Analisi e implementazione di un sistema distribuito per la gestione e l'organizzazione di una flotta di droni

Monte Stefano

Dittaro Federico

DMIF, Università degli Studi di Udine, Italia

A.A 2023-2024

Indice

Capitolo 1

Introduzione

1.1 Il problema

Si dispone di una flotta di droni incaricati di effettuare consegne in una specifica area geografica. Questi droni sono veicoli autonomi di base, il cui compito principale è decollare da un punto di partenza e volare in linea retta verso la destinazione assegnata, mantenendo una velocità e un'altitudine costanti. Non sono dotati di capacità avanzate, come regolare la velocità, cambiare direzione durante il volo o rilevare ostacoli tramite sensori; le loro uniche abilità sono la misurazione della propria posizione e la comunicazione con gli altri droni tramite una connessione wireless.

L'obiettivo del progetto è sviluppare un sistema distribuito e flessibile che sfrutti la capacità di comunicazione dei droni per consentire loro di decidere in modo autonomo chi deve decollare e quando, evitando così collisioni, senza fare affidamento su un sistema centralizzato. Oltre alla prevenzione delle collisioni, il sistema deve affrontare altre sfide, come evitare situazioni di deadlock e starvation, e minimizzare i tempi di attesa per le consegne.

1.2 Assunzioni

- Considerato che l'altitudine di volo è la stessa per tutti i droni e che operano in un'area geografica limitata, il problema può essere modellato in due dimensioni e rappresentato su un piano cartesiano;
- Si assume che ogni drone sia in grado di comunicare con tutti gli altri droni della flotta;
- L'obiettivo è minimizzare i tempi di attesa per l'esecuzione delle consegne;
- La soluzione proposta deve includere meccanismi di tolleranza e gestione dei guasti, sia quelli rilevabili (che comportano un atterraggio di emergen-

za) sia quelli non rilevabili (ad esempio, dovuti alla perdita della connessione);

- Si suppone che un drone compaia fisicamente nel punto di partenza e scompaia nel punto di arrivo o in seguito a un atterraggio di emergenza.

1.3 Soluzione

La soluzione si basa sull'idea di far comunicare i droni tra loro prima del decollo, scambiando reciprocamente le informazioni sulle rispettive posizioni di partenza e destinazione. In questo modo, ciascun drone può calcolare in anticipo potenziali collisioni. Il problema può essere quindi modellato come un grafo, dove ogni nodo rappresenta un drone e ogni arco indica un conflitto di rotta. A partire da questo grafo, è possibile identificare facilmente le consegne che possono essere avviate immediatamente.

Per evitare collisioni, ogni gruppo di droni in conflitto decide autonomamente l'ordine di partenza, basandosi sullo scambio di una metrica che considera diversi fattori (descritti in dettaglio nella sezione 3.2.9). Al termine di questa procedura, ogni drone avrà una chiara comprensione dell' ordine di decollo.

Capitolo 2

Analisi

In questo capitolo verranno descritti i requisiti funzionali e non funzionali imposti dal problema.

2.1 Requisiti funzionali

Requisito	**Descrizione**	**Input**	**Output**
Nuova consegna	Viene istanziato un drone responsabile della consegna. La consegna ha successo se e solo se raggiunge il punto stabilito.	Due coppie di coordinate (X_{start}, Y_{start}) che rappresentano il punto di origine e (X_{end}, Y_{end}) che rappresentano l'arrivo	Messaggio di consegna avvenuta con successo oppure di errore.

2.2 Requisiti non funzionali

Requisito	**Descrizione**
Coordinamento droni	I droni devono collaborare per decidere chi e quando può volare. Un drone in volo non può essere fermato.
Minimizzazione tempi	Le consegne devono essere portate a termine nel minor tempo possibile.
Attese limitate	Il tempo di attesa prima della partenza di un drone deve essere limitato in modo da evitare deadlock o starvation.
Nuove consegne	Una nuova consegna può essere istanziata in qualsiasi momento, non ci sono inoltre limiti sul numero di droni contemporaneamente attivi nella rete.
Tolleranza ai guasti	I droni possono guastarsi durante il tragitto. Questi guasti non devono compromettere le altre consegne.
Localizzazione	I droni sono in grado di conoscere la loro posizione nello spazio.

Capitolo 3

Progetto

3.1 Architettura Logica

3.1.1 Rete distribuita di droni

La struttura principale del sistema è una rete distribuita, in cui ogni nodo rappresenta un drone. Considerando le assunzioni di partenza, secondo cui ogni drone può comunicare con tutti gli altri, la rete è rappresentata come un grafo completo. I compiti principali di ciascun nodo sono i seguenti:

- Comunicare il proprio ingresso nella rete a tutti i droni già presenti;
- Trasmettere la tratta che si intende percorrere e, se necessario, negoziare l'orario di partenza con gli altri droni con rotta conflittuale;
- Segnalare il completamento della consegna o la rilevazione di un guasto sia all'utente sia agli altri droni coinvolti, comunicando la propria uscita dalla rete;
- Comunicare l'annullamento della consegna all'utente e agli altri droni coinvolti in caso di perdita di connessione con la rete;
- In caso di guasto rilevabile, segnalare la causa e la posizione attuale, in modo da consentire all'utente di recuperare il drone.

3.1.2 Registro dei nodi

La seconda componente della rete è rappresentata da un registro di indirizzo noto, che contiene l'elenco di tutti i nodi attualmente attivi. Quando un drone desidera entrare nella rete, consulta innanzitutto il registro per ottenere gli indirizzi degli altri droni, ai quali comunicherà successivamente le proprie intenzioni e la tratta da percorrere.

Al momento dell'uscita dalla rete, sia essa standard o forzata, il drone notificherà l'evento al registro, affinché venga rimosso dalla lista dei droni attivi.

Un'alternativa al registro potrebbe essere l'uso di un broadcast per scoprire gli altri nodi già presenti nella rete. Tuttavia, il registro fornisce un metodo più rapido per ottenere questa informazione, sebbene richieda un aggiornamento continuo per mantenere l'elenco dei nodi accurato e aggiornato.

3.2 Protocolli e algoritmi

In questa sezione vengono descritte le caratteristiche che permettono di identificare ogni drone e le informazioni essenziali per il corretto sviluppo dei loro spostamenti.

