per il corrispondente modello Simulink, tra cui:

X	posizione corrente del drone lungo X
Y	posizione corrente del drone lungo Y
Z	posizione corrente del drone lungo Z
X1	oordinate iniziali degli ostacolo 1
X2	oordinate iniziali degli ostacolo 2
X3	oordinate iniziali degli ostacolo 3
V1	Velocita ostacolo 1
V2	Velocita ostacolo 2
V3	Velocita ostacolo 3
t	tempo, che diventa di fondamentale importanza, volendo guidare la simulazione con Cameo e utilizzare Simulink per simulazioni one-shot, è necessario un riferimento in tempo per continuare il ciclo

	di simulazione e non simulare sempre la stessa iterazione (il tempo senza integrazione è gestito da Simulink);

Tab.1 value property di Detect_Obstacle_System

Il secondo blocco, VECTRA, è dedicato alla stima dei parametri dell'ostacolo e contiene anch'esso le value property necessarie per il corretto funzionamento del relativo modello Simulink. Come già discusso nel capitolo sull'integrazione, a ciascuna value property viene assegnato un nome, un tipo e una molteplicità coerente con la controparte Simulink, che verrà descritta nel dettaglio nel capitolo successivo.

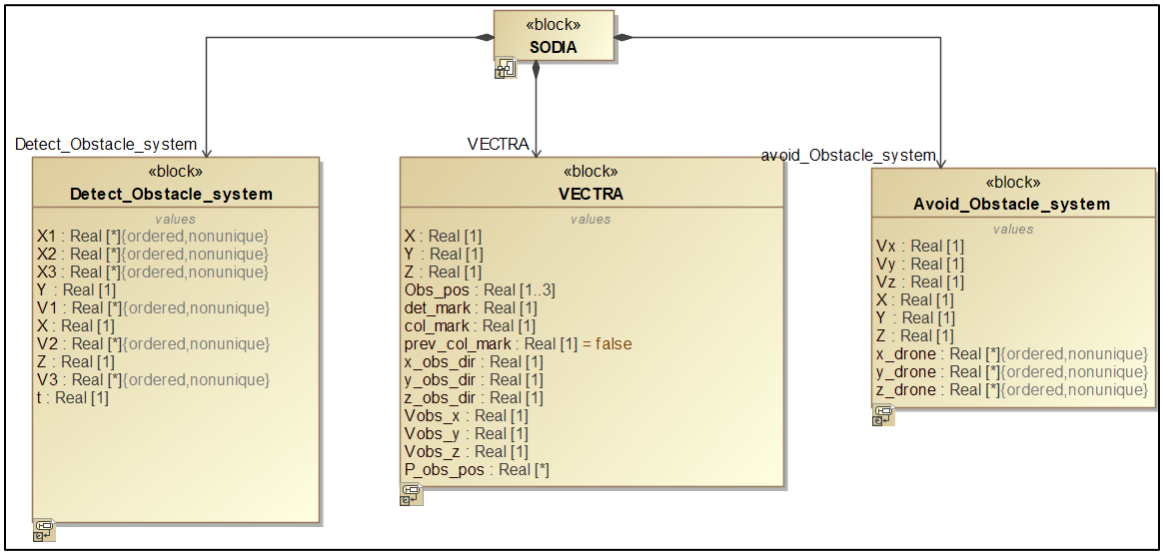

Fig.48 Body Block Diagram Detect and Avoid Obstacle System

Infine, il terzo blocco, Avoid_Obstacle_system, riceve i dati elaborati da VECTRA e calcola la nuova posizione del drone, completando il ciclo funzionale del sistema. Anche in questo caso, le value property sono definite in modo da garantire coerenza con il modello Simulink corrispondente.

Nel complesso, questo diagramma evidenzia chiaramente la logica funzionale distribuita del sistema, in cui ogni sottosistema svolge un ruolo specifico e contribuisce al comportamento globale del drone. Il blocco centrale SODIA garantisce una comunicazione fluida tra i moduli, permettendo una gestione coordinata delle informazioni e delle decisioni.

Infine, si anticipa che, coerentemente con questa decomposizione architetturale, anche il modello Simulink verrà adattato in tre modelli distinti, ciascuno corrispondente a uno dei sottosistemi definiti in Cameo. Questo adattamento, che verrà descritto nel capitolo successivo, permetterà di rafforzare l'allineamento tra la modellazione architetturale e quella simulativa, migliorando la modularità e la tracciabilità del sistema.

8.3.1 Adattamento modello Simulink

Come anticipato precedentemente, il passo successivo ha riguardato l'adattamento del modello Simulink in funzione dell'integrazione architetturale sviluppata in Cameo Systems Modeler. In particolare, si è scelto di decomporre il modello Simulink in tre sottosistemi distinti, corrispondenti alle principali funzioni operative del sistema SODIA: Detect Obstacle System, VECTRA (dedicato alla stima dei parametri dell'ostacolo), e Avoid Obstacle System. Questa suddivisione riflette la struttura funzionale già definita nel modello Cameo e consente una maggiore modularità e chiarezza nella gestione dei flussi informativi.

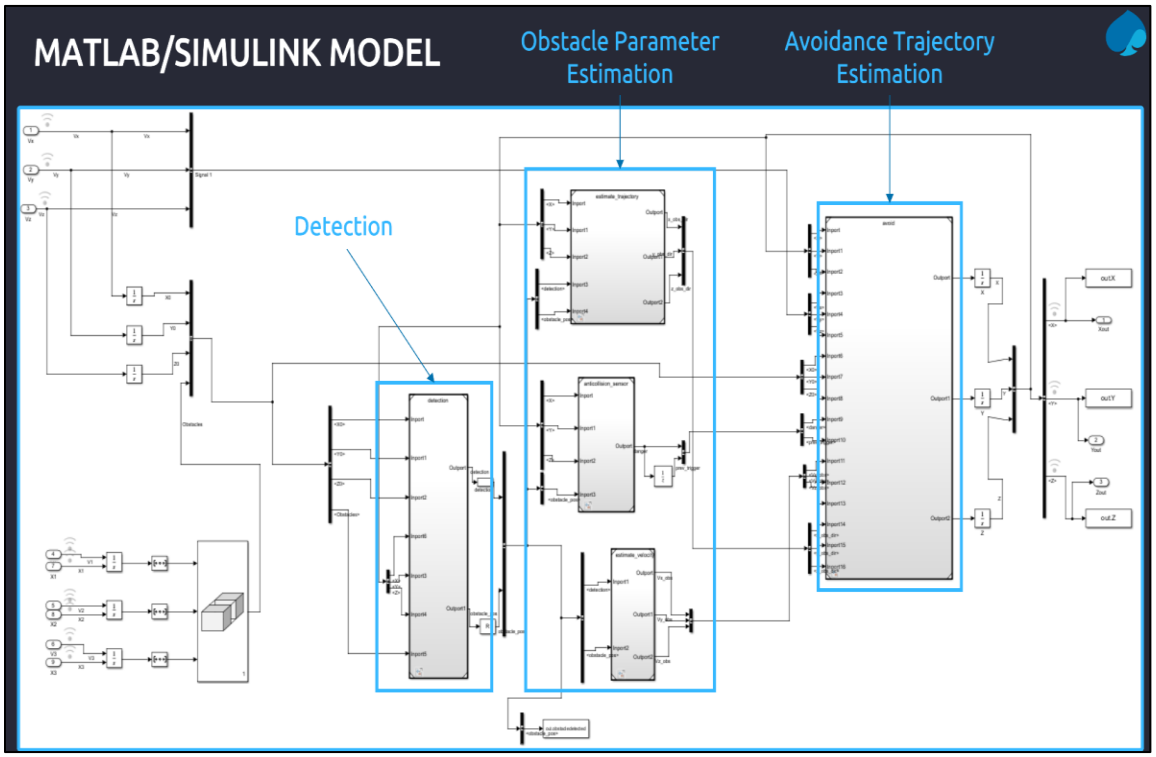

Fig.49 suddivisione del modello Simulink

La rappresentazione grafica di questa decomposizione è visibile nel modello Simulink mostrato in Figura 49, dove si distinguono chiaramente i tre moduli funzionali: il modulo di Detection, posizionato nella parte sinistra, il modulo di Obstacle Parameter Estimation al centro, e il modulo di Avoidance Trajectory Estimation sulla destra. Ciascun modulo contiene blocchi e connessioni che rappresentano le componenti e le interazioni necessarie per il corretto funzionamento del sistema, in coerenza con le value property definite nei rispettivi blocchi Cameo.

Le variabili utilizzate all'interno di questi moduli Simulink sono coerenti con le part property definite nei sottosistemi del modello SysML, garantendo una corrispondenza semantica tra i due ambienti e facilitando lo scambio di dati durante la simulazione.

In figura 50 è mostrato il primo di questi modelli, relativo al sottosistema di detection, che riprende il principio di funzionamento della prima parte del modello Simulink originale: riceve in input la posizione corrente del drone e la posizione degli ostacoli in un determinato istante di tempo e all'occorrenza nota i vari ostacoli. Poiché l'obiettivo è rappresentare lo scambio di informazioni tra i sottosistemi a ogni iterazione temporale, ogni modello Simulink viene eseguito come una funzione che riceve input e restituisce output in un singolo passo di simulazione.

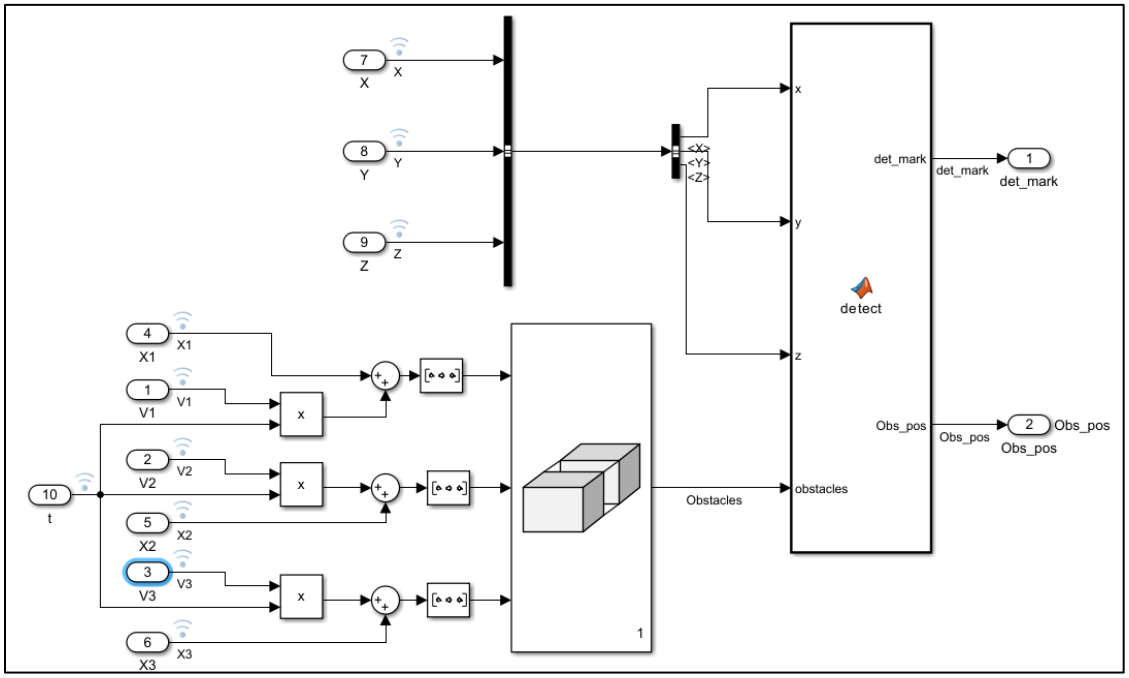

Figura 50 modello Simulink per il sottosistema detection

In questo nuovo approccio, Simulink non gestisce più direttamente il tempo, che viene invece controllato da Cameo; Il tempo viene fornito come input esplicito ai modelli Simulink, come nel caso del modulo di detection, dove è necessario per aggiornare dinamicamente la posizione degli ostacoli durante la simulazione; infatti, Il calcolo della posizione avviene secondo la formula del moto rettilineo uniforme:

$$s=s0 +v\cdot t$$

dove s è la posizione attuale dell'ostacolo, s0 la posizione iniziale, v la velocità dell'ostacolo e t il tempo fornito da Cameo riferito al passo di simulazione corrente.

