

TAXI AUTONOMO

SIMULAZIONE ROBOTICA + INTERAZIONE INTELLIGENTE

CORSO DI LAUREA MAGISTRALE IN INGEGNERIA INFORMATICA
CURRICULUM INTELLIGENZA ARTIFICIALE
ANNO 2025-2026

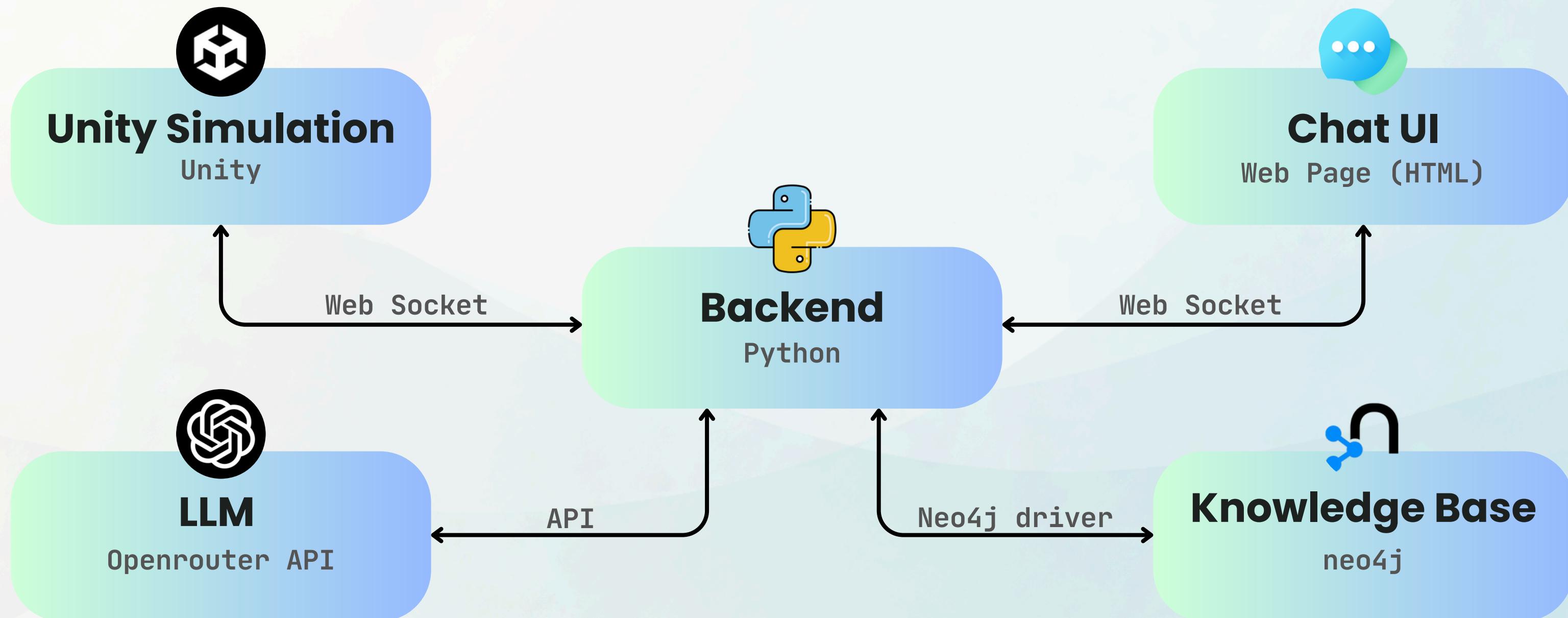
Membri del team:

- Davì Ernesto
- Dioguardi Roberto
- Ferrara Salvatore
- Spezia Salvatore

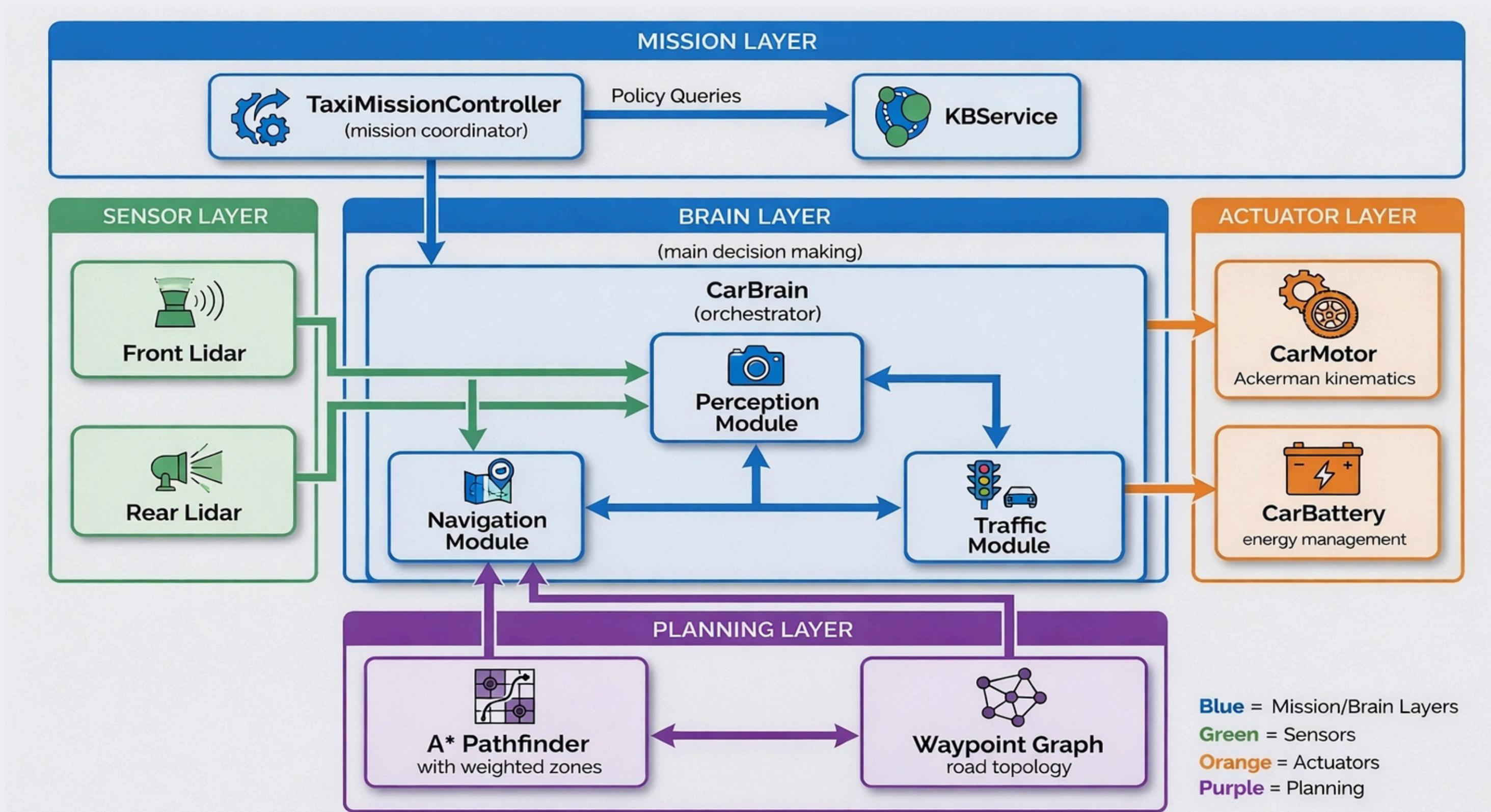
Docenti:

- Chella Antonio (Robotica)
- Seidita Valeria (Intelligenza Artificiale 2)