3.2.1 Logica delle tratte

Ogni tratta viene identificata da:

- **Identificatore univoco (ID)**: un codice distintivo che consente di identificare in modo univoco ciascun itinerario di volo.
- **Timestamp di ingresso nella rete**: l'istante esatto in cui il drone si connette alla rete e inizia la comunicazione con gli altri nodi. Questo valore permette di determinare l'età del nodo, che verrà utilizzata per stabilire l'ordine di partenza in caso di conflitti.
- **Tratta di volo**: il percorso che il drone deve seguire, definito da un punto di partenza e un punto di arrivo, entrambi rappresentati da coordinate cartesiane sul piano.

3.2.2 Logica interna al drone

Ogni nodo (drone) possiede uno stato che può essere:

- **Init**: il drone è in fase di inizializzazione e si sta connettendo alla rete.
- **Arranging**: il drone sta negoziando il momento di partenza con altri droni in conflitto.
- **Waiting**: il drone è in attesa del proprio turno di decollo.
- **Travelling**: il drone è in volo e sta eseguendo la sua consegna.

Lo stato del nodo influisce direttamente sul modo in cui esso interpreta e risponde ai diversi tipi di messaggi scambiati all'interno della rete. In base allo stato attuale, il nodo può adottare comportamenti differenti, come accettare, rifiutare, elaborare o ignorare i messaggi ricevuti.

Per la gestione dei conflitti tra le tratte, ogni drone mantiene internamente tre liste:

- **Higher-Priority-List**: contiene le tratte che hanno una priorità maggiore rispetto alla propria e che quindi decolleranno prima.

- **Lower-Priority-List**: contiene le tratte che hanno una priorità inferiore rispetto alla propria e che dovranno attendere il proprio decollo.
- **Travelling-List**: contiene tutte le tratte che sono attualmente in volo.

Quando si verifica un conflitto tra la tratta di un drone e quella di un altro, quest'ultima viene inserita in uno dei suddetti set, in modo che il drone possa monitorarla e comportarsi di conseguenza in base alla priorità stabilita.

Se una tratta non interferisce con nessun altro percorso, può essere servita immediatamente e non verrà inserita in nessuna delle precedenti liste, poiché non richiede alcuna gestione aggiuntiva.

3.2.3 Ingresso di una tratta nella rete

Per prevenire collisioni, è fondamentale che ogni nodo sia consapevole dell'esistenza degli altri e stabilisca una comunicazione iniziale con essi; di conseguenza, il protocollo di avvio riveste un ruolo essenziale.

All'avvio, un nodo comunica con il gestore della rete per ottenere una lista di tutti i nodi presumibilmente attivi in quel momento. Successivamente, il nodo comunica a tutti i nodi attivi il percorso che intende seguire e attende di ricevere informazioni sui percorsi degli altri nodi, insieme ai dettagli sul loro stato operativo, con particolare attenzione ai nodi in stato *Travelling*.

Ad ogni ricezione di un messaggio da parte di un altro nodo, il nodo identifica eventuali conflitti e aggiunge i nodi coinvolti in una delle tre liste interne. Per garantire la sicurezza delle tratte, si può assumere che un nuovo viaggio, la cui priorità non è ancora stata determinata e che non è attualmente in volo, venga temporaneamente inserito nella *Higher-priority-List* per partecipare alle negoziazioni e stabilire l'ordine di partenza..

In linea di principio, se un nodo riceve una richiesta di negoziazione prima di aver completato la propria fase di inizializzazione, risponderà con una priorità pari a zero, consentendo di essere inserito nella *Lower-Priority-List* dagli altri nodi. Una volta completata l'inizializzazione, il nuovo nodo può avviare un nuovo ciclo di negoziazione per migliorare la propria posizione nella lista di priorità e ottenere l'autorizzazione a partire anticipatamente, se possibile.

3.2.4 Negoziazione

La fase di negoziazione è il momento in cui un insieme di nodi decide quali tratte possono partire immediatamente e quali devono attendere. Questo processo viene modellato come un grafo non orientato, dove ogni nodo rappresenta una tratta e ogni arco indica un conflitto tra due tratte. L'obiettivo è identificare un insieme indipendente di nodi che possono partire subito. Per i nodi rimanenti, viene definito un grafo aciclico diretto (DAG) per stabilire l'ordine delle partenze, dove un arco da un nodo a un altro indica che il primo deve attendere il secondo per poter partire. Alla fine della negoziazione, i nodi che possono partire immediatamente sono quelli che non hanno archi in ingresso e non sono in attesa di nessun altro nodo.

Dal punto di vista di ciascun nodo, gli archi in ingresso rappresentano i nodi con priorità inferiore (quelli nella *Lower-Priority-List*), mentre gli archi in uscita collegano i nodi con priorità superiore (quelli nella *Higher-Priority-List*).

Il protocollo di negoziazione segue i seguenti passi:

1. **Calcolo della Priorità**: ogni nodo calcola la propria priorità utilizzando una metrica definita e condivisa tra tutti i nodi. Questo calcolo avviene una sola volta per ogni round di negoziazione e la priorità calcolata viene poi comunicata a tutti i nodi presenti nelle due priority list.

2. **Aggiornamento delle Priorità**: all'inizio della negoziazione, tutti i nodi nella *Higher-Priority-List* sono considerati con priorità infinita. Man mano che i nodi ricevono le metriche dai vicini, aggiornano le priorità e spostano i nodi con priorità inferiore nella *Lower-Priority-List*. Questo processo permette a tutti i nodi di avere una visione chiara e coerente delle gerarchie di priorità. Alla fine dello scambio di metriche, ci sarà almeno un nodo che avrà priorità su tutti i suoi vicini e potrà partire per primo.

3. **Autorizzazione alla Partenza**: se, al termine della negoziazione, un nodo non ha archi in uscita (ovvero non è più in conflitto con nessuno nella *Higher-Priority-List*), può partire immediatamente, poiché ha vinto tutti i conflitti e non sta aspettando nessun altro nodo. La partenza deve essere notificata a tutti i nodi nella *Lower-Priority-List*, che di conseguenza sposteranno il nodo dalla *Higher-Priority-List* alla *Travelling-List*.