Una volta calcolate le posizioni, queste vengono inviate come input a una MATLAB Function, che restituisce in output il segnale detection mark, il quale indica se un ostacolo è stato rilevato, e la posizione corrente dell'ostacolo.

All'interno della MATLAB Function sono state apportate due modifiche principali rispetto alla versione originale:

- Semplificazione degli input: non è più necessario fornire sia la posizione iniziale del drone che quella successiva a un eventuale cambio di traiettoria. L'inizializzazione e la gestione delle variabili avvengono ora direttamente in Cameo, mentre Simulink si limita a eseguire i calcoli.
- Gestione dell'output in assenza di ostacoli: poiché Cameo non supporta valori NaN come output, è stato deciso che, in caso di detection mark = 0 (cioè nessun ostacolo rilevato), la posizione dell'ostacolo venga rappresentata da un vettore fittizio [10000,10000,10000]. Questo valore è stato scelto arbitrariamente, ma sufficientemente elevato da non interferire con il normale range di rilevamento, evitando così ambiguità o errori nella simulazione.

Successivamente, è stato applicato lo stesso approccio per la realizzazione del modello relativo al secondo sottosistema, dedicato alla stima dei parametri utili per le manovre di evitamento, rappresentato in figura 51

La sua struttura e il suo funzionamento riprendono fedelmente quelli già presenti nella sezione del modello Simulink completo dedicata alla stima dei parametri, ma con alcune modifiche necessarie per adattarlo al nuovo contesto di simulazione integrata con Cameo.

Gli input principali del modello VECTRA sono:

- la posizione corrente del drone,
- il segnale detection_mark,
- la posizione dell'ostacolo (Obs_pos).

Questi dati non vengono più generati internamente, ma derivano direttamente dall'output del modello Simulink di Detection, eseguito in precedenza. Tuttavia, l'integrazione con Cameo ha introdotto una nuova esigenza: la gestione dello stato delle variabili nel tempo; in Simulink, questa funzione era affidata alle variabili persistent, che permettevano di memorizzare informazioni tra un passo di simulazione e l'altro, ora invece con il nuovo approccio, in cui ogni modello Simulink viene eseguito come funzione autonoma che compie una singola iterazione temporale, e dove la gestione delle variabili e dello scambio di dati è demandata a Cameo, l'uso delle variabili persistent non è più possibile.

Per ovviare a questa limitazione, è stata introdotta una nuova variabile denominata P_Obs_pos, che rappresenta la posizione dell'ostacolo al tempo precedente. La soluzione adottata consiste nel creare un output port P che trasmette direttamente il valore di Obs_pos, valore che viene poi salvato in Cameo e fornito come input nella simulazione successiva, assumendo il ruolo di P_Obs_pos (in figura 51 è possibile visualizzare queste porte).

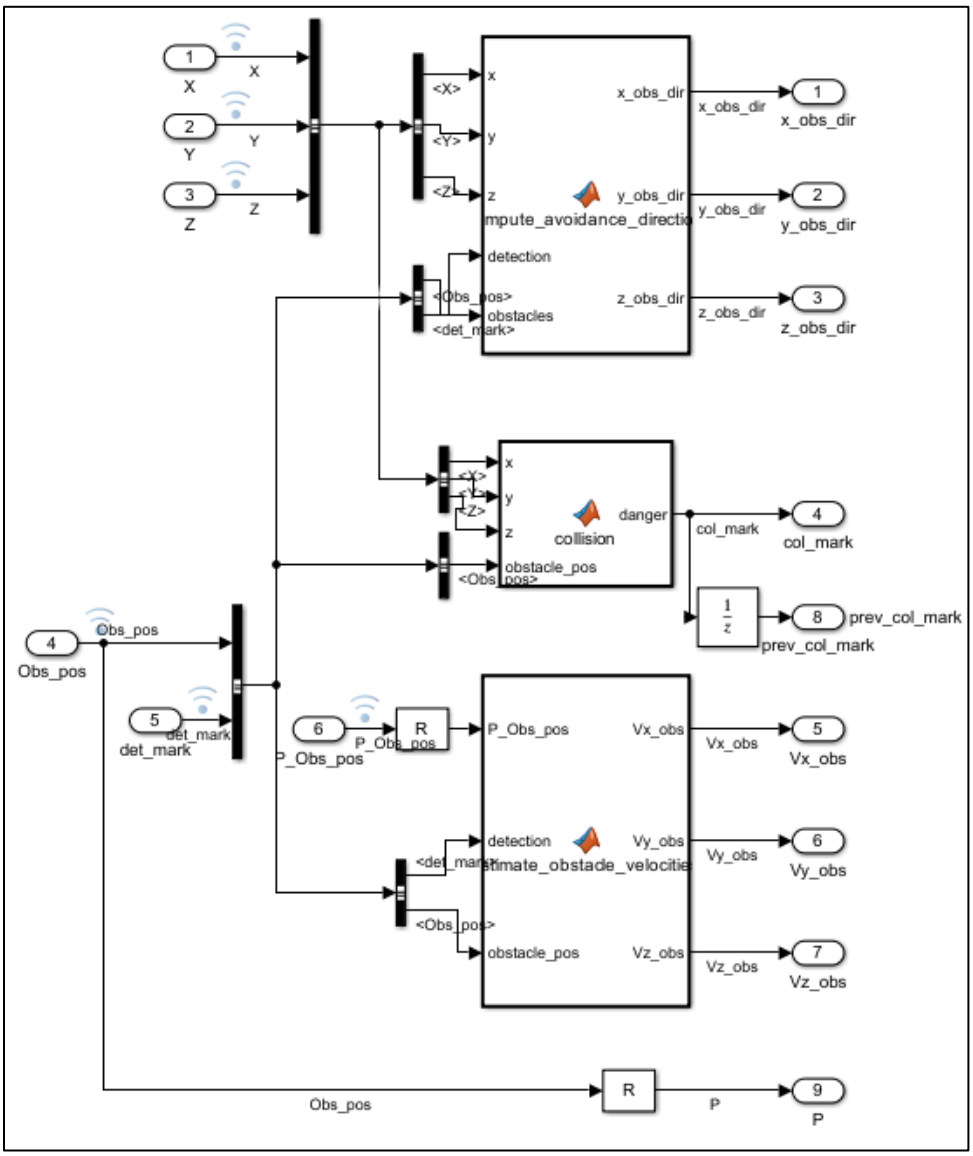

Fig.51 modello Simulink VECTRA

In questo modo, è possibile calcolare la variazione di posizione dell'ostacolo tra due istanti temporali consecutivi:

$$\Delta x = |Obs\_pos - P\_Obs\_pos|$$

e successivamente stimare la velocità dell'ostacolo dividendo per l'intervallo temporale:

$$v = \Delta x / \Delta t$$

Questa soluzione consente di mantenere la coerenza del calcolo senza ricorrere alle variabili persistent, garantendo al contempo la compatibilità con l'architettura modellata in Cameo.

Le altre due MATLAB Function presenti nel modello VECTRA sono rimaste invariate rispetto alla versione originale; gli output generati sono gli stessi e includono:

P	Posizione dell'ostacolo al tempo corrente
X_obs_dir	Direzione dell'ostacolo lungo l'asse
Y_obs_dir	Direzione dell'ostacolo lungo l'asse
Z_obs_dir	Direzione dell'ostacolo lungo l'asse
Vobs_x	Componenti delle velocità dell'ostacolo
Vobs_y	Componenti delle velocità dell'ostacolo
Vobs_z	Componenti delle velocità dell'ostacolo
Prev_col_mark	Trigger per rilevazione collisione
Col_mark	Trigger per rilevazione collisione

Tab.2 Output Vectra

Il terzo e ultimo modello sviluppato riguarda il sottosistema di Avoidance (figura 52), responsabile della generazione della nuova posizione del drone a ogni passo di simulazione. Questo modulo rappresenta la fase conclusiva del processo di navigazione autonoma, in cui il drone, sulla base delle informazioni ricevute dai moduli precedenti, modifica la propria traiettoria per evitare ostacoli rilevati.

Anche in questo caso, la variabile temporale t, gestita da Cameo, viene fornita come input al modello Simulink e viene utilizzata in combinazione con le componenti di velocità del drone per calcolare la traiettoria desiderata; inoltre, viene passata direttamente alla MATLAB Function che implementa la logica di evitamento, permettendo di generare posizioni in uscita che risultano funzione del tempo.

Durante la fase di simulazione, è emersa la necessità di semplificare il codice della MATLAB Function, perchè il modello richiedeva l'utilizzo di variabili persistent e un numero elevato di punti di simulazione per ottenere un comportamento realistico, tuttavia, questo approccio risultava inefficiente in termini di tempo di calcolo. Considerando il caso d'uso specifico, si è deciso di ridurre la complessità del codice, concentrandosi sull'obiettivo principale: modificare la posizione del drone per evitare gli ostacoli, mantenendo un comportamento reattivo e compatibile con l'architettura modellata in Cameo.

La logica implementata nella funzione MATLAB prevede tre comportamenti principali:

- Inizio dell'evitamento: quando viene rilevata una collisione imminente, il sistema entra in modalità di evitamento; in base alla velocità dell'ostacolo e alla sua direzione di movimento, viene scelta una strategia di deviazione.
  - Se l'ostacolo è statico, il drone devia lungo l'asse y.
  - Se si muove prevalentemente in verticale, la deviazione avviene ancora lungo y.
  - Se invece si muove orizzontalmente, la deviazione avviene lungo z.

In tutti i casi, viene applicata una distanza di fuga fissa (4 m scelti arbitrariamente), che garantisce un margine di sicurezza.

- Ritorno alla traiettoria originale: una volta che la situazione di pericolo è rientrata, il drone torna alla traiettoria pianificata, riprendendo il percorso originario.
- Movimento normale: in assenza di ostacoli, il drone prosegue lungo la traiettoria definita dalle sue velocità iniziali e dal tempo corrente.