Architettura del Sistema



Architettura del Controllore Robotico



Ambiente Dinamico

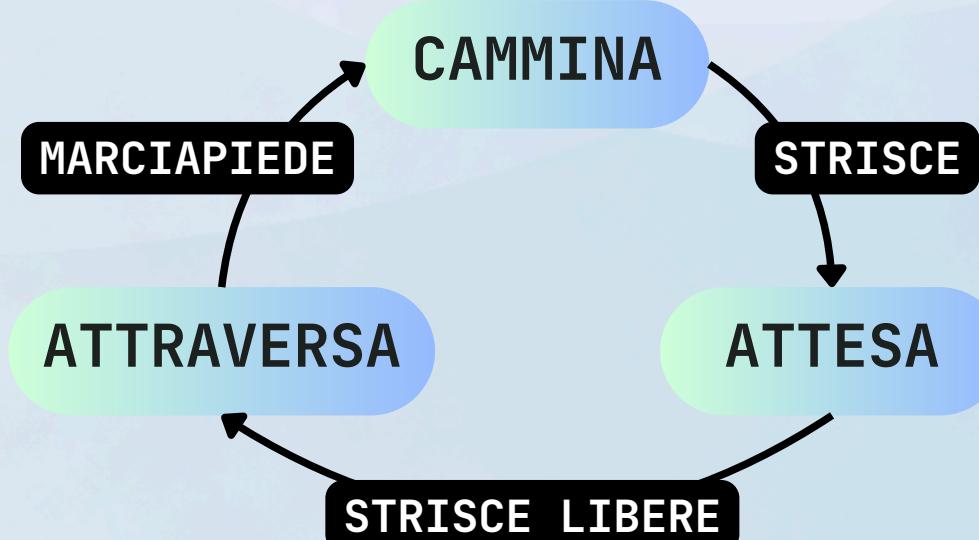
Auto NPC 🚗

- FSM
- LiDAR per Ostacoli
- Rispondono ai **Semafori**



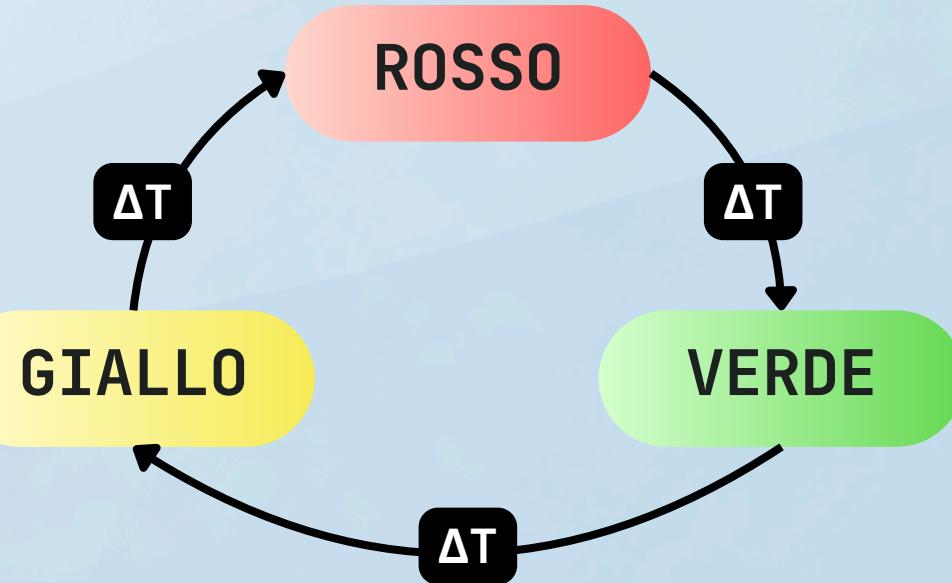
Pedoni NPC 🚶

- Area pedonale e Strisce
- Personalità:
 - **Calma**: FSM
 - **Impulsivo**: corre e attraversa senza guardare



Semafori 🚦

- FSM
- **Simulazione V2I**: viene rilevato automaticamente dalle auto (5G)



Ambiente Dinamico

Barriere

- **Spawn** in posizione **casuale** nel percorso verso la destinazione
- **Ricalcolo del percorso** (A*)

Meteo

- **Pioggia** con timeout casuale
- **Superfici scivolose** → **Ricalcolo del percorso** (A*) (Policy di Guida)

Tempo

- **Orari di apertura/chiusura POI**
 - **Regole di traffico** (KB)
 - **Ricalcolo del percorso** (A*) (Policy di Guida)
-
- **Fattore di scala pari a 12x**: ogni secondo reale equivale a 12 s simulati
 - **Tempi di stima** dei percorsi scalati sul tempo simulato

Ambiente Dinamico



Mappa divisa in **Zone**:

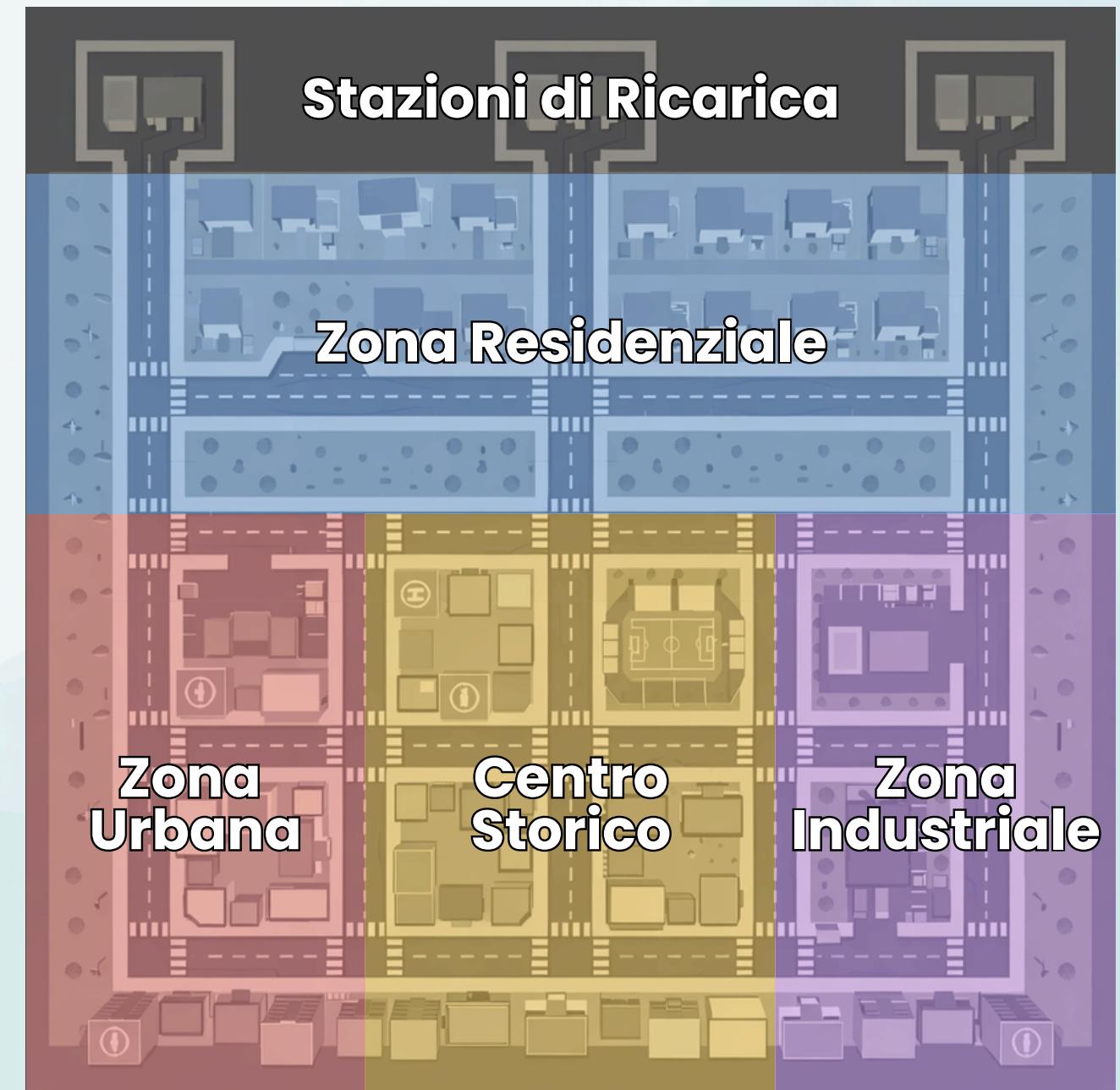
Zone	Superficie
Centro Storico	Sanpietrino/Pavé
Zona Residenziale	Dossi
Zona Urbana	Asfalto liscio
Zona Industriale	Strada dissestata