4. **Attesa e Comunicazione**: i nodi che non sono autorizzati a partire devono attendere ulteriori informazioni dai nodi nella *Travelling-List* o nella *Higher-Priority-List* per verificare se sono effettivamente in volo o ancora fermi. Dopo aver ricevuto queste informazioni, i nodi che rimangono fermi devono comunicare la propria intenzione di restare in attesa a tutti i nodi nella propria *Lower-Priority-List* attraverso un messaggio *StillHere*.

La procedura di negoziazione viene eseguita ogni volta che una nuova tratta entra nel sistema, garantendo che tutte le partenze siano coordinate in modo sicuro ed efficiente, evitando collisioni e minimizzando i tempi di attesa.

3.2.5 Fase di volo

Quando un drone comunica la sua intenzione di partire, viene inserito in un insieme denominato *Travelling-List*. La rimozione di un nodo da questo insieme avviene solo quando è garantito che, partendo in quel momento, il drone riuscirà a superare il punto di conflitto senza essere raggiunto da un altro in volo.

Per evitare attese superflue e garantire l'assenza di collisioni, quando un velivolo riceve una comunicazione riguardante la partenza o lo stato di volo di

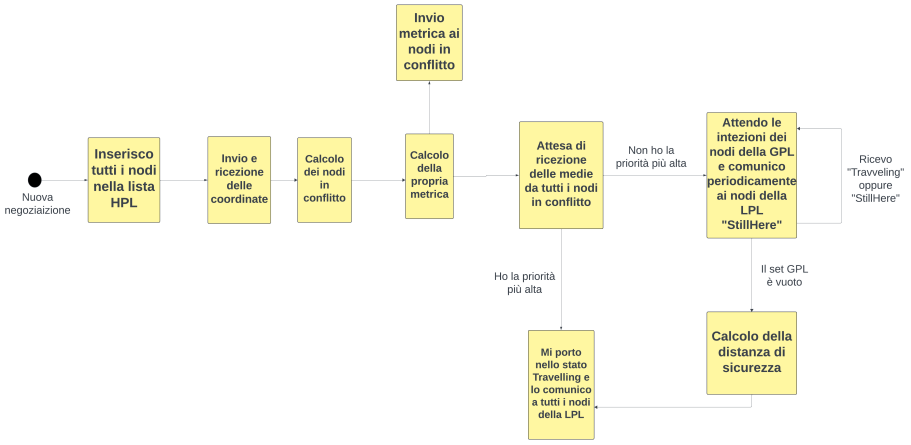

Figura 3.1: Round di negoziazione all'ingresso di un nuovo nodo della rete

un altro drone, può calcolare se ci sarà un conflitto. Questo calcolo si basa sulla previsione dei tempi di volo e sulle traiettorie. Se si prevede che due velivoli che partono simultaneamente rischiano di collidere, il drone con priorità inferiore deve attendere un intervallo di sicurezza minimo prima di iniziare il volo.

Questo intervallo di sicurezza assicura che il drone con priorità superiore completi il suo percorso senza incorrere in collisioni. La durata dell'intervallo è determinata in base a alla velocità, dalla distanza tra i punti di partenza e arrivo e dalla posizione prevista dei velivoli al momento del passaggio attraverso il punto di conflitto.

3.2.6 Complessità temporale e spaziale

Considerando un grafo dei conflitti con N nodi, in cui ogni nodo è connesso in media a M archi, la comunicazione avviene tramite l'invio di M messaggi per condividere la propria metrica e, al massimo, altri M messaggi per notificare l'intenzione ai nodi con cui ha prevalso. Di conseguenza, la complessità media del numero di messaggi scambiati risulta essere $\theta(M \cdot N)$. Tuttavia, nel caso peggiore, ossia nei grafi completi, tale complessità può raggiungere $\theta(N^2)$.

Dal punto di vista dello spazio, ogni nodo deve memorizzare informazioni relative a $\theta(M)$ conflitti con altre tratte. Dato che ogni messaggio occupa uno spazio costante di $\theta(1)$, la complessità spaziale complessiva rimane $\theta(M \cdot N)$.

3.2.7 Gestione dei conflitti

In questo sistema, ogni tratta è rappresentata come un segmento tra due punti specifici su un piano cartesiano. I conflitti tra tratte possono sorgere se i segmenti che le rappresentano si intersecano o sono troppo ravvicinati. Nel primo

caso, per determinare un conflitto tra due tratte, è necessario verificare l'intersezione tra i rispettivi segmenti e, in caso positivo, identificare il punto di intersezione. Nel secondo caso, in cui due tratte sono molto vicine ma non si intersecano direttamente, ogni tratta rappresenta le altre come rettangoli, ottenuti contornando i segmenti con un margine di sicurezza. In questo modo, il conflitto viene determinato in base all'intersezione più vicina al punto di partenza della propria tratta con il rettangolo dell'altra o delle altre in caso di più intersezioni.

In particolare, per calcolare il punto di conflitto tra una tratta del drone, indicata come S, e una tratta ricevuta per messaggio, indicata come S', si segue la seguente procedura:

1. **Costruzione del Rettangolo:** si determinano i vertici di un rettangolo che avvolge il segmento S'.
2. **Verifica del Punto di Partenza:** si verifica se il punto di partenza di S è contenuto all'interno del rettangolo. In tal caso, questo punto è considerato il punto di conflitto.
3. **Calcolo dei Punti di Incrocio:** se il punto di partenza di S non è contenuto nel rettangolo, si calcolano i punti di incrocio tra il segmento di S e il rettangolo di S'. Tra questi punti di incrocio, si seleziona il più vicino al punto di partenza.

3.2.8 Calcolo delle collisioni e tempi di sicurezza

Sulla base delle nostre assunzioni, il comportamento di volo dei nodi è altamente prevedibile: una volta che un nodo decolla, seguirà una traiettoria rettilinea a velocità costante, salvo in caso di un eventuale atterraggio di emergenza.

Quando il nodo Y decolla, il nodo X continuerà a muoversi lungo una traiettoria prevedibile. Pertanto, se Y decolla mentre X si trova a una distanza sufficientemente grande dal punto di conflitto, si può garantire che non ci sarà collisione. Questo perché X avrà già superato il punto di conflitto prima dell'arrivo di Y.