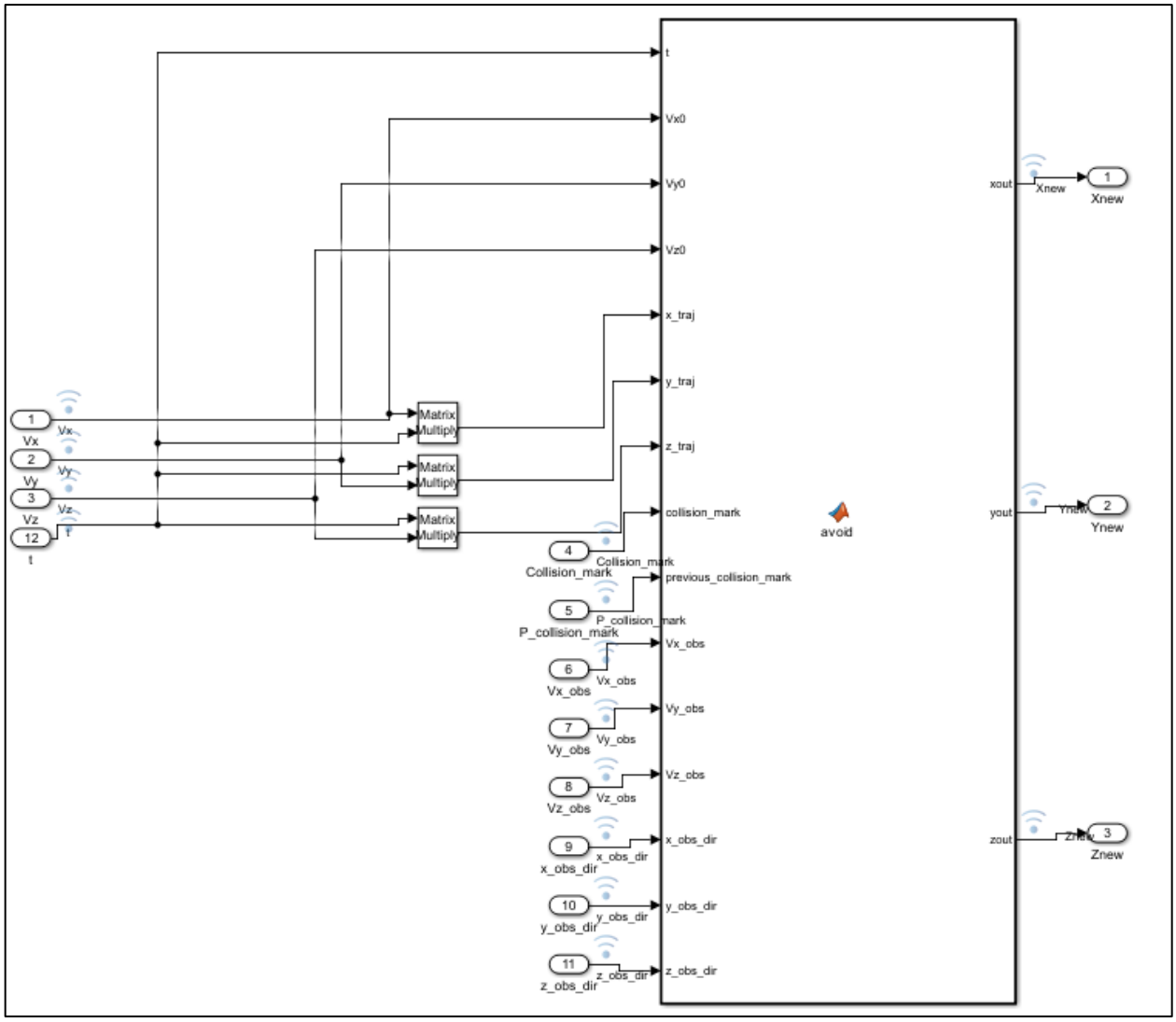

Fig.52 modello Cameo di Avoidance

Questa implementazione consente di simulare un comportamento realistico e reattivo del drone, mantenendo al contempo una struttura semplice e compatibile con l'architettura modellata in Cameo. Il sistema è in grado di adattarsi dinamicamente alle condizioni dell'ambiente, garantendo una navigazione sicura e autonoma.

8.3.2 Integrazione

Per rappresentare in modo completo e strutturato lo scambio di informazioni tra i sottosistemi, si è fatto uso congiunto di Block Definition Diagram (fig.48) e Internal Block Diagram; in particolare, l'ibd del sistema SODIA (figura 53) mostra come Detect_Obstacle_system, VECTRA e Avoid_Obstacle_system siano modellati come part property di SODIA, evidenziando le connessioni tra i blocchi e la natura dei segnali scambiati.

Come si nota in figura 53, alle part property sono stati associati sono stati definiti diversi pin di comunicazione che permettono lo scambio strutturato di informazioni tra i sottosistemi.

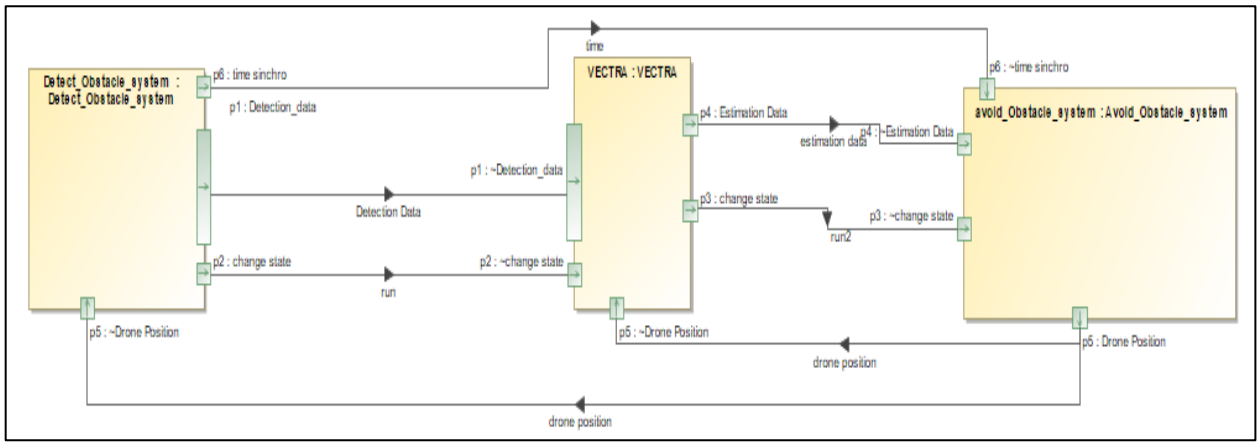

Fig.53 Internal block diagram del Sistema

Entrando nel dettaglio, i pin utilizzati sono:

- Detection_data: questo pin consente il trasferimento delle variabili calcolate durante la fase di rilevamento degli ostacoli, in particolare il detection mark e la posizione degli ostacoli (Obs_pos), dal sottosistema Detect_Obstacle_system al sottosistema VECTRA. Tale scambio è essenziale per permettere a VECTRA di stimare i parametri dinamici dell'ostacolo, necessari per la successiva fase di evitamento.
- Drone Position: rappresenta il pin attraverso cui viene inviata la nuova posizione del drone ai sottosistemi Detect_Obstacle_system e VECTRA. Questi sottosistemi, una volta completata la propria funzione all'interno di un ciclo di simulazione, restano in attesa della nuova posizione per riavviare il ciclo successivo. Questo meccanismo garantisce la realizzazione di un loop dinamico continuo, fondamentale per la simulazione iterativa del comportamento del drone.
- change_state: pin utilizzato per trasmettere un segnale di attivazione (run, run2) che consente l'ingresso nei diversi stati operativi del sistema. Questi stati, che verranno descritti nei successivi diagrammi di macchina a stati, regolano la sequenza e la logica di

esecuzione della simulazione, assicurando che ogni sottosistema operi nel momento corretto.

- time_sinchro: questo pin ha il compito di distribuire la variabile temporale (t) dal sottosistema Detect_Obstacle_system agli altri moduli che ne necessitano per il proprio funzionamento. La sincronizzazione temporale è cruciale per mantenere la coerenza tra le operazioni dei sottosistemi e per garantire una simulazione realistica e ordinata.
- Estimation Data: pin che trasferisce i segnali di output generati da VECTRA al sottosistema Avoid_Obstacle_system. Questi segnali includono le stime relative alla posizione, direzione e velocità dell'ostacolo, e costituiscono l'input fondamentale per il calcolo della nuova traiettoria del drone.

Questi pin non sono definiti arbitrariamente, ma sono tipizzati attraverso Interface Blocks, la cui struttura è descritta nel Block Definition Diagram (fig. 54). Tali blocchi definiscono le flow properties associate ai segnali, ciascuna delle quali rappresenta un'informazione specifica trasmessa tra sottosistemi. La loro definizione all'interno delle interfacce garantisce coerenza semantica e strutturale nel passaggio attraverso i pin, permettendo una modellazione precisa dello scambio informativo tra moduli e supportando l'integrazione tra Cameo Systems Modeler e Simulink, nonché la simulazione del comportamento dinamico del drone.

Il bdd organizza in modo gerarchico e modulare i Signal Blocks, che rappresentano le informazioni scambiate tra i sottosistemi. Ogni segnale è modellato con attributi specifici che ne descrivono le caratteristiche fondamentali:

- Il nome della variabile, coerente con quello utilizzato nei diagrammi di attività (che verranno presentati nei capitoli successivi) e nei modelli Simulink, in modo da garantire una corrispondenza diretta tra la modellazione concettuale e la simulazione esecutiva;
- Il tipo di dato associato alla variabile (ad esempio Real, Boolean, ecc.), necessario per la corretta interpretazione e gestione del segnale nei diversi ambienti di modellazione;
- La molteplicità, che specifica il numero di istanze del segnale e ne definisce la struttura (singolo valore, vettore, intervallo, ecc.).

Questa formalizzazione consente di stabilire un riferimento univoco per ciascun dato scambiato durante la co-simulazione, facilitando l'integrazione tra Cameo Systems Modeler e Simulink e assicurando la coerenza tra i diversi livelli di astrazione del sistema.

Vediamo ora nel dettaglio i segnali utilizzati e i loro attributi:

Segnale	Segnale incluso	attributo	ruolo
Detection Data	Detection Mark	det_mark Real[1]	indicatore di rilevamento ostacolo
	Obstacle Position	Real[1]{ordered, nonunique}	posizione notata dell'ostacolo, posto det_mark=1 (diversamente passerà un vettore [10000 10000 10000] di default);

time	time	Real[1]	tempo
Obstacle Data	Obstacle trajectory x	Real[1]	Traiettoria dell'ostacolo lungo gli assi
	Obstacle trajectory y	Real[1]	Traiettoria dell'ostacolo lungo gli assi
	Obstacle trajectory z	Real[1]	Traiettoria dell'ostacolo lungo gli assi
	Obstacle velocity x	Real[1]	velocità dell'ostacolo lungo gli assi
	Obstacle velocity y	Real[1]	velocità dell'ostacolo lungo gli assi
	Obstacle velocity z	Real[1]	velocità dell'ostacolo lungo gli assi
Collision Signals	p_mark	Real[1]	segnale booleano di collisione utilizzato per riferimento temporale allo stato precedente
	mark	Real[1]	segnale booleano di collisione
Drone Position	X	Real[1]	Posizione del drone lungo l'asse
	Y	Real[1]	Posizione del drone lungo l'asse
	Z	Real[1]	Posizione del drone lungo l'asse

Tab.3 Segnali

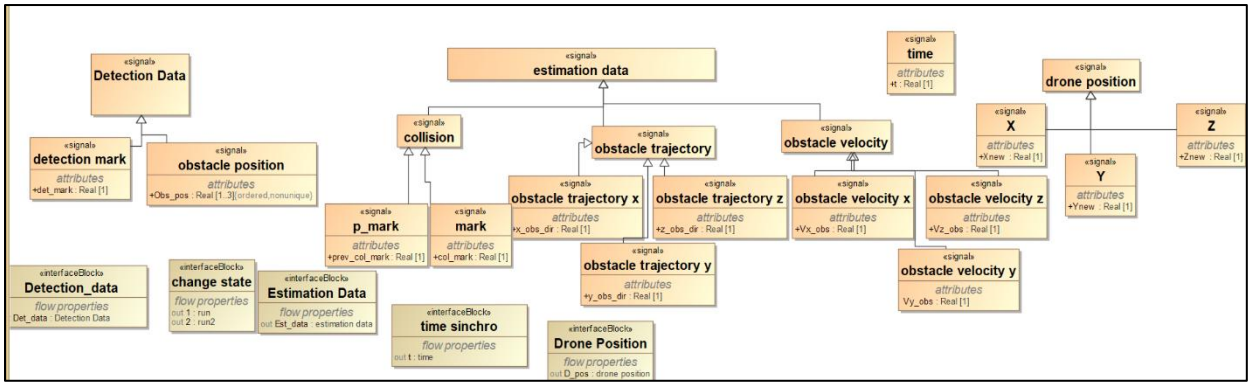

Fig.54 Body Block Diagram of Exchanged Items

La combinazione tra IBD e BDD consente di visualizzare chiaramente la logica di interazione tra i sottosistemi, evidenziando modularità, scalabilità e tracciabilità del sistema SODIA. La centralità del blocco SODIA garantisce una gestione coordinata delle informazioni, mentre la definizione esplicita dei segnali e delle interfacce facilita verifica, validazione ed estensibilità del modello.

Questa struttura si rivela particolarmente efficace nell'ambito dell'approccio Model-Based Systems Engineering (MBSE), che permette di gestire la complessità dei sistemi cyber-fisici attraverso modelli formali e integrati.

Per rendere possibile l'integrazione e la co-simulazione tra Cameo Systems Modeler e MATLAB Simulink, è stato necessario modellare il comportamento dinamico di ciascun sottosistema del sistema SODIA attraverso la creazione di diagrammi di macchina a stati e diagrammi di attività. Questi diagrammi permettono di descrivere in modo formale e modulare la logica operativa dei sottosistemi, facilitando la sincronizzazione tra i modelli e la corretta esecuzione della simulazione.