Policy di Guida:

Comfort
Odia
vibrazioni

Eco
Ottimizza i
consumi

Sport
Odia
rallentamenti



Navigazione



- **Grafo di WayPoints**
- A* trova il percorso ottimo
- **Distanza:** distanza euclidea tra waypoints
- I **pesi** degli archi dipendono da:
 - **Policy e Superficie delle Zone**
 - **Meteo** (es. Pioggia)
 - **Traffico**

PESO ARCO = DISTANZA × PESO POLICY



Partenza



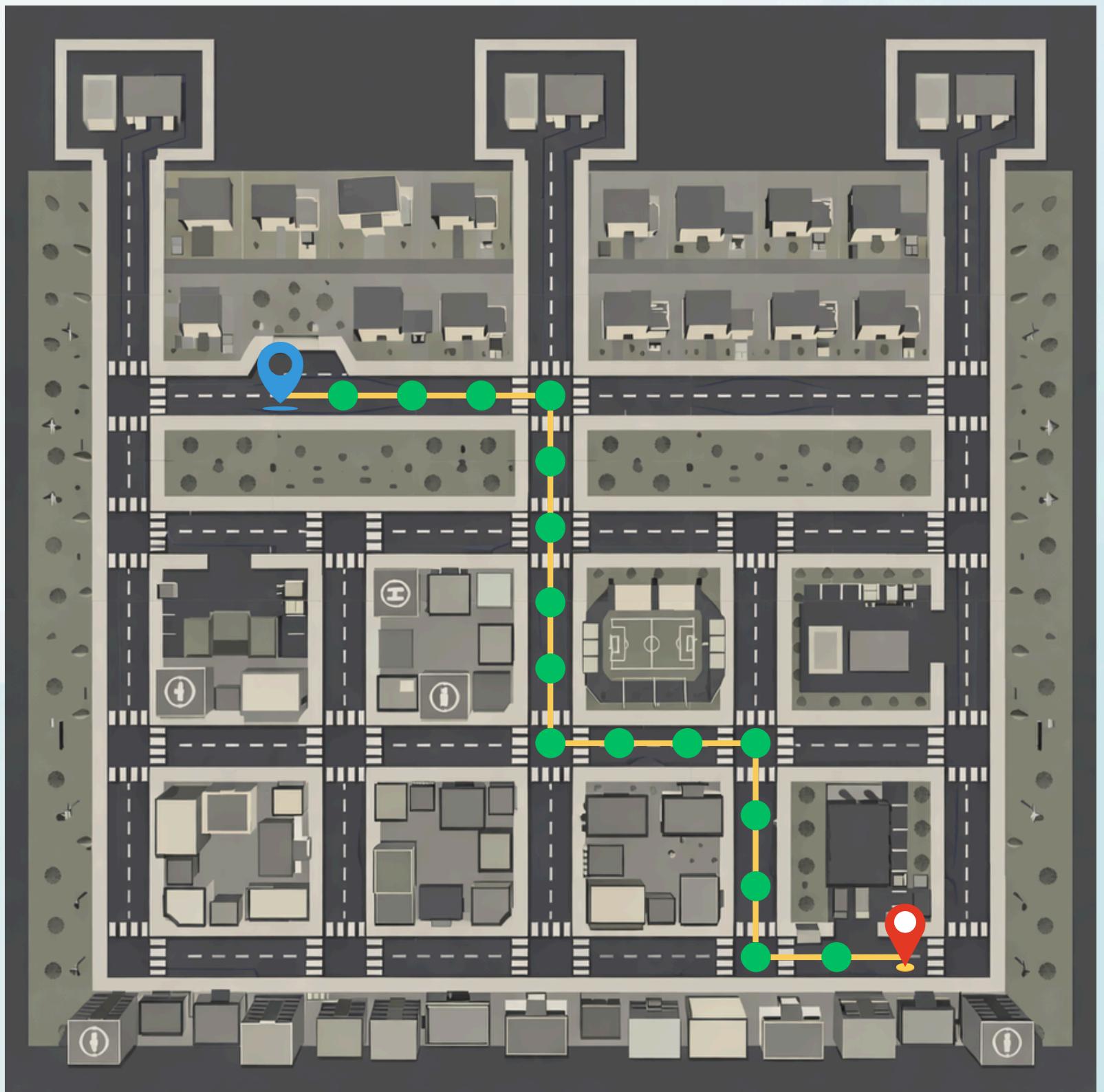
Destinazione



Waypoints



Archi pesati



Sistema Reattivo ⚡

Priorità: Sicurezza > Traffico

Percezione:

- **LiDAR** → distanza e posizione
- **Tag Unity** per **object detection** (cv simulation)
- **Semafori via V2I**: lo stato è letto da *SmartTrafficLight* quando il taxi è dentro un raggio azione del semaforo

Decisione diversa per oggetto:

Object	Trigger	Azione	Note
Pedoni	LiDAR + Tag	Stop se distanza < soglia Rallenta altrimenti	Priorità Massima
Auto	LiDAR + Tag	Stop / Rallenta in coda Sorpasso se sicuro	Sorpasso solo se corsia libera e semaforo verde
Barriera	LiDAR + Tag	Stop + Ricalcolo A*	Nodo bloccato
Semaforo	V2I	Rallenta / Stop	Rosso o Giallo
Altro ostacolo	LiDAR + Tag	Stop / Sorpasso	Pali, Transenne, ecc.