Per formalizzare questa intuizione, possiamo calcolare la differenza di tempo in secondi tra X e Y relativamente al punto di conflitto utilizzando la seguente formula:

$$\Delta T_{YX} = \frac{\text{dist}(Y.\text{start}, P)}{v_Y} - \frac{\text{dist}(X.\text{start}, P)}{v_X}$$

Se

$$\Delta T_{YX} > 0$$

significa che il drone in volo X arriverà in ritardo rispetto a quello in partenza Y al punto di collisione. In questo caso Y dovrà aspettare il tempo necessario affichè il drone X riesca a superare il punto critico per primo.

Pertanto, se la differenza di tempo ΔT_{YX} è > 0 il nodo con priorità inferiore dovrà attendere un tempo sufficiente per garantire che il nodo con priorità superiore superi la zona critica in sicurezza. Il tempo di sicurezza da rispettare sarà:

$$\text{Tempo di sicurezza} = \Delta T_{YX} + \alpha$$

Dove α rappresenta il tempo, espresso in secondi, necessario al drone per coprire una distanza di n metri, supponendo una velocità costante di m metri al secondo. Nel nostro caso, la velocità del drone è impostata a 8,3 m/s (pari a 30 km/h) e la distanza di sicurezza è fissata a 10 metri. Pertanto, α assume il valore di 1,2 secondi (arrotondato ad intero diventa 1 secondo).

Ad esempio, se la differenza di tempo ΔT_{YX} tra il drone Y e il drone X è di 2 secondi e il drone X ha una priorità maggiore rispetto a Y, quest'ultimo dovrà attendere $2 + 1 = 3$ secondi prima di poter partire. Al contrario, se $\Delta T_{YX} < 0$, non sono necessari provvedimenti, poiché il drone già in volo supererà la zona critica prima che il drone in partenza la raggiunga. In questo caso, il drone in attesa può decollare immediatamente senza ritardi.

3.2.9 Metriche sulla priorità dei voli

Per stabilire la priorità delle spedizioni, è stato adottato un metodo di valutazione che rispetta i seguenti criteri:

- **Prevenzione della starvation**: la metrica favorisce le consegne più vecchie rispetto a quelle più recenti.
- **Tempestività**: la metrica tiene conto della rapidità con cui una consegna può essere avviata, rispetto a quelle che devono attendere a causa di conflitti con consegne già in volo.
- **Gestione dei conflitti**: la metrica premia le consegne con meno conflitti attivi rispetto a quelle con un numero maggiore di conflitti.

La metrica proposta viene calcolata una sola volta per round, prevenendo così deadlock e incoerenze. La formula suggerita è la seguente:

$$m = (\text{età}) - |HPL + LPL| - \sum(\text{tempo di sicurezza di } HPL + LPL)$$

Questa metrica si basa su due informazioni principali conosciute da ogni nodo:

- **Tempo di attesa accumulato**: il tempo totale da cui la tratta è in attesa senza essere stata servita.
- **Tempo di attesa aggiuntivo**: il tempo che la consegna imporrebbe ad altre consegne nel caso fosse avviata, ossia la somma dei tempi di attesa imposti agli altri nodi.

Poiché la metrica è rappresentata da un numero naturale, in caso di parità si può utilizzare l'ID univoco della consegna per determinare un ordinamento e identificare un "vincitore" tra i nodi coinvolti in un conflitto, evitando così situazioni di deadlock. Al termine della negoziazione tra nodi, emergerà sempre una visione coerente su chi ha prevalso e chi ha perso.

3.2.10 Uscita dei nodi dalla rete, timeout e gestione dei crash

Un aspetto da considerare è la gestione dell'uscita dei nodi dalla rete, che può avvenire per vari motivi e deve essere gestita adeguatamente dai nodi rimanenti. Le uscite possono essere classificate in due categorie principali: quelle regolari e quelle eccezionali.

Uscite regolari:

- **Conclusione della consegna**: al termine della sua consegna, un drone comunica la propria uscita all'utente e al registro in modo da non essere più considerato come attivo.
- **Annullamento della consegna**: se un drone decide di annullare la consegna, ad esempio a causa di un malfunzionamento, deve notificare la decisione all'utente, al registro e a tutti i nodi con cui è in contatto, in modo che possano aggiornare le informazioni e gestire la situazione in modo appropriato.

Uscite eccezionali:

- **Inizializzazione**: se durante la fase di inizializzazione un drone non riesce a contattare alcuni nodi, non potrà acquisire informazioni sul loro stato, come se sono già in volo o se ci sono potenziali conflitti. In tali situazioni, il drone deve procedere con l'annullamento della consegna se non riesce a ottenere le informazioni necessarie entro un intervallo di tempo ragionevole.
- **Negoziazione**: durante la fase di negoziazione, se un drone non riesce a comunicare con uno dei nodi con cui è in conflitto, deve evitare di decollare per prevenire collisioni o situazioni di starvation per gli altri nodi coinvolti. Anche in questo caso, se non riceve risposte adeguate entro il termine stabilito, la consegna deve essere annullata.

In entrambi i casi di uscita eccezionale, il drone deve avvisare tutti gli altri nodi coinvolti per prevenire effetti a catena che potrebbero compromettere il funzionamento della rete.

3.3 Architettura fisica e dispiegamento

Nel nostro approccio, ogni nodo corrisponde a un drone e, di conseguenza, a un'unità di calcolo dedicata. Si può quindi immaginare che i droni siano connessi a una rete Wi-Fi e identificabili univocamente tramite il loro indirizzo IP. Questo consente di risolvere i problemi legati al routing e all'assegnazione degli identificatori univoci.

Un altro aspetto rilevante riguarda la scelta del protocollo di comunicazione tra TCP [3] e UDP [2]. In questo contesto, abbiamo optato per TCP, poiché consente di ridurre il numero di consegne fallite dovute alla mancata ricezione di un messaggio da parte di un altro drone.