In figura 55 sono riportati i diagrammi di macchina a stati dei tre sottosistemi principali:

- Stm Detect_Obstacle_system: il diagramma è composto da due stati principali, initialization e obstacle_detecting. La transizione dallo stato iniziale a quello attivo avviene tramite l'evento start, mentre il ritorno allo stato di inizializzazione è attivato dall'evento stop. All'ingresso nello stato obstacle_detecting viene eseguito l'activity diagram (fig.56) che comanda l'invio del segnale run, in modo da permettere a VECTRA di entrare nello stato operativo, e in seguito viene eseguita l'azione detection, contenente il diagramma di attività Obstacle_detection.
- Stm VECTRA: il diagramma presenta due stati, idle e obstacle data elaboration. La transizione verso lo stato attivo avviene tramite il segnale run, mentre il ritorno allo stato di inattività è gestito dall'evento stand-by. All'ingresso nello stato di elaborazione viene eseguito l'activity diagram che gestisce il segnale run2 (fig.57) (permette ad Avoid_Obstacle_system di entrare nello stato operativo), dopo di che eseguita l'azione estimation, contenente il diagramma di attività Obstacle_Trajectory_Estimation.
- Stm Avoid_Obstacle_system: il diagramma è composto dagli stati Initialization e control avoidance. La transizione verso lo stato attivo avviene tramite la ricezione del segnale run2, ricevuto dal sottosistema VECTRA. Durante lo stato control avoidance, viene eseguito l'azione Avoidance Elaboration, che calcola la nuova traiettoria del drone per evitare l'ostacolo.

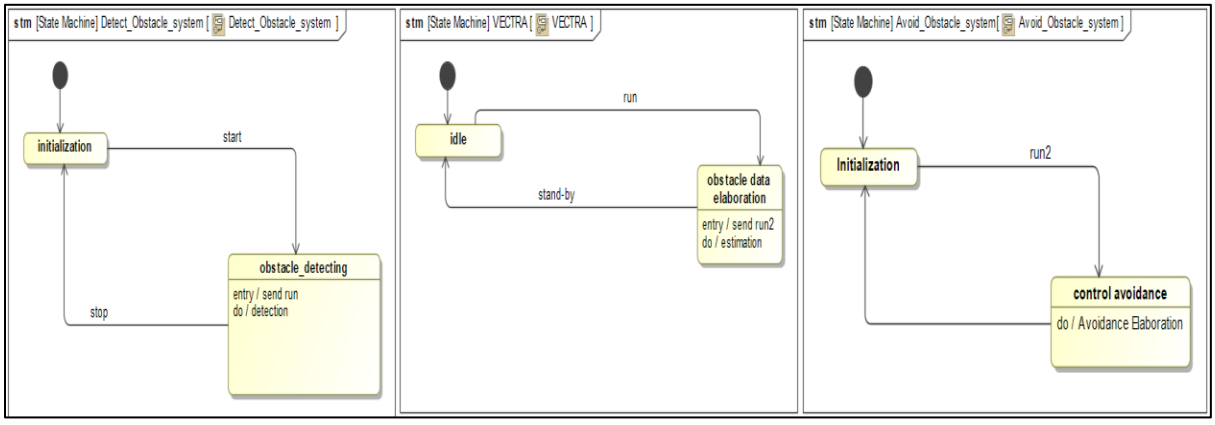

Fig.55 state machine diagrams del modello Cameo

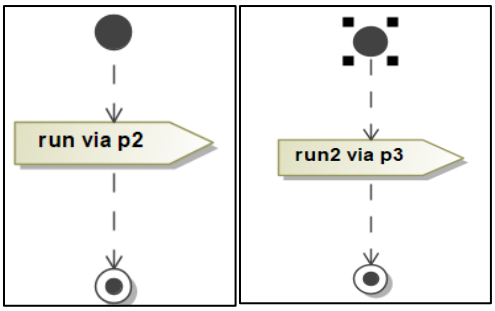

Fig.56 activity diagram run Fig.57 activity diagram run 2

Questi diagrammi permettono di modellare il comportamento sequenziale e reattivo dei sottosistemi, definendo chiaramente gli eventi che regolano le transizioni tra stati e le azioni eseguite in ciascuna fase. La presenza di segnali come run e run2 garantisce la corretta sincronizzazione tra i moduli, mentre l'uso di azioni specifiche (detection, estimation, Avoidance Elaboration) consente di associare ogni stato a una funzione operativa ben definita.

Questa modellazione è essenziale per assicurare una coerenza funzionale tra la rappresentazione concettuale in Cameo e l'esecuzione simulata in Simulink, contribuendo alla validazione del comportamento del sistema SODIA in scenari dinamici e complessi.

Infine, per completare la modellazione SODIA, sono stati realizzati i diagrammi di attività associati agli stati operativi dei sottosistemi. Questi diagrammi descrivono nel dettaglio la sequenza di operazioni che ciascun modulo esegue durante la simulazione, gestendo il flusso delle informazioni, l'interazione con i modelli Simulink e la comunicazione tra i sottosistemi.

In figura 58 è rappresentato il diagramma di attività relativo allo stato operativo di Detect_Obstacle_sistem. Il diagramma si apre richiamando tramite un opaque action (in linguaggio Matlab) il modello Simulink detection_03 (il numero si riferisce alla versione più aggiornata del modello, la stessa presentata in precedenza), dopo di che vi è un nodo fork che si occupa di inizializzare in parallelo tutte le variabili (definite all'interno di opaque action in linguaggio Matlab) necessarie per il funzionamento del modello Simulink, quindi velocità e posizione iniziale degli ostacoli, tempo e posizione iniziale del drone.

Si possono notare in questa parte superiore del diagramma diversi nodi merge, se posti prima della opaque action indicano che quel valore della variabile è costante in tutti i cicli di simulazione, mentre se i nodi merge sono posti sotto la opaque action vuol dire che la variabile viene inizializzata durante il primo ciclo e verrà poi aggiornata durante i cicli successivi quindi ponendo il nodo merge al di sotto si evita di sovrascrivere il valore iniziale. Una volta inizializzate le variabili,vengono mandate all'interno dei pin del simulink block che sfrutta il Matlab Engine per eseguire la simulazione e restituisce tramite i pin di output i risultati ottenuti dal modello Simulink integrato.

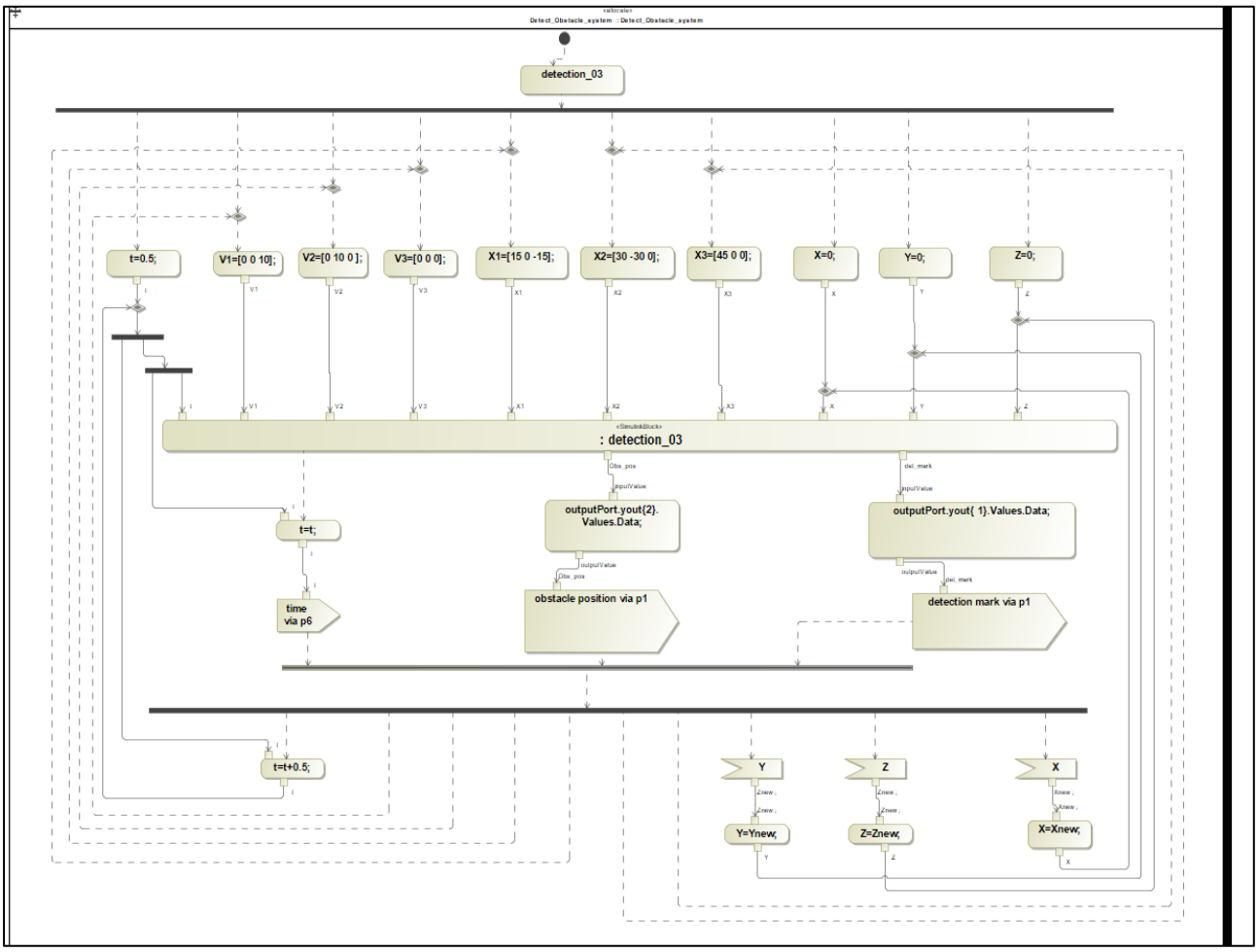

Fig.58 diagramma di attività Obstacle_detection, contenuto nell'azione detection

Gli output vengono richiamati all'interno delle due opaque action successive attraverso la dicitura "outputPort.yout{}.Values.Data", in modo da richiamare la variabile completa direttamente dalla console comune ai due ambienti; questo procedimento è stato necessario soprattutto per la variabile Obs_pos, variabile vettoriale, che in fase di simulazione veniva considerata scalare senza questo passaggio. La comunicazione con gli altri sottosistemi avviene attraverso l'azione send_signal_action, che invia i segnali di output ai moduli destinatari, utilizzando i pin e le interfacce precedentemente definiti (nota, si ricorda che tutti i pin presenti nei diagrammi di attività sono tipizzati in modo congruente rispetto alle Value Property definite nel Body Block Diagram del sistema SODIA, trattandosi delle stesse variabili).

Al termine del processo, il sistema entra in uno stato di attesa di nuovi dati, pronto per riavviare il ciclo operativo non appena disponibili nuove informazioni, garantendo così la continuità e la dinamicità della simulazione. Inoltre, queto diagrammi di attività una volta inviato le relative variabili a VECTRA ed Avoid_Obstacle_system, si occupa anche di aggiornare la variabile tempo in attesa delle nuove posizioni del drone per poter incominciare il nuovo ciclo di simulazione.

In figura 59 è rapresentato il diagramma di attività associato allo stato operativo del sottosistema VECTRA, che si occupa della stima dei parametri dell'ostacolo rilevato. Questo diagramma, come nel caso del sottosistema Detect_Obstacle_system, è strutturato per gestire il flusso delle informazioni in ingresso, elaborarle tramite il modello Simulink integrato e restituire i risultati agli altri moduli del sistema SODIA.