Gestione delle Missioni (FSM)



Coda di Prenotazioni

Gestione FIFO delle richieste di prenotazione

Gestione della Batteria

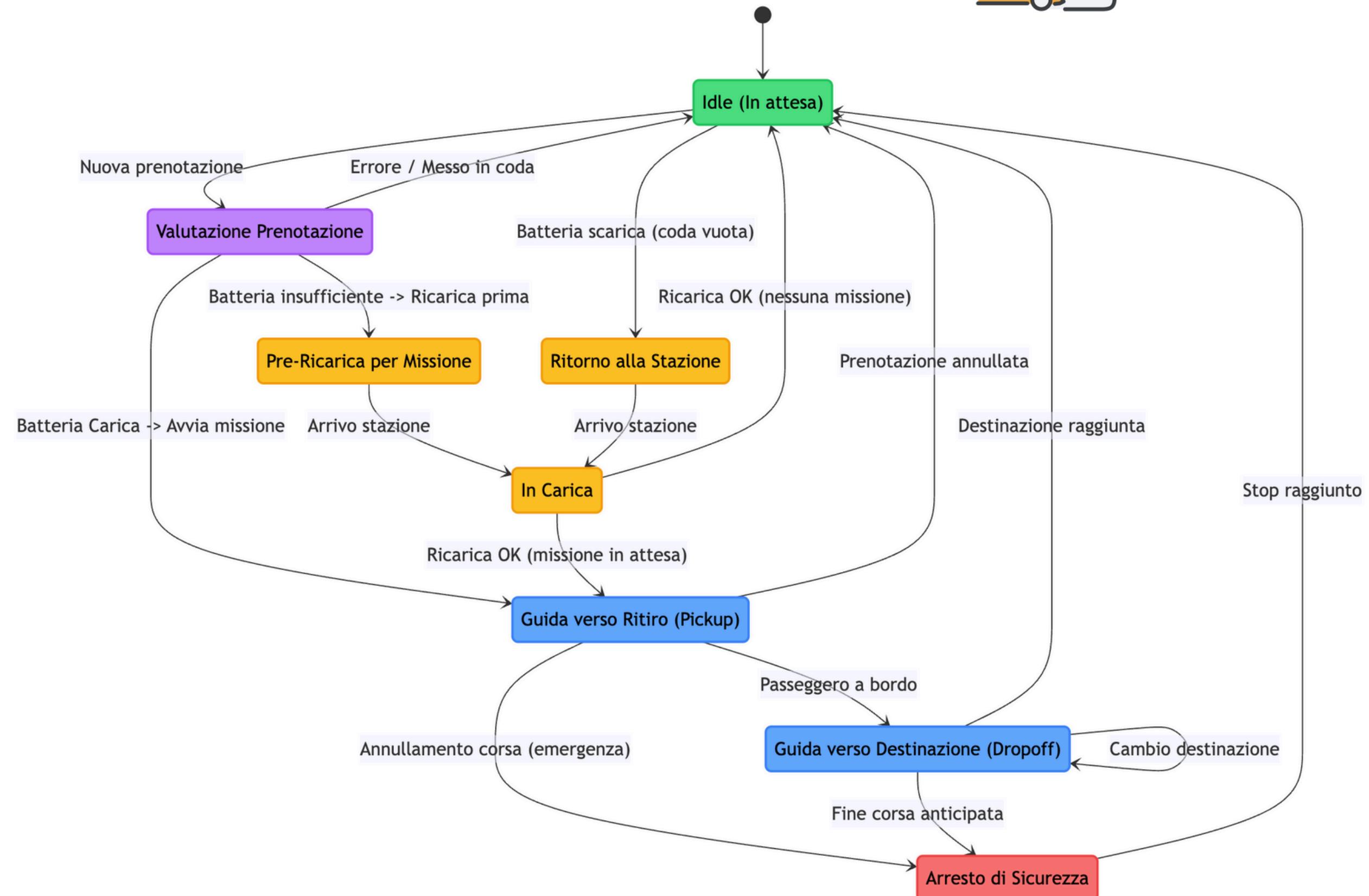
Stima della batteria (km) e andare a ricaricare

Pickup & Dropoff

- Caricare e scaricare il cliente
- Cambio di destinazione

Arresto

- Annullamento corsa
- Fine corsa anticipata



Gestione della Batteria



Consumo reale ⚡

- **Consumo** basato sulla **distanza**
- **Non consuma** quando è fermo o è in ricarica
- La **policy** attiva modifica il consumo