3.4 Piano di sviluppo

Il piano di sviluppo si articola nelle seguenti fasi:

1. **Implementazione dei protocolli e degli algoritmi fondamentali**, che comprende:
   - **Inizializzazione dei nodi**: configurazione e avvio dei nodi all'interno della rete.
   - **Calcolo dei conflitti**: identificazione e analisi dei conflitti tra le consegne dei nodi.
   - **Negoziazione tra nodi e schedulazione delle partenze**: definizione delle priorità e pianificazione delle partenze in base ai conflitti e alle disponibilità.
   - **Implementazione del processo ambiente e creazione delle consegne**: configurazione dell'ambiente di esecuzione e definizione delle consegne da svolgere.
2. **Integrazione delle procedure di timeout e gestione degli errori**: per garantire la robustezza e l'affidabilità del sistema in situazioni di errore e malfunzionamento.

Capitolo 4

Implementazione

4.1 Linguaggio utilizzato

Il progetto è stato realizzato utilizzando il linguaggio C# [1] e la libreria AKKA.NET [4]. Quest'ultima è un framework open-source utilizzato per lo sviluppo di applicazioni distribuite, scalabili e altamente concorrenti su piattaforme .NET. Ispirato ad Akka per la JVM, AKKA.NET implementa il modello di attori (actor model), un paradigma di programmazione concorrente che consente di gestire in modo efficiente il parallelismo e la concorrenza in ambienti multithread. Ogni processo logico è rappresentato da un attore, ovvero un'entità capace di inviare e ricevere messaggi, identificato unicamente dall'*ActorRef* e creato all'interno di un *ActorSystem*.
La comunicazione tra attori avviene tramite lo scambio di messaggi:

- Per quanto riguarda l'invio di un messaggio, si può utilizzare il metodo **Tell** (*Fire-And-Forget*) oppure il metodo **Ask** (*Send-And-Receive-Future*);
- I messaggi inviati vengono ricevuti all'interno di una mailbox e gestiti dal rispettivo attore tramite la direttiva **Receive<Type>(Condition, Action)** che permettono di collegare una specifica procedura ad un certo tipo di messaggio.

Nella soluzione realizzata ed illustrata nei prossimi capitoli si è deciso di creare un *ActorSystem* per ogni drone. Quest'ultimo quando desidera avviare una consegna spawna un'attore per poterla rappresentare a livello logico. Inoltre alcuni servizi, come ad esempio la simulazione di volo, generano degli ulteriori sotto-attori.

4.2 Protocolli principali

In questa sezione verrà descritta l'implementazione dei protocolli eseguiti da un drone dal momento in cui entra nella rete a quello in cui esce (per completamento

della consegna o a causa di errori). Sono quindi compresi anche gli algoritmi necessari per la negoziazione e per il volo.

4.2.1 Attore del drone

Il file *Drone* implementa la componente principale del sistema. Le responsabilità affidate a questa classe sono l'avviamento di tutte le procedure di inizializzazione e la ricezione dei vari messaggi (che verranno poi gestiti dalle altre componenti interne). La classe **Drone** contiene le seguenti due classi principali:

- **DroneContext**: contiene le informazioni basilari del drone e della rispettiva consegna, come il timestamp di spawn, l'elenco di tutti gli altri droni e le specifiche della consegna;
- **DroneState**: contiene la logica dei vari stati descritti nella sottosezione 3.2.2

4.2.2 Stati del drone

Oltre ai quattri stati definiti nella sottosezione 3.2.2, è stato aggiunto un quinto stato (*ExitState*) per gestire la fase di uscita di un drone dalla rete. Sebbene riguardino momenti diversi di vita di un drone, possono essere trovati degli elementi comuni quali l'esecuzione di una procedura di inizializzazione, la gestione dei messaggi (a volte in modo identico ad altri stati mentre altre volte in maniera totalmente differente), il mantenimento di informazioni interne e la capacità di decisione di quale deve essere lo stato successivo.
La classe base per la realizzazione di questa struttura è il file **DroneState** le cui classi figlie rappresentano l'implementazione dei cinque stati appena descritti. La classe contiene una serie di metodi a volte implementati direttamente (ad esempio il caso di ricezione di una richiesta di connessione) mentre altri forniti dalle classi figlie. Al termine del file è inoltre presente un metodo astratto *RunState* utilizzato dalle classi figlie per l'implementazione della procedura di inizializzazione. Tutti i metodi **OnReceive(msg, sender)** e **RunState** restituiscono come output un'istanza di **DroneState** che rappresenta lo stato successivo.
In questo modo l'attore riesce sempre a eseguire la procedura di gestione dei messaggi corretta e inoltre, grazie all'ereditarietà, i vari stati possono condividere le parti comuni.

4.2.3 Strutture dati ausiliarie

Nel progetto si è fatto spesso uso di dati non primitivi (implementati come classi). Di seguito vengono riportati i più significativi.

1. **DeliveryPath**: rappresentazione della tratta, costituita da un punto di partenza, un punto di arrivo ed una velocità (secondo le assunzioni di partenza identica per tutti i droni) . Questa classe ha la responsabilità

di svolgere tutte le operazioni di tipo geometrico, ovvero la verifica di un eventuale conflitto e la possibilità di ricavare la distanza temporale tra il punto di partenza e una determinata posizione;

2. **Delivery**: costituita da una tratta e da un riferimento al drone che la sta eseguendo, rappresenta una generica consegna. Nello specifico questa classe viene implementata tramite due realizzazioni concrete: **WaitingDelivery** che rappresenta una consegna ancora in attesa caratterizzata da una priorità mutabile e **TravellingDelivery** che rappresenta una consegna in volo. Quest'ultima non è caraterizzata da una priorità ma dall'istante in cui si è venuti a conoscenza della sua partenza. Questa classe offre inoltre un metodo per calcolare il tempo necessario per una partenza sicura.

3. **Priority**: è costituita da una metrica e dal corrispondente nodo, rappresenta la priorità di un determinato drone. Questa classe contiene un metodo che permette di confrontare due metriche, l'identificatore viene utilizzato solo nel caso in cui la metrica sia identica (secondo le assunzioni non possono esistere due droni con la stessa priorità). Sono inoltre presenti due classi figlie: **MaxPriority** utilizzata per vincere tutte le negoziazioni (quando il nodo è già in volo) e **MinPriority** utilizzata per perderle tutte (caso in cui il nodo è ancora in fase di inizializzazione).