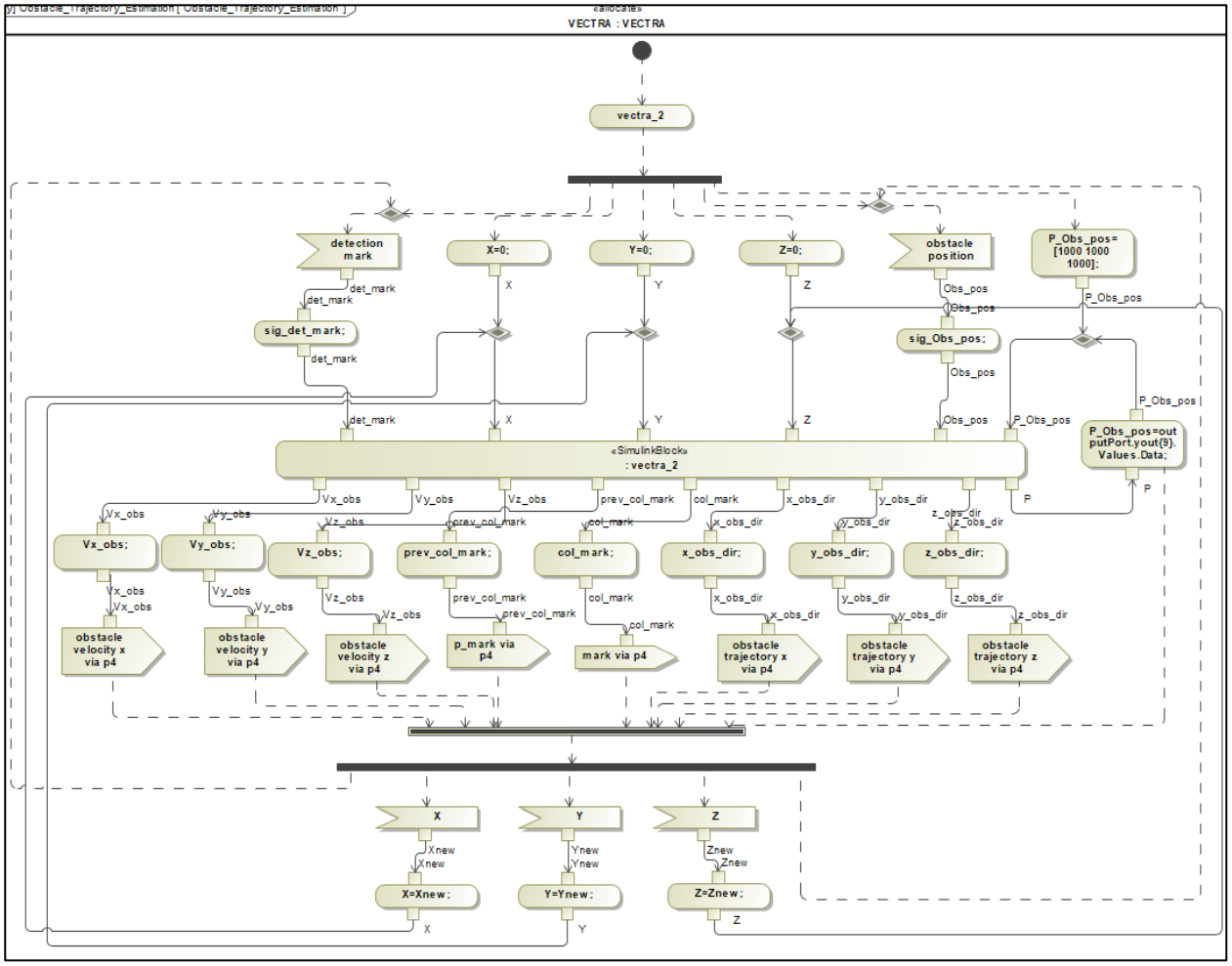

Fig.59 Activity Diagram VECTRA

Il diagramma si apre con una serie di azioni di input, che inizializzano le variabili necessarie per il primo ciclo di simulazione; Tra queste si trovano la posizione iniziale del drone e la variabile P_Obs_pos necessaria per il calcolo della velocità dell'ostacolo (già spiegata in precedenza), che vengono definite all'interno di opaque action in linguaggio MATLAB. Inoltre, si possono notare anche i due accept event action in attesa delle variabili detection mark e obstacle position ricavate e inviate dal diagramma precedente (si possono notare opaque action con la dicitura "sig_..." che forzano la lettura del contenuto del segnale relativo in modo da garantire la corretta continuazione

della simulazione, in alternativa potrebbe esserci anche il passaggio diretto dall'accept event action al simulink block anche se con determinati tipi di variabili, come i vettori, potrebbe causare problemi).

Una volta inizializzate le variabili, il flusso prosegue verso l'elaborazione del modello Simulink, che viene richiamato tramite una call behaviour action specifica (vectra_2). Come nel diagramma precedente, anche qui si osservano nodi merge e fork, che gestiscono la logica di parallelismo e aggiornamento delle variabili: i merge posti sopra le opaque action indicano variabili costanti, mentre quelli posti sotto permettono l'aggiornamento ciclico dei dati simulati.

Gli output del modello Simulink, come le direzioni e le velocità dell'ostacolo lungo i tre assi, in questo caso vengono semplicemente richiamate con il loro nominativo all'interno della opaque action, essendo tutti valori scalari (tranne P_Obs_pos che viene richiamato nello stesso modo di prima) non vi è il bisogno di forzare la lettura da console (si nota che il nominativo delle variabili è lo stesso delle porte di output nei modelli simulink).

Infine, i risultati elaborati vengono trasmessi agli altri sottosistemi tramite l'azione send_signal_action, sfruttando i pin e le interfacce tipizzate definite nell'Internal Block Diagram. Anche in questo caso, il diagramma si conclude con una fase di attesa di nuovi dati, durante la quale il sistema resta in stand-by fino alla ricezione della nuova posizione del drone, necessaria per avviare un nuovo ciclo di simulazione.

Questo diagramma evidenzia la modularità e la coerenza operativa del sottosistema VECTRA, che riceve i dati elaborati dal modulo di rilevamento, li processa tramite Simulink e li trasmette al modulo di evitamento, contribuendo alla gestione intelligente del comportamento del drone.

In figura 60 è rappresentato il diagramma di attività relativo allo stato operativo del sottosistema Avoid_Obstacle_system, che ha il compito di elaborare la traiettoria di evitamento del drone sulla base dei dati ricevuti da VECTRA. Questo diagramma conclude il ciclo operativo del sistema SODIA, modellando la fase in cui il drone reagisce alle informazioni elaborate e calcola una nuova posizione sicura.

Il diagramma si apre con un nodo iniziale che attiva il processo di controllo, seguito da una serie di accept event action e opaque action che gestiscono l'elaborazione dei dati ricevuti. In alto a destra si puo notare una opaque action in cui vengono inizializzati i vettori vuoti delle posizioni lungo i tre assi, in modo da poter essere riempiti coi valori risultati da ogni ciclo e visualizzare man mano i risultati (si nota che questa opaque action viene eseguita unicamente durante il primo ciclo di simulazione).

Una volta ricevute, le variabili necessarie vengono lette all'interno delle opaque action con dicitura "sig_.." e inviate all'interno del Simulink Block in modo da poter eseguire la co-simulazione del relativo modello Simulink e poter ottenere la nuova posizione corrente del drone, che verrà salvata all'interno dei vettori delle posizioni precedentemente inizializzati e in seguito visualizzata tramite un plot visivo, reso possibile grazie alla opaque action con script Matlab presente nel centro dell'immagine.

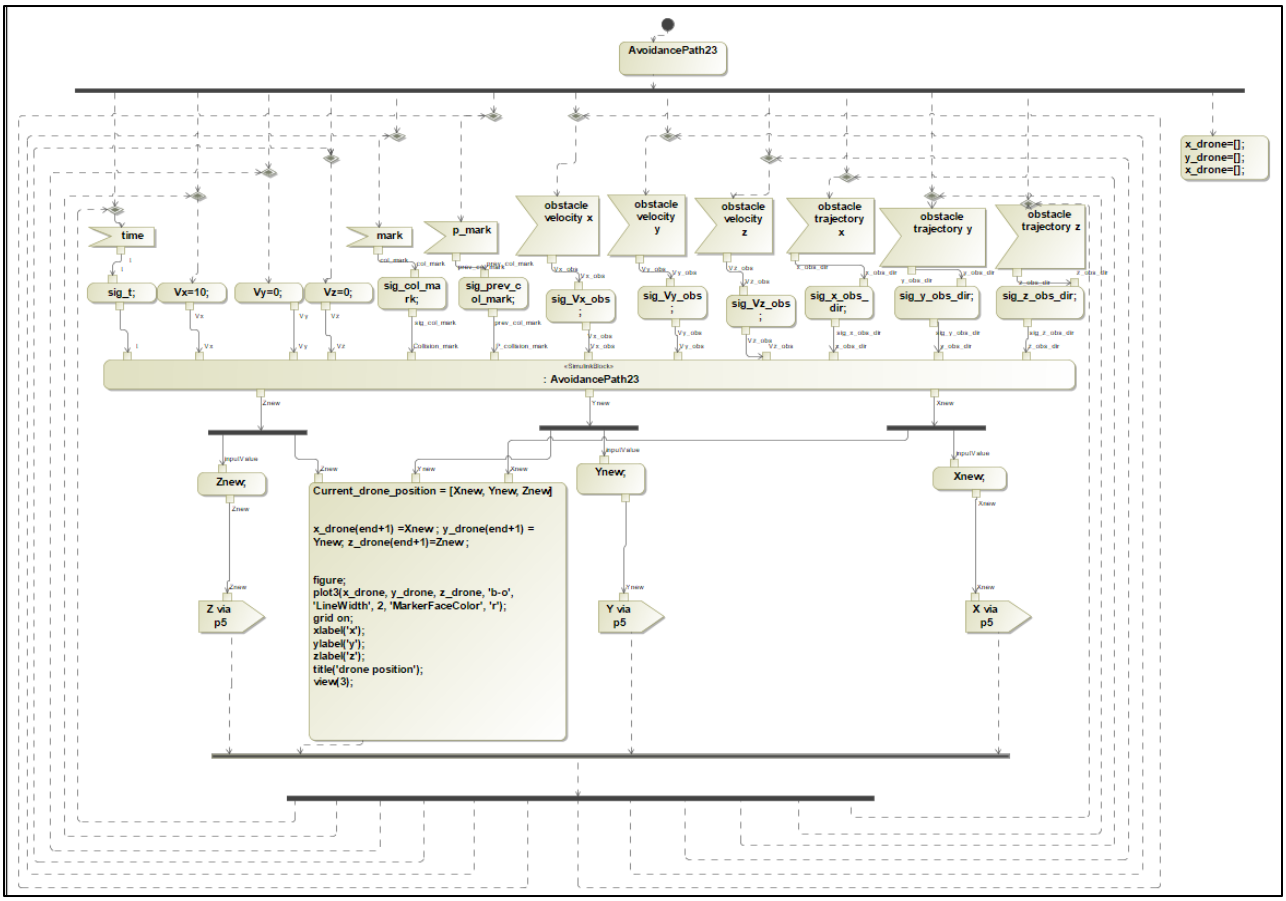

Fig.60 diagramma di attività Avoidance_Elaboration

Al termine dell'elaborazione, il diagramma di attività del sottosistema Avoid_Obstacle_system prevede l'invio dei risultati tramite l'azione send_signal_action, che comunica la nuova posizione del drone ai sottosistemi Detect_Obstacle_system e VECTRA. Questo passaggio è fondamentale per consentire l'avvio del ciclo di simulazione successivo, poiché, come evidenziato nel diagramma di attività associato a Detect_Obstacle_system, il sottosistema resta in attesa proprio di questi dati per poter iniziare una nuova iterazione.

Questo meccanismo chiude il loop dinamico del sistema SODIA, garantendo una simulazione continua, reattiva e coordinata tra i sottosistemi. Ogni modulo opera in modo sequenziale ma interdipendente: i dati elaborati in una fase diventano input per la successiva, e la nuova posizione del drone aggiornata rappresenta il punto di partenza per un nuovo ciclo. Tale struttura consente di modellare il comportamento del sistema in modo iterativo e adattivo, simulando in maniera realistica l'interazione del drone con l'ambiente circostante e la sua capacità di reagire dinamicamente agli ostacoli rilevati.