CONSUMO = DISTANZA × CONSUMO PER KM × MOLTIPLICATORE POLICY

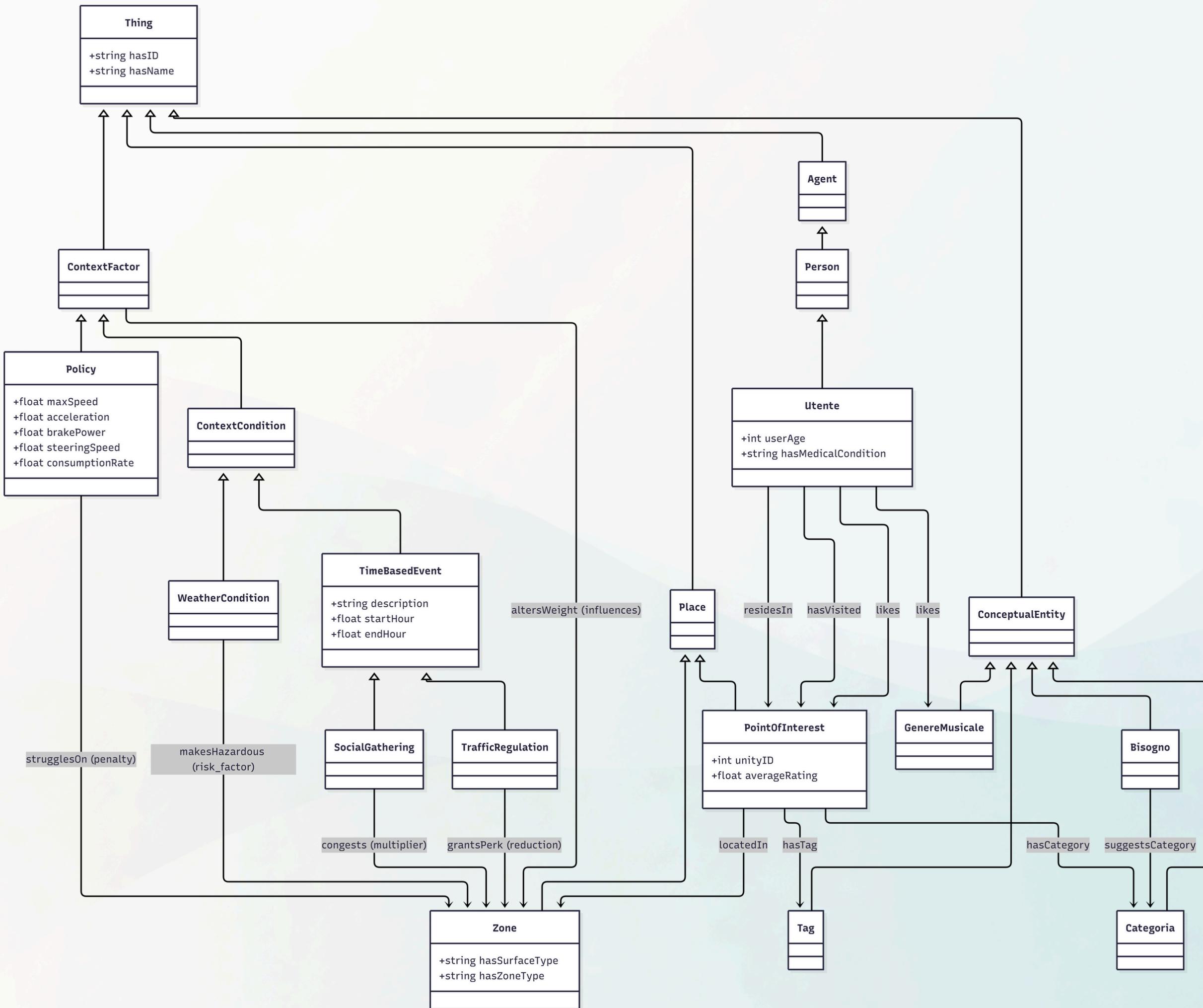
Stime & Sicurezza 🔒

- La **stima** include nella distanza: **Pickup + Corsa + Ritorno alla stazione**
- Per **Pickup e Ritorno alla stazione di ricarica** si usa la policy di guida **ECO**
- **Se la batteria è insufficiente:** si pianifica una **ricarica** prima del Pickup, **auto-switch a Eco, blocco policy e stop di emergenza**

CONSUMO STIMATO = DISTANZA × CONSUMO PER KM × MOLTIPLICATORE POLICY + MARGINE + SOGLIA MINIMA

Ontologia della Base di Conoscenza

L'ontologia OWL definisce il **vocabolario semantico** che poi verrà implementato operativamente con Neo4j.



- **Classi** (rettangoli) → entità del dominio
- **ObjectProperty** (frecce) → relazioni tra entità
- **DataProperty** (attributi) → proprietà dei nodi

Knowledge Graph: neo4j

Perché Neo4j?