Per implementare strutture come *ConflictList*, *TravellingList* e i due set per gestire i droni con priorità maggiore o inferiore alla propria, sono state create le classi **ConflictList** e **TravellingList**. Entrambe le strutture estendono la classe astratta **IDeliveryList<D>** che rappresenta una collezione di consegne in cui le istanze si creano internamente e la gestione avviene solo tramite riferimento al nodo.

4.2.4 Spawn dei droni

L'aspetto probabilmente più cruciale di tutto il progetto riguarda lo spawn del drone ed in particolare la capacità di reperire la lista di tutti i nodi già presenti in rete. Nel caso in cui il drone non sia in grado di reperire anche solo un riferimento, rischia di causare collisioni sconosciute. Così come discusso nel capitolo 3, si è scelta l'opzione di utlizzare un registro di indirizzo noto per repirire questa informazione permettendo così di poter spawnare i droni da diversi terminali. L'idea è quindi quella che ogni drone durante la fase di inizializzazione contatti il registro per ottenere la lista dei droni attivi (a sua volta il registro aggiungerà il nuovo nodo). Il registro è implementato nella classe **DronesRegister**. Il registro deve essere inoltre in grado di mantenere una lista coerente dei nodi attivi anche in caso di uscite irregolari da parte dei droni. Per ottenere questo risultato si utilizza il metodo **Context.Watch** già presente in AKKA.NET

4.2.5 Simulazione del volo

Quando un drone entra nello stato *Travelling* ci si aspetta che la sua posizione vari nel corso del tempo e che si renda conto di quanto ha raggiunto il punto di destinazione. La simulazione avviene creando, al momento della partenza, un attore figlio che periodicamente aggiorna la propria posizione e che all'arrivo a destinazione notifichi all'attore principle l'avvenimento (attraverso un messaggio interno).

4.2.6 Attesa delle consegne in volo

Un aspetto importante dell'algortimo è l'attesa delle consegne in volo. Per risolvere il problema è stata definita la classe **ControlTower** che contiene un riferimento alla consegna corrente e alla *TravellingList*. Nel momento in cui un drone comunica la sua partenza o il suo stato di volo si esegue quanto descritto di seguito:

- Alla ricezione di uno dei due messaggi, l'istanza della consegna viene rimossa dalla *ConflictList* e passata alla classe *ControlTower* che la aggiunge alla *TravellingList* e crea un timer per garantire una partenza sicura;
- Scaduto il timer viene inviato al drone il messaggio interno **ClearAirSpaceMessage**. Questo evento interno viene gestito dallo stato corrente del drone.

4.2.7 Terminazione della consegna

Quando un drone raggiunge la destinazione o avviene un errore osservabile il nodo passa nello stato *Exit*. Questo comporta l'invio del messaggio **ExitMessage** verso tutti i nodi conosciuti che a loro volta rimuovono il mittente dalla lista dei nodi e dalla *ConflictList* o dalla *TravellingList*.

4.3 Gestione degli errori

La gestione degli errori è una parte fondamentale dello sviluppo del sistema, garantendo un comportamento prevedibile anche in situazioni straordinarie e non previste.

Durante lo sviluppo, la principale criticità riscontrata è stata la gestione dei messaggi, in particolare quando questi non venivano consegnati correttamente al destinatario e venivano persi. Questo causava il blocco dei droni, che rimanevano in attesa di messaggi che non sarebbero mai arrivati, senza sapere come procedere.

Per risolvere questa criticità, è stato implementato un sistema di timeout per la ricezione dei messaggi nelle varie fasi del protocollo di inizializzazione e negoziazione.

In fase di inizializzazione, il timeout riguarda la ricezione dei messaggi contenenti le tratte degli altri nodi presenti nella rete. In fase di negoziazione,

invece, il timeout viene applicato durante la ricezione delle metriche dai nodi della propria lista dei conflitti, dopo aver inviato le proprie metriche, durante la ricezione delle intenzioni dei droni con priorità superiore alla propria e dopo aver ricevuto le loro metriche.

Dal punto di vista implementativo, per ciascuna di queste fasi critiche è stato introdotto un timer con un tempo massimo di attesa. Se tale tempo viene superato, per evitare il collasso del sistema, al drone in attesa viene inviato un messaggio che segnala la scadenza del tempo. Una volta ricevuto il messaggio, il drone fallisce automaticamente la consegna ed esce dalla rete. Se invece tutti i messaggi arrivano nei tempi previsti la schedulazione viene semplicemente annullata.

4.4 Interfaccia

Si è deciso di implementare un'interfaccia per poter comunicare con il sistema, in modo da poter visualizzare dall'esterno ciò che sta accandendo.

4.4.1 API per interfacciarsi con le consegne

Si è deciso di implementare delle API per monitorare le consegne, in particolare per comunicare il loro inserimento, la loro partenza e lo loro fine.

La classe astratta per la loro implementazione è *IDeliveryAPI*. Essa definisce i metodi principali con cui interagire con il sistema di droni, come per esempio inserire una nuova consegna nel sistema, ricevere una notifica in caso di partenza e arrivo a destinazione e segalare l'eliminazione della consegna dal sistema una volta completata.

L'implementazione vera e propria viene fatta nella classe DeliveryAPI utilizzando le primitive di Akka.NET Ask() per inviare i messaggi, tramite l'utilizzo di un attore temporaneo e ricevere le risposte che verranno proiettate nel terminale in modo da essere facilmente fruibili dall'utente.

4.4.2 API per la creazione dell'ambiente

Si sono sviluppate anche delle classi API per rendere più intuitivo e comodo il dispiegamento del sistema e dei suoi attori (droni e registro).

Nel codice utilizzato per la creazione dei droni, viene inizializzato un ActorSystem all'interno del quale viene creato un attore spawner. Per rendere il processo più elegante esso può essere generato attraverso una classe Factory che permette di creare l'ActorSystem e avviare lo spawner con una singola chiamata di metodo. Nel codice dell'interfaccia o del servizio di prenotazione, il programmatore utilizza la classe *DeliveryAPI* per eseguire diverse operazioni. Prima di tutto, viene impostato un registro, successivamente, le consegne vengono avviate tramite il metodo '*Create-Delivery*, dove il parametro *Port* consente di specificare la porta del drone. Infine, le istanze restituite di *IDeliveryAPI* vengono utilizzate per monitorare la partenza e arrivo delle consegne.