8.3.3 Simulazione e risultati

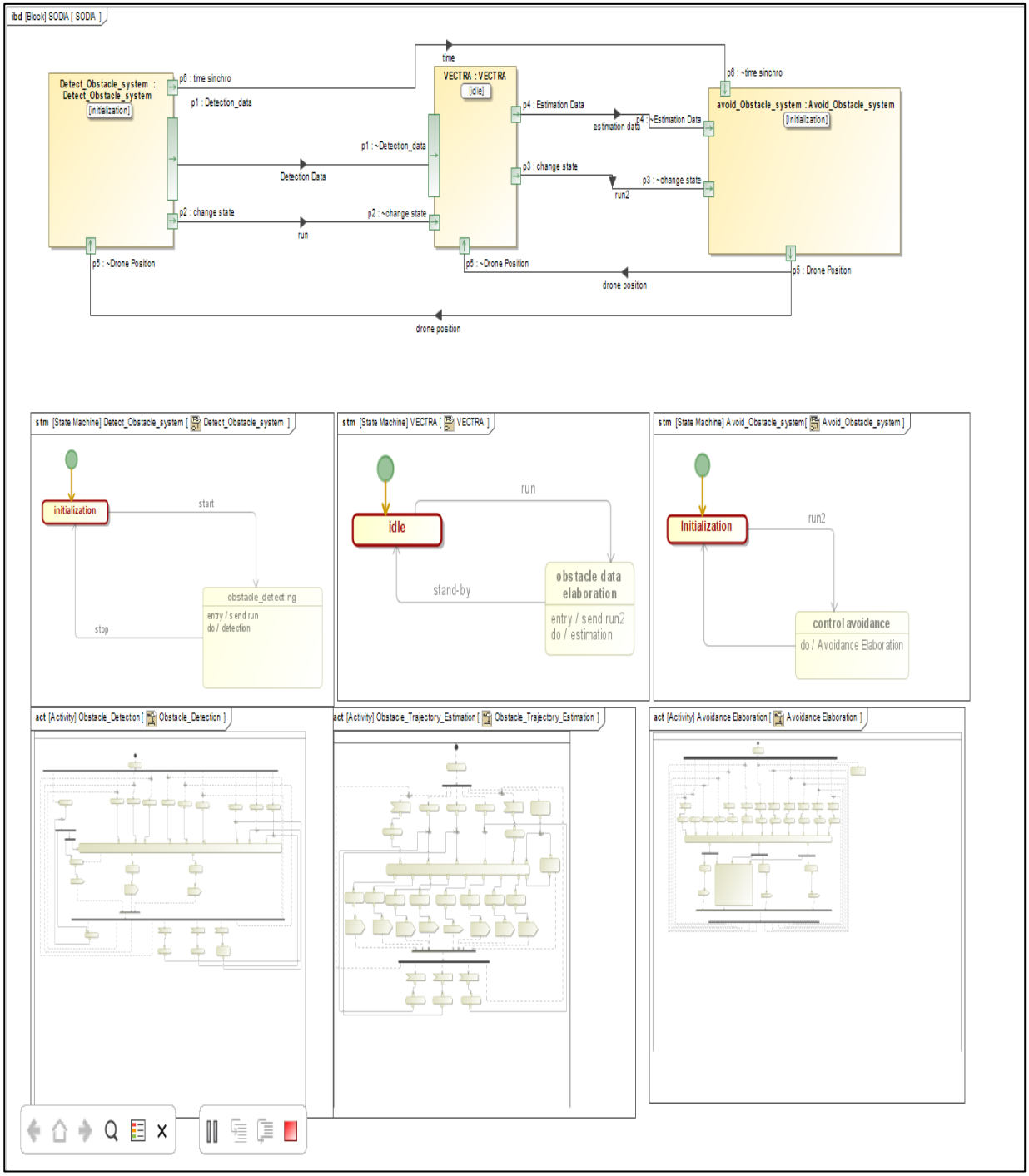

Fig.61 IBD con tutti i diagrammi che entrano in gioco in simulazione e i tre sottosistemi in stato di inizializzazione

Dopo aver completato la modellazione strutturale e comportamentale del sistema SODIA, è stata avviata la fase di simulazione, con l'obiettivo di verificare il corretto funzionamento dei

sottosistemi e la coerenza dell'interazione tra i modelli Cameo e Simulink. In questa sezione vengono presentati i risultati ottenuti e le modalità con cui è stata condotta la co-simulazione.

Per avviare la simulazione del sistema, si utilizza l'Internal Block Diagram, all'interno del quale vengono trascinati i diagrammi di macchina a stati e i diagrammi di attività relativi ai sottosistemi, in modo da visualizzare e gestire tutte le informazioni operative su un'unica interfaccia(fig.61). Una volta predisposta la struttura, si avvia la simulazione dell'Internal Block Diagram: inizialmente, i tre sottosistemi entrano nello stato di inizializzazione, in attesa del segnale di attivazione start, che deve essere fornito manualmente dall'utente(fig.61).

Dopo l'attivazione del trigger, il sottosistema Detect_Obstacle_system transita nello stato operativo obstacle_detecting. In questa fase, viene immediatamente inviato il segnale run, che consente al sottosistema VECTRA di passare allo stato attivo. Successivamente, viene eseguita l'azione detection, che attiva il relativo diagramma di attività (Obstacle_detection), responsabile dell'interazione con il modello Simulink e dell'elaborazione dei dati rilevati.

Il sottosistema VECTRA segue una logica analoga: una volta ricevuto il segnale run, invia a sua volta il segnale run2, che permette al modulo Avoid_Obstacle_system di entrare nello stato operativo; a quel punto, viene eseguita l'azione estimation, che attiva il diagramma di attività Obstacle_Trajectory_estimation, incaricato di stimare la traiettoria dell'ostacolo e di preparare i dati per la fase di evitamento. Questa sequenza di attivazioni e transizioni tra stati operativi, gestita attraverso segnali e diagrammi di attività, consente di simulare in modo coordinato e iterativo il comportamento del sistema SODIA.

Una volta eseguite le azioni associate ai diagrammi di attività, è possibile osservare come questi vengano attivati in successione ordinata, a partire dal modulo Obstacle_detection. Come descritto nel sottocapitolo precedente, questo primo diagramma si occupa di inizializzare le variabili necessarie, richiamare il modello Simulink dedicato alla rilevazione degli ostacoli, eseguire i calcoli, memorizzare gli output e inviarli al sottosistema successivo; al termine, il modulo resta in attesa di nuovi dati per poter avviare un nuovo ciclo.

Lo stesso schema operativo si ripete per tutti i diagrammi di attività dei sottosistemi successivi, contribuendo alla costruzione di un loop dinamico che regola l'intera simulazione; ogni sottosistema riceve i dati elaborati dal precedente, li processa tramite il proprio modello Simulink, e trasmette i risultati al modulo successivo, in un flusso continuo e coordinato; questo ciclo iterativo prosegue fino a quando la simulazione non viene interrotta manualmente dall'utente.

Un aspetto fondamentale della simulazione del sistema SODIA è la possibilità di visualizzare e tracciare in modo esplicito lo scambio di informazioni tra i sottosistemi (uno degli obiettivi desiderati per questo lavoro) nell'Internal Block Diagram (fig.62), ma anche le variabili all'interno dei diagrammi di attività (fig.63). Le connessioni tra i pin dei vari moduli permettono di osservare il flusso dei dati che viaggiano da un sottosistema all'altro, seguendo la logica definita nei diagrammi comportamentali.

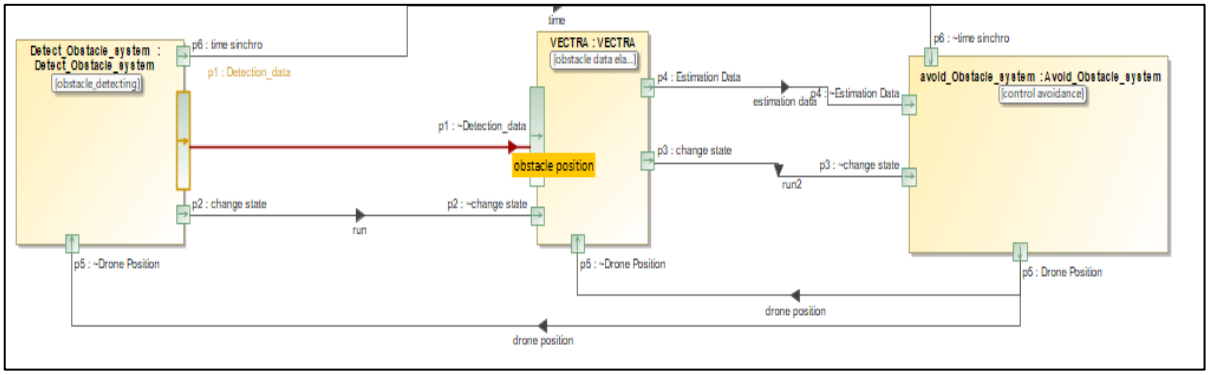

Fig. 62 esempio di scambio di informazioni tra sottosistemi

I diagrammi di attività permettono lo scambio di dati tra sottosistemi grazie all'uso combinato di send_signal_action e accept_event_action, infatti quando un sottosistema termina la propria elaborazione, invia i dati tramite un send_signal_action, sfruttando i pin tipizzati definiti nell'Internal Block Diagram; per garantire che il segnale venga correttamente ricevuto, è essenziale che il sottosistema destinatario abbia già attivato la propria accept_event_action, cioè sia in attesa del segnale. In caso contrario, il messaggio verrebbe perso, compromettendo la sincronizzazione tra i moduli e interrompendo il flusso operativo fig.64).

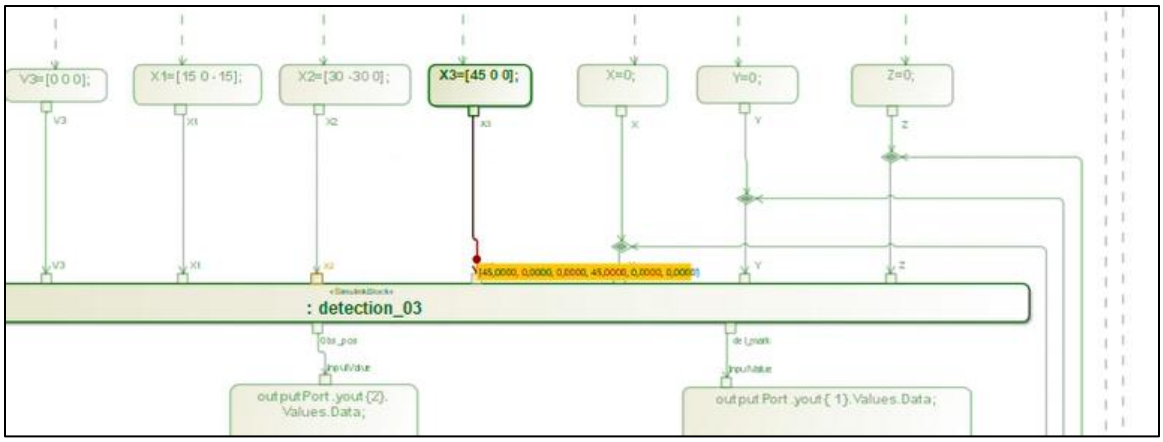

Fig.63 esempio di variabile che va dalla opaque action di inizializzazione al Simulink Block, activity diagram detect obstacle

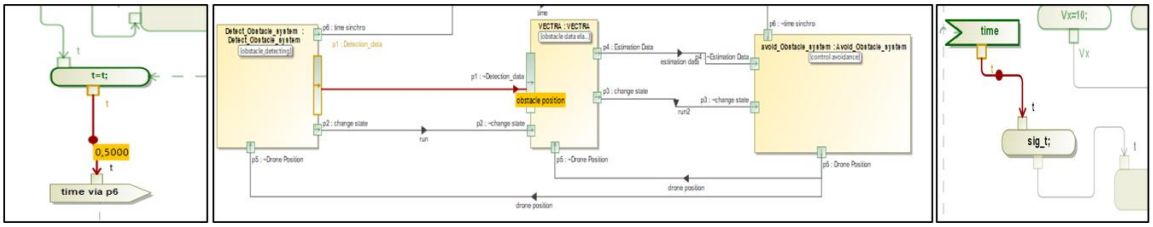

Fig.64 esempio di funzionamento send signal action / accept event action

Questo meccanismo di attivazione condizionale assicura che tutti i diagrammi di attività vengano eseguiti in sequenza, rispettando l'ordine logico del sistema e contribuendo alla realizzazione di un loop dinamico. Ogni sottosistema riceve i dati elaborati dal precedente, li processa, e trasmette i risultati al successivo, in un ciclo continuo che si ripete fino a quando la simulazione non viene interrotta manualmente.