- Dati fortemente connessi
- Schema flessibile → evoluzione rapida del dominio
- Ottima integrazione con Python

Da Protégé a Neo4j

- Classi OWL → Nodi
- ObjectProperty → Relazioni
- DataProp → Proprietà
- Individuals → Istanze dei nodi

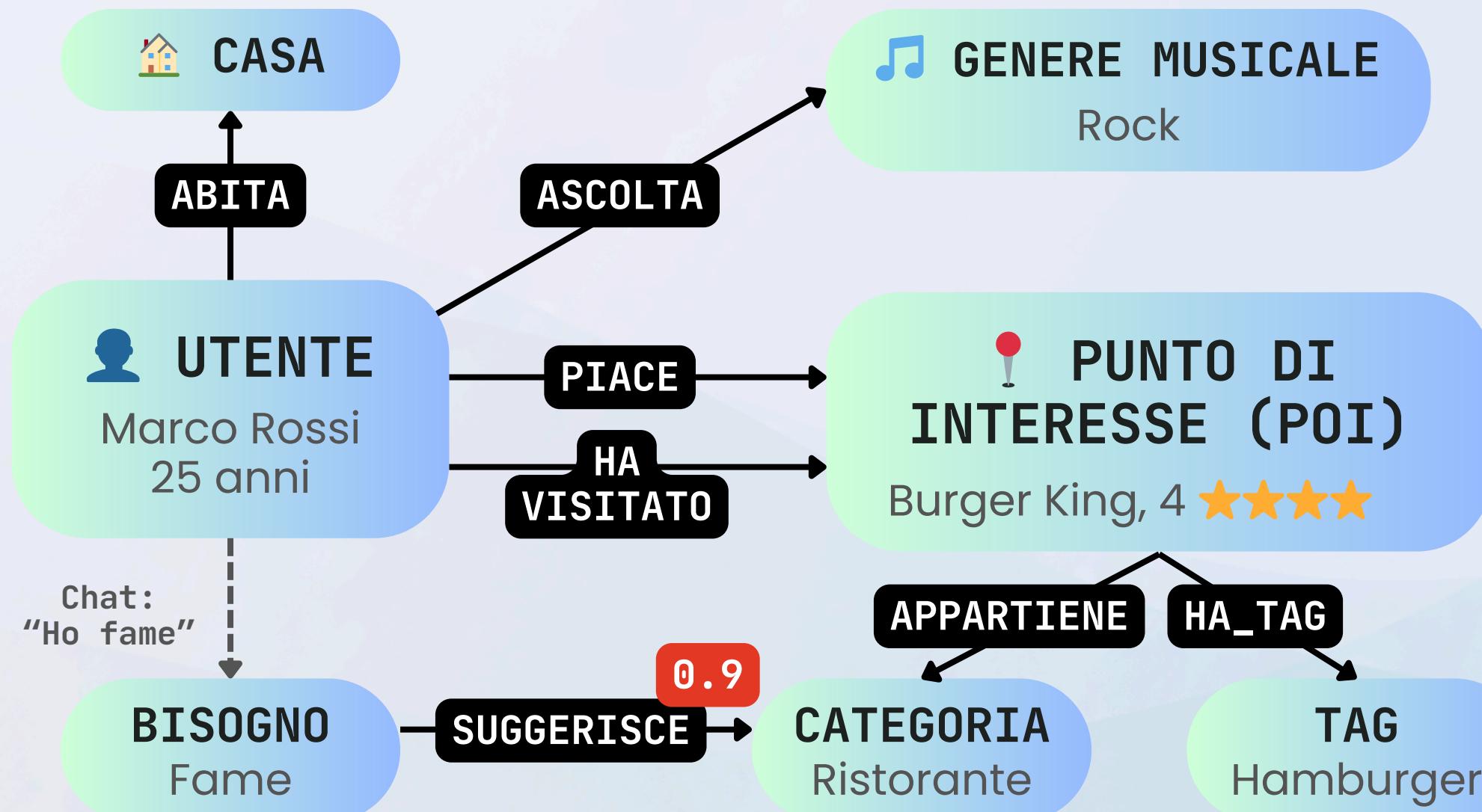
Inferenza

- In Protégé:
 - Inferenza basata su logica descrittiva (owl reasoner)
- In Neo4j:
 - Inferenza applicativa tramite query sul grafo
 - Linguaggio Cypher: query orientate al significato delle relazioni

” L'ontologia definisce il vocabolario concettuale, Neo4j il modello operativo

Knowledge Graph: neo4j

Ricerca del POI e Personalizzazione



+ La relazione “**Piace**” viene inserita dopo **3 visite**

🤔 La relazione “**Suggerisce**” ha un attributo “**Priorità**” che esprime il grado di **compatibilità**:

$$\text{SCORE} = \text{LIKE_BOOST} \times \text{VISIT_BOOST} \times \text{RATING} \times \text{PRIORITY}$$

Policy e Zone



$$\text{MOLTIPLICATORE} = \text{POLICY} \times \text{METEO} \times \text{ZTL} \times \text{TRAFFICO}$$

Chat UI



Prenotazione Intelligente 🧠

- **Selezione destinazione** (POI search o mini chat)
- **Stati live**: in attesa, confermata, taxi in arrivo
- **Gestione coda**: utente sceglie *attendo / annullo*

Chat conversazionale 💬

- **Messaggi**: utente/sistema/explainability
- **Pulsanti rapidi contestuali**

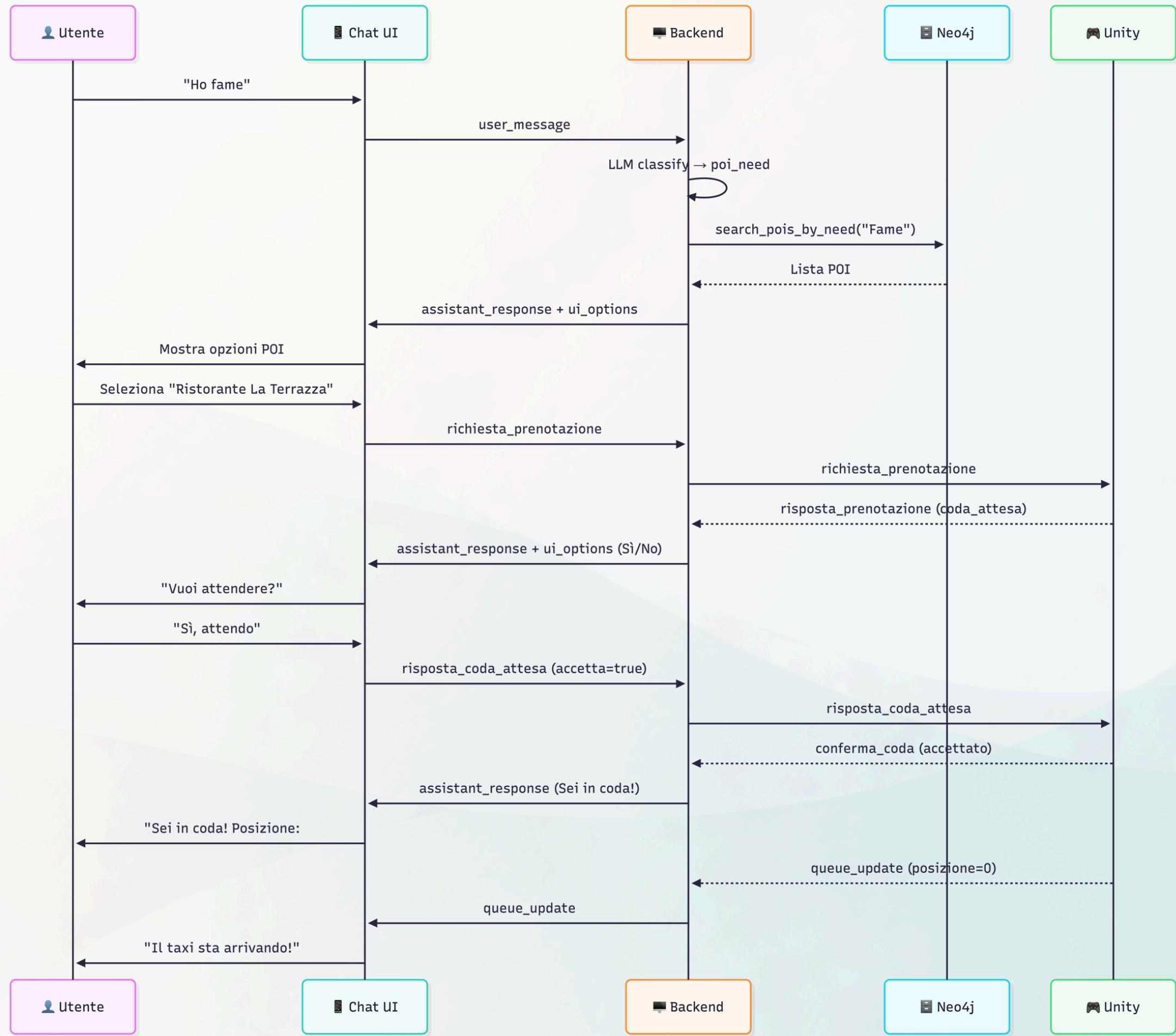
Voce e AI settings 🎵

- **Speech-to-Text + Text-to-Speech**
- **Scelta modello LLM + Voice Settings**

Cosa posso fare in chat ?

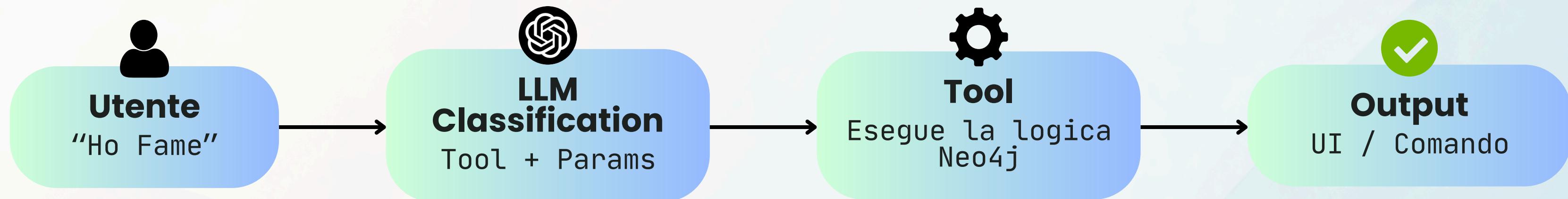
- **Cambio di destinazione**
- **Scelta della policy di guida**
- **Musica** (play/stop/vol)
- **Annnullare la corsa**

Esempio di Sequence Diagram



- Richiesta iniziale:** l'utente esprime un bisogno ("Ho fame"), il backend usa l'LLM per classificarlo e interroga Neo4j per trovare i POI
- Selezione destinazione:** l'utente sceglie un POI dalla lista ordinata in base alle preferenze
- Gestione coda:** Unity risponde che il taxi è occupato e propone l'attesa in coda
- Conferma utente:** L'utente accetta di attendere

LLM & Tooling Conversazionale



POI tools:

- Bisogno
- Tag
- Nome

Music tools:

- Play
- Stop
- Volume