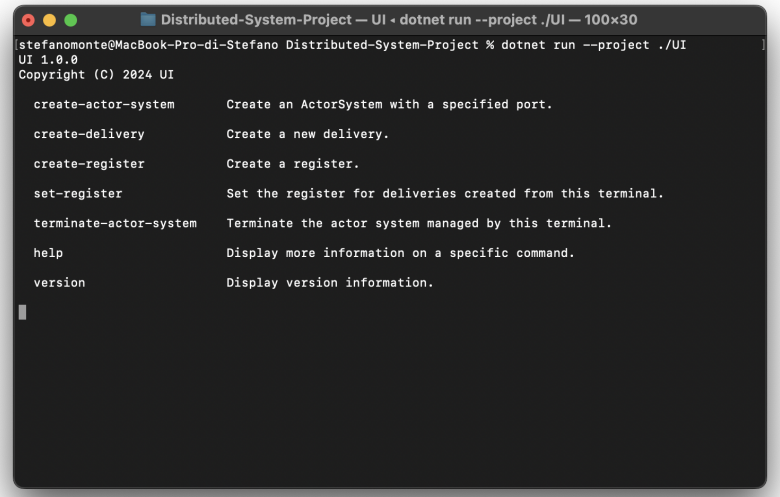

Figura 4.1: Schermata principale dell'interfaccia da terminale.

4.4.3 Interfaccia da terminale

Come accennato, si è deciso di sviluppare un'interfaccia da terminale per la gestione delle consegne. In particolare, la creazione del sistema, del registro e di nuove consegne è possibile attraverso l'interfaccia. Ogni comando fornisce un feedback sullo stato attuale del sistema in modo da poter valutare in ogni momento lo stato delle consegne (partenza e arrivo a destinazione). L'applicazione opera in modalità stateful, il che significa che i comandi modificano lo stato interno dell'applicazione, influenzando anche il relativo output. In particolare, vengono memorizzati gli *ActorSystem* creati, insieme alle porte TCP associate, e le consegne generate o monitorate. Per gestire il parsing dei comandi, è stata utilizzata la libreria CommandLine.

4.4.4 Funzionamento dell'interfaccia

Per prima cosa, è necessario compilare il progetto eseguendo il comando:

```
dotnet build ./DistributedSystemsProject.sln
```

Per utilizzare l'interfaccia utente da terminale, è possibile avviare una o più istanze della console seguendo i passaggi indicati:

1. **Avvio delle console**: aprire una o più istanze della console all'interno della directory del progetto.

2. **Avvio dell'interfaccia**: per avviare l'interfaccia, eseguire il comando seguente in ciascuna istanza:

   ```
   dotnet run --project ./UI
   ```

3. **Inizializzazione dell'ActorSystem**: in ogni istanza, creare un `ActorSystem` utilizzando il comando:

   ```
   create-actor-system -p PORTA
   ```

   Dove `PORTA` rappresenta il numero di porta da utilizzare per la comunicazione.

4. **Creazione del registro**: su una delle istanze dell'`ActorSystem`, avviare il registro con il comando:

   ```
   create-register -p PORTA
   ```

5. **Connessione al registro**: nelle altre console, collegare ogni `ActorSystem` al registro utilizzando il comando:

   ```
   set-register -p PORTA
   ```

   Inserire la stessa porta specificata nel comando di creazione del registro.

6. **Avvio di una consegna**: per iniziare una consegna, utilizzare il comando:

   ```
   create-delivery x1 y1 x2 y2 -n NOME -p PORTA
   ```

   Questo comando avvierà una consegna dalle coordinate `x1, y1` a `x2, y2`, associata al nome e alla porta scelti.

7. **Terminazione di un ActorSystem**: per terminare un `ActorSystem`, utilizzare il comando:

   ```
   terminate-actor-system
   ```

8. **Uscita dall'interfaccia**: per uscire dall'interfaccia terminale, eseguire il comando:

   ```
   exit
   ```

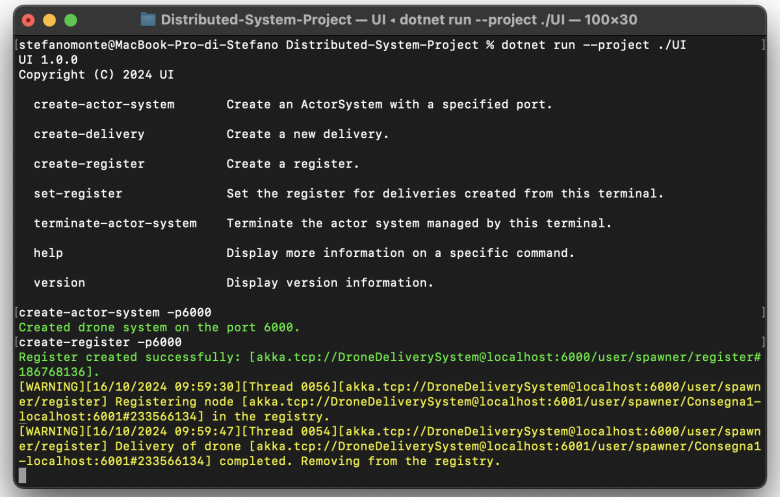

Figura 4.2: Schermata dell'interfaccia da terminale dalla parte del registro dei droni dopo aver effettuato una consegna.

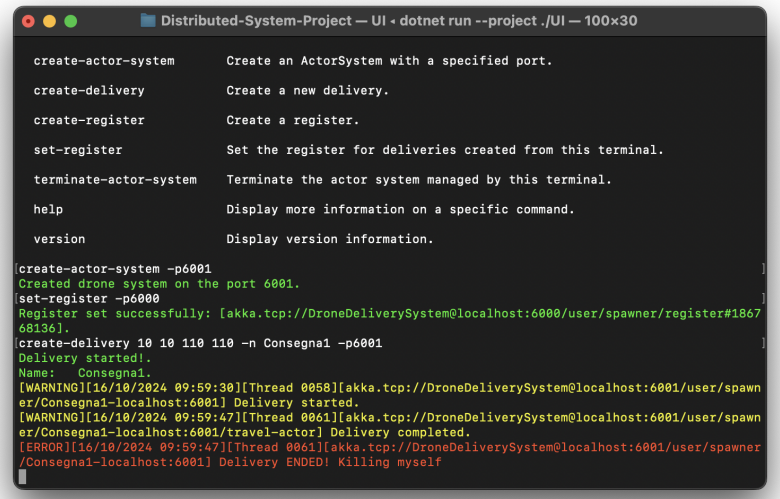

Figura 4.3: Schermata dell'interfaccia da terminale dalla parte del drone dopo aver effettuato una consegna.