Al termine di ogni ciclo iterativo, grazie all'integrazione tra Cameo Systems Modeler e MATLAB Simulink, viene generato automaticamente un Simulink plot che rappresenta il percorso aggiornato del drone; questo output grafico consente di monitorare visivamente l'evoluzione del comportamento del sistema, offrendo un riscontro immediato sull'efficacia delle strategie di rilevamento, stima ed evitamento adottate.

In figura 65 si possono osservare i plot dei risultati; più nello specifico a sinistra vi è il risultato della prima iterazione, a destra il risultato della seconda iterazione mentre in figura 66 vi è il plot finale.

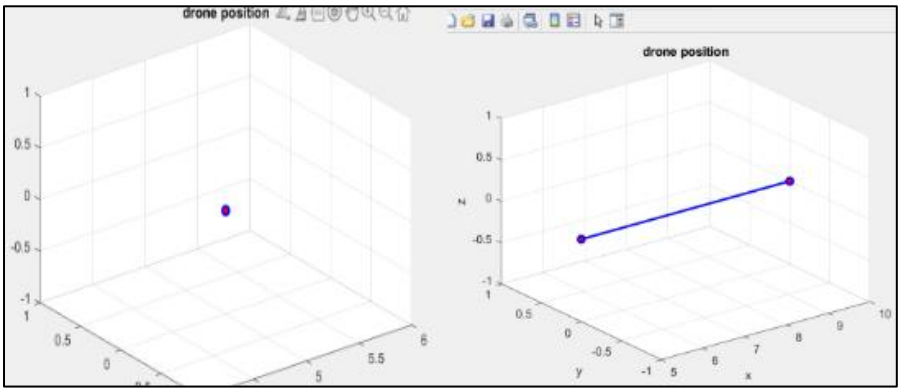

Fig.65 plot che compaiono a ogni iterazione

Come si può osservare dalle immagini presentate, le linee che rappresentano il percorso del drone risultano segmentate. Questa caratteristica non è casuale, ma è frutto di una scelta progettuale consapevole adottata durante la realizzazione del modello integrato. A differenza di un modello sviluppato interamente in Simulink, che è ottimizzato per eseguire simulazioni rapide anche su un elevato numero di punti, il modello integrato tra Cameo Systems Modeler e MATLAB Simulink è strutturalmente più complesso e quindi notevolmente più lento nell'esecuzione.

Questo rallentamento è dovuto al fatto che si è scelto di modellare il sistema a livello di sottosistema, rappresentando in modo esplicito tutti i flussi di informazioni tra i moduli, attraverso diagrammi di attività, diagrammi di macchina a stati e pin tipizzati nell'Internal Block Diagram; tale approccio, sebbene metodologicamente più rigoroso e aderente ai principi dell'MBSE, comporta un notevole carico computazionale durante la simulazione.

In termini pratici, il modello integrato impiega circa 2–3 minuti per completare un singolo ciclo iterativo e fino a 30 minuti per generare un Simulink plot con il percorso del drone su 13 posizioni (fig.66); questo rende impraticabile l'utilizzo del modello completo per simulazioni estese o in tempo reale, come inizialmente previsto. Per rendere la simulazione più gestibile e compatibile con i tempi di calcolo, si è quindi proceduto con una serie di semplificazioni, descritte nel capitolo dedicato al modello Simulink di Avoidance (8.3.1).

Queste semplificazioni hanno permesso di eseguire il caso d'uso "Detect and Avoid Obstacle", mantenendo la struttura logica del sistema e la coerenza tra i sottosistemi; tuttavia, il prezzo da pagare è stata una perdita di fluidità nei movimenti del drone: il comportamento simulato, che idealmente dovrebbe essere continuo, si traduce in una sequenza discreta di posizioni, con transizioni visibili e segmentate. Questo effetto è chiaramente osservabile nei grafici generati, dove il percorso del drone appare frammentato, riflettendo la natura iterativa e non continua della simulazione.

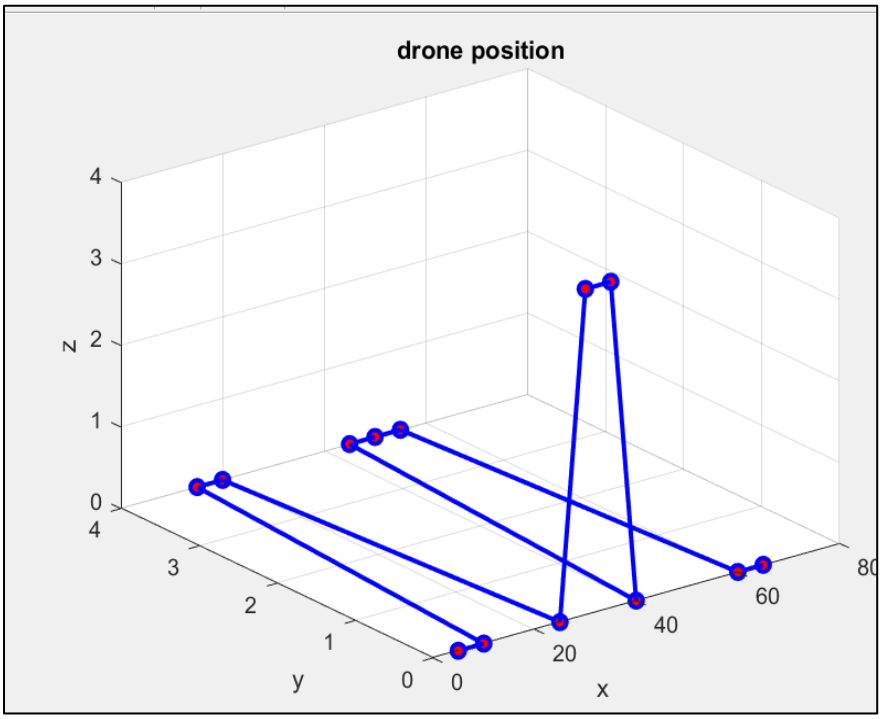

Fig.66 plot completo percorso drone

Dal punto di vista metodologico, il modello mantiene un elevato valore dimostrativo, poiché consente di analizzare nel dettaglio il funzionamento interno dei sottosistemi, lo scambio di segnali e la logica di attivazione tra moduli. Tuttavia, sul piano simulativo, l'approccio adottato (basato sull'integrazione e simulazione dei singoli sottosistemi) non è adatto per scenari realistici, a causa dei tempi di esecuzione troppo elevati. Nonostante ciò, questa soluzione si dimostra efficace per finalità dimostrative, in quanto permette di mettere in evidenza le potenzialità dell'integrazione tra i modelli e di mostrare concretamente le capacità di interoperabilità tra gli strumenti impiegati

CAPITOLO 9 CONCLUSIONE

Il presente lavoro di tesi ha avuto come obiettivo la modellazione dell'architettura e la co-simulazione comportamentale di un sistema per il rilevamento e l'evitamento degli ostacoli ("*Detect & Avoid Obstacle*"), attraverso l'integrazione tra Cameo Systems Modeler e MATLAB Simulink. L'approccio adottato ha permesso di esplorare le potenzialità e i limiti di una co-simulazione tra ambienti di modellazione architetturale(funzionale/logica) e simulazione dinamica.

A partire dall'analisi del caso d'uso, sono stati sviluppati diversi modelli in grado di soddisfare i requisiti funzionali del sistema. In particolare, è stato realizzato un modello completo in Simulink che simula l'intero comportamento del sistema, e poi realizzati tre modelli dedicati ai sottosistemi di SODIA (Sistema di *Detection VECTRA* e *Avoidance*), capaci di riprodurre in modo verosimile le dinamiche dei componenti reali. L'architettura logica e funzionale è stata invece modellata in Cameo, con l'obiettivo di rappresentare il funzionamento del sistema, i flussi informativi e le interazioni tra i moduli.

Tra i risultati più significativi si evidenzia la realizzazione di un loop dinamico all'interno del modello Cameo, una sfida metodologica che ha richiesto una gestione avanzata delle variabili e delle opaque action. È emersa inoltre una notevole compatibilità tra le variabili dei due ambienti, facilitata dalla possibilità di utilizzare il linguaggio MATLAB direttamente in Cameo; questo ha consentito, tra l'altro, di accedere al contenuto del workspace condiviso e di visualizzare i dati di simulazione tramite la funzione whos (utile per capire come Cameo memorizzasse le variabili), nonché di generare grafici direttamente all'interno del modello architetturale.

L'integrazione tra Cameo e Simulink si è rivelata tecnicamente semplice, come illustrato nei capitoli dedicati, tuttavia, sono emerse alcune criticità che meritano attenzione; ad esempio, la mancata tipizzazione di un pin all'interno di un'activity diagram può compromettere l'intero processo di simulazione. Inoltre, per sostenere il carico computazionale richiesto dalla simulazione, è stato necessario allocare fino a due terzi della memoria RAM disponibile a Cameo, pena l'insorgenza di errori casuali e difficilmente diagnosticabili, data la scarsa chiarezza dei messaggi di errore forniti dalla console di Cameo rispetto a quella di MATLAB.

Un ulteriore limite riguarda l'impossibilità di utilizzare variabili persistent in Simulink, dovuta alla scelta metodologica di eseguire un solo passo di simulazione per ciclo, al fine di mantenere la coerenza tra i sottosistemi e rappresentare correttamente il flusso informativo ad ogni passo temporale; questo ha comportato la perdita della capacità di memorizzare lo stato interno del sistema tra un ciclo e l'altro.

Dal punto di vista prestazionale, si è osservato che i modelli Simulink, per come erano stati concepiti, prevedevano l'elaborazione di un elevato numero di punti; tuttavia, l'integrazione con Cameo ha reso impraticabile questa soluzione: un singolo ciclo di simulazione richiede circa due minuti, mentre la generazione del grafico finale può impiegare fino a trenta minuti, nonostante il

numero limitato di punti (13 posizioni, a titolo di confronto, il modello realizzato interamente in MATLAB è in grado di produrre migliaia di punti in pochi secondi). Per rendere la simulazione compatibile con Cameo, è stato necessario semplificare i modelli, ridurre il numero di punti e modificare alcune logiche interne, con conseguente perdita di continuità nel movimento simulato del drone; il comportamento, idealmente continuo, si traduce quindi in una sequenza discreta di posizioni, con transizioni visibili e segmentate, come evidenziato nei grafici generati (fig.67).

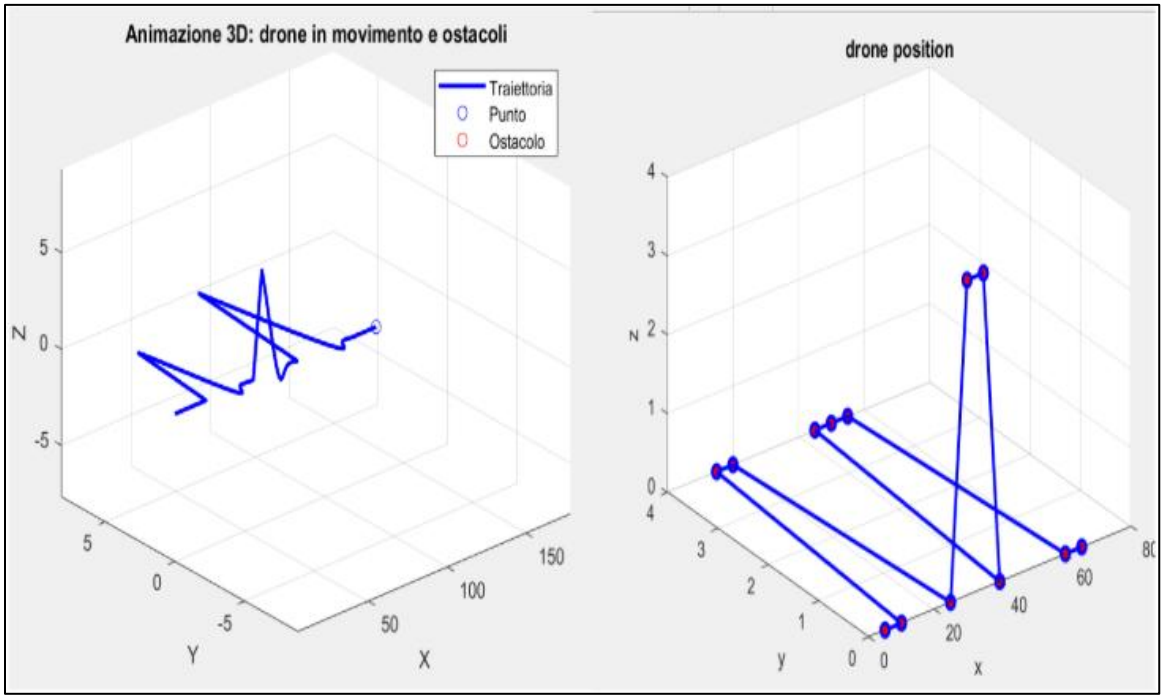

Fig.67 confronto tra i risultati ottenuti con SImulink a sinistra e con l'integrazione a destra

Nonostante tali limitazioni, il modello mantiene un elevato valore dimostrativo, consentendo di analizzare nel dettaglio il funzionamento interno dei sottosistemi, lo scambio di segnali e la logica di attivazione. L'approccio adottato, pur non adatto a simulazioni realistiche per via dei tempi di esecuzione elevati, si è rivelato efficace per finalità dimostrative, evidenziando le potenzialità dell'integrazione tra modelli e la capacità di interoperabilità tra gli strumenti impiegati.

Nella seguente tabella vengono riassunti i vantaggi e svantaggi descritti precedentemente.

Aspetti	Vantaggi	Limiti / Criticità
Interoperabilità	Elevata compatibilità tra variabili nei due ambienti	Mancata tipizzazione di un pin può compromettere la simulazione

Accesso ai dati	Possibilità di usare MATLAB direttamente in Cameo	Messaggi di errore poco chiari nella console Cameo rispetto a MATLAB
Visualizzazione	Generazione di grafici direttamente nel modello architetturale	Tempi di generazione molto lunghi (fino a 30 minuti per un grafico)
Simulazione	Co-simulazione tra architettura e dinamica del sistema	Un solo passo per ciclo impedisce uso di variabili persistent
Gestione delle risorse	Integrazione tecnicamente semplice	Elevato consumo di RAM (fino a 2/3 della memoria disponibile)
Modellazione architetturale	Rappresentazione dettagliata di flussi informativi e logiche di attivazione	Perdita di continuità nel movimento simulato del drone
Valore dimostrativo	Analisi approfondita dei sottosistemi e del loro comportamento	Non adatto a simulazioni realistiche per via dei tempi di esecuzione elevati
Scalabilità futura	Possibilità di richiamare dinamicamente modelli Simulink da Cameo	Necessità di semplificare modelli per compatibilità con Cameo

Tab.4 Risultati

Alla luce delle esperienze maturate, si ritiene opportuno, per sviluppi futuri, concentrare la simulazione all'interno di MATLAB Simulink, sfruttando appieno le sue potenzialità computazionali e mantenendo Cameo come strumento di modellazione architetturale a livello di sistema; in particolare, si suggerisce di rappresentare l'intero drone in Cameo, assegnando a ciascun sistema (es. SODIA, sistema di illuminazione, sistema di emergenza) un modello Simulink dedicato. In questo modo, si potrebbero definire casi studio condizionati, in cui, in base allo stato del sistema o a specifici eventi, Cameo richiama dinamicamente il modello Simulink più appropriato e ne visualizza i risultati.

Questa strategia permetterebbe di preservare la coerenza architetturale e la rappresentatività del sistema, senza compromettere le prestazioni simulative, offrendo una soluzione scalabile e più aderente alle esigenze di progettazione e verifica in ambito ingegneristico.

References

1. INCOSE. (n.d.). Systems Engineering Handbook: A guide for system life cycle processes and activities (5th ed.). Wiley.
2. ISO/IEC/IEEE. (2015). Systems and software engineering — System life cycle processes (ISO/IEC/IEEE 15288:2015).
3. Pyster, A., et al. (2019). Systems Engineering Body of Knowledge (SEBoK) (Version 2.2).
4. Capgemini Engineering. (n.d.). Systems Engineering Academy for Aerospace, Defense & Rail: Systems Engineering Introduction [Day 1].
5. INCOSE. (2015). A Complexity Primer for Systems Engineers.
6. INCOSE. (2019). Systems Engineering and System Definitions.
7. Camelia, F., & Ferris, T. L. J. (2016). Systems thinking in systems engineering. INCOSE International Symposium.
8. Capgemini Engineering. (n.d.). Systems Engineering Academy for Aerospace, Defense & Rail: Systems Engineering Introduction [Day 2].
9. ISO/IEC/IEEE. (2018). Systems and software engineering — Life cycle processes — Requirements engineering (ISO/IEC/IEEE 29148:2018).
10. INCOSE Requirements Working Group. (2023). Guide to writing requirements (Version 4, INCOSE-TP-2010-006-04).
11. Capgemini Engineering. (n.d.). Systems Engineering Academy for Aerospace, Defense & Rail: Systems Engineering Introduction [Day 3].
12. INCOSE Fellows. (2019).
13. Capgemini Engineering. (n.d.). MBSE Introduction.
14. INCOSE. (2007). Systems Engineering Vision 2020 (INCOSE-TP-2004-004-02).
15. Friedenthal, S., Moore, A., & Steiner, R. (2014). A practical guide to SysML: The systems modeling language (3rd ed.). Morgan Kaufmann.
16. Estefan, J. A. (2008). Survey of model-based systems engineering (MBSE) methodologies. INCOSE MBSE Initiative.
17. Dassault Systèmes. (n.d.). MagicGrid Book of Knowledge: A Practical Guide to Systems Modeling using MagicGrid. Retrieved from https://discover.3ds.com/magicgrid-book-of-knowledge
18. Capgemini Engineering. (n.d.). System Modeling Language (First Part).
19. Delligatti, L. (n.d.). SysML Distilled: A Brief Guide to the Systems Modeling Language. OMG SysML.
20. Dassault Systèmes. (n.d.). Retrieved from https://www.3ds.com
21. MathWorks. (n.d.). Retrieved from https://it.mathworks.com
22. No Magic. (n.d.). Integration with MATLAB. Retrieved from https://docs.nomagic.com

23. C.PULSE HANDOVER DOCUMENT This document includes all aspects about the journey of the C.PULSE development
24. Capgemini Engineering. (2025). Detect and Avoid Obstacles.
25. Capgemini Engineering. (2025). Detect and Avoid Obstacles, Simulink Integration.
26. Clarus Concept of Operations Archived 2009-07-05 at the Wayback Machine, Publication No. FHWA-JPO-05-072, Federal Highway Administration (FHWA), 2005.
27. https://bussinessofhardwareandnetworking.blogspot.com/2010/02/system-development-life-cycle-of.html
28. Israr Sharif (2022) The Difference between Business Process Management, System Analysis and Other Types of Business Process Modeling
29. Developing A Systems Engineering Capability That Meets the Needs of Your Organization - Scientific Figure on ResearchGate. Available from: https://www.researchgate.net/figure/Levels-of-Stakeholder-needs-and-Requirements-Ryan-2013-Reprinted-with-permission-from_fig1_319396633
30. Requirements for Devices Around Us: Embedded Systems. Available from: https://www.modernanalyst.com/Resources/Articles/tabid/115/ID/3150/Requirements-for-Devices-Around-Us-Embedded-Systems.aspx
31. Dr Awais Majeed (2015) Introduction to Requirements Engineering SEN-312 Software Requirements Engineering
32. https://commons.wikimedia.org/wiki/File:UML_diagrams_overview.svg
33. von Bertalanffy, L. (2004). Teoria generale dei sistemi. Fondamenti, sviluppo, applicazioni (E. Bellone, Trad.). Milano: Mondadori. (Opera originale pubblicata nel 1968)
34. Allen, W., & Kilvington, M. (n.d.). An introduction to systems thinking and tools for systems thinking. Learning for Sustainability. Retrieved from https://learningforsustainability.net/pubs/systemicdesign-intro.pdf

Appendix

List of Figures

List of Tables

List of Acronyms

Acronimo	significato
MBSE	Model-Based Systems Engineering
MBD	Model_Based Design
MBE	Model_Based Engineering
UML	Unified Modeling Language
SYSML	Systems Modeling Language
V&V	Verification and Validation
SOI	System of Interest
CONOPS	Concept of Operations
MOE	Mesure of Effectiveness
MOP	Mesure of Performance
TPM	Technical Performance Measures
RTM	Requirements Traceability Matrix
BRS	Business Requirements Specification
StRS	Stakeholder Requirements Specification
SyRS	System Requirements Specification
OpsCon	Operational Concept
bdd	Body Block Diagram
ibd	Internal Block Diagram
act	Activity Diagram
seq	Sequence Diagram
stm	State Machine Diagram
uc	Use Case Diagram
par	Parametric Diagram
pkg	Package Diagram
req	Requirement Diagram
CAD	Computer-Aided Design
HIL	Hardware-In-The-Loop
VR	Virtual Reality
AR	Augmented Reality
eVITOL	Electrical Vertical Take-OFF and Landing
API	Application Programming Interface
LSCO	Large Scale Combat Operation
AI-SODIA	Detect, Identify And Avoid System (with AI)
SODIA	Detect, Identify And Avoid System (without AI)
SW	software

SLAM	Simultaneous Localization And Mapping
PID	Proportional-Integrative-Derivative Controller
MPC	Model Predictive Control
X0	Coordinata x iniziale (2D model, static obstacle)
Y0	Coordinata y iniziale (2D model, static obstacle)
xnew	Coordinata x aggiornata (2D model, static obstacle)
ynew	Coordinata y aggiornata (2D model, static obstacle)
x	Coordinate x aggiornata (2D model, static obstacle)
y	Coordinate y aggiornata (2D model, static obstacle)
Vx0	Velocita lungo x drone (2D model, static obstacle)
Vy0	Velocita lungo y drone (2D model, static obstacle)
X_traj	Traiettoria desiderata lungo x (2D model, static obstacle)
Y_tray	Traiettoria desiderata lungo y (2D model, static obstacle)
trigger	Segnale booleano
Prev_trigger	Segnale booleano all'iterazione precedente
D_prev	Distanza all'istante precedente tra drone e ostacolo
D_curr	Distanza corrente tra drone e ostacolo
Vx_out	Velocità in uscita lungo x
Vy_out	Velocità in uscita lungo y
Vx	Velocità drone lungo x
Vy	Velocità drone lungo x
Vz	Velocità drone lungo x
V1	Velocità ostacolo 1
V2	Velocità ostacolo 2
V3	Velocità ostacolo 3
X1	Posizione iniziale ostacolo 1
X2	Posizione iniziale ostacolo 2
X3	Posizione iniziale ostacolo 3
det_Mark	Trigger che indica un ostacolo rilevato
Obs_pos	Posizione salvata dell'ostacolo

X	Posizione corrente drone lungo l'asse X
Y	Posizione corrente drone lungo l'asse Y
Z	Posizione corrente drone lungo l'asse Z
X_obs_dir	Direzione ostacolo lungo X
Y_obs_dir	Direzione ostacolo lungo Y
Z_obs_dir	Direzione ostacolo lungo Z
Col_mark	Trigger di collisione
Prev_col_mark	Trigger di collisione all'istante precedente
Xnew	Nuova posizione corrente lungo l'asse X
Ynew	Nuova posizione corrente lungo l'asse Y
Znew	Nuova posizione corrente lungo l'asse Z
t	tempo
P_Obs_pos	Posizione del drone all'istante precedente
P	Posizione del drone memorizzata