Capitolo 5

Validazione

Per validare il sistema e verificare il corretto funzionamento di tutte le sue componenti, sono stati sviluppati test unitari automatizzati utilizzando xUnit [6], insieme al TestKit [5] fornito da Akka.NET. È importante notare che tutti i test sono stati eseguiti in un ambiente controllato, con un singolo ActorSystem. Inoltre, è fondamentale ricordare che il principio fondamentale del testing stabilisce che il superamento dei test non garantisce la correttezza del programma, mentre il fallimento di un test rappresenta una chiara indicazione di non correttezza.

5.1 Test sugli attori

Per il testing degli attori, si è scelto un approccio in cui, inizialmente, vengono dispiegati uno o più droni nell'ambiente. Successivamente, si interagisce con essi simulando il comportamento di un drone o di un registro, infine verificando il corretto funzionamento dei protocolli tramite una sequenza di asserzioni. È fondamentale che gli eventi osservati nel sistema corrispondano a quanto previsto a priori.

Queste tre attività vengono automatizzate per semplificare e velocizzare il processo di testing.

I test di base sugli attori delle consegne includono il controllo del comportamento in diverse condizioni. Ad esempio, viene testato il corretto avvio di una consegna in uno spazio libero, dove ci si aspetta che la consegna parta senza problemi. Un test simile viene effettuato in uno spazio occupato ma privo di conflitti, con lo stesso esito previsto. Successivamente, vengono simulate situazioni di conflitto, verificando che il drone esegua correttamente il protocollo di negoziazione in caso di conflitto vinto e parta, oppure attenda in caso di conflitto perso. Un altro scenario simulato prevede lo spawn in un ambiente in cui è già in corso una consegna a rischio di collisione, con il drone che attende prima di partire. È stato anche testato un conflitto in cui due nodi hanno la stessa metrica, verificando che parta il nodo con ID minore.

I test più complessi hanno incluso la simulazione di un caso che richiede un secondo round di negoziazione, oltre a verifiche su conflitti senza una vera intersezione delle tratte, per controllare il calcolo corretto del margine di sicurezza necessario per evitare collisioni.

Sono stati inoltre eseguiti test per valutare la capacità dei droni di gestire errori, simulando timeout durante la fase di inizializzazione, negoziazione o durante la ricezione delle intenzioni dopo la negoziazione. Alcuni test sono stati ripetuti introducendo un registro dei nodi, per verificare il suo corretto funzionamento.

5.2 Test sulle API

Analogamente, sono stati condotti test sulle API, in particolare per verificare la creazione e l'impostazione del registro dei droni, nonché le funzioni legate alle consegne.

Per quanto riguarda il registro, è stato eseguito un primo test che prevedeva l'avvio del registro dei droni tramite l'API e la successiva verifica del suo corretto funzionamento. Un secondo test ha valutato l'avvio del registro senza l'uso dell'API, seguito dalla connessione tramite API per accertarne l'integrità operativa.

I test legati alle consegne hanno incluso l'avvio di due consegne utilizzando l'API e il lancio di una consegna senza l'API. È stato inoltre testato uno scenario in cui si tentava di avviare una consegna senza un registro attivo; in questo caso, il fallimento dell'operazione era previsto e confermato. Successivamente, si è ripetuto il test con un registro attivo, ottenendo un esito positivo.

Capitolo 6

Conclusioni e sviluppi futuri

Il progetto ha raggiunto tutti gli obiettivi prefissati con successo, dimostrando un funzionamento conforme alle aspettative. Gli obiettivi principali, tra cui la creazione e gestione degli attori, l'implementazione di protocolli di comunicazione e la gestione delle consegne, sono stati completati senza incontrare criticità significative. Il sistema è ora in grado di avviare e gestire consegne in maniera affidabile, risolvendo efficacemente eventuali conflitti e operando in modo coordinato con più nodi e attori distribuiti.

Inoltre, è stata implementata con successo una gestione degli errori, inclusi meccanismi di timeout e notifiche per gli utenti, che garantiscono una maggiore solidità del sistema anche in condizioni non ottimali. Questo rende il sistema resiliente, capace di rispondere prontamente a eventi imprevisti e di ripristinare la normale operatività in caso di anomalie.

Guardando agli sviluppi futuri, il sistema potrebbe beneficiare dell'introduzione di ulteriori funzionalità. Un'interfaccia grafica utente (GUI) rappresenterebbe un miglioramento significativo in termini di usabilità, rendendo il sistema più accessibile e intuitivo. Inoltre, si potrebbe integrare un modulo di analisi predittiva per ottimizzare la pianificazione e l'esecuzione delle consegne, aumentando l'efficienza operativa. Infine, l'espansione dell'API per supportare la comunicazione con piattaforme esterne o l'inclusione di nuovi protocolli di comunicazione consentirebbe una maggiore flessibilità, rendendo il sistema adatto a un numero più ampio di applicazioni e scenari operativi.

Bibliografia

[1] Microsoft. C# programming language. https://docs.microsoft.com/en-us/dotnet/csharp/, 2024. Accessed: 2024-10-14.

[2] Jon Postel. User datagram protocol (udp). *RFC 768*, 1980. Accessed: 2024-10-14.

[3] Jon Postel. Transmission control protocol (tcp). *RFC 793*, 1981. Accessed: 2024-10-14.

[4] Akka.NET Team. Akka.net. https://getakka.net/, 2024. Accessed: 2024-10-14.

[5] Akka.NET Team. Akka.net testkit. https://getakka.net/articles/testing/testkit.html, 2024. Accessed: 2024-10-14.

[6] xUnit Team. xunit: A free, open-source unit testing tool for .net. https://xunit.net/, 2024. Accessed: 2024-10-14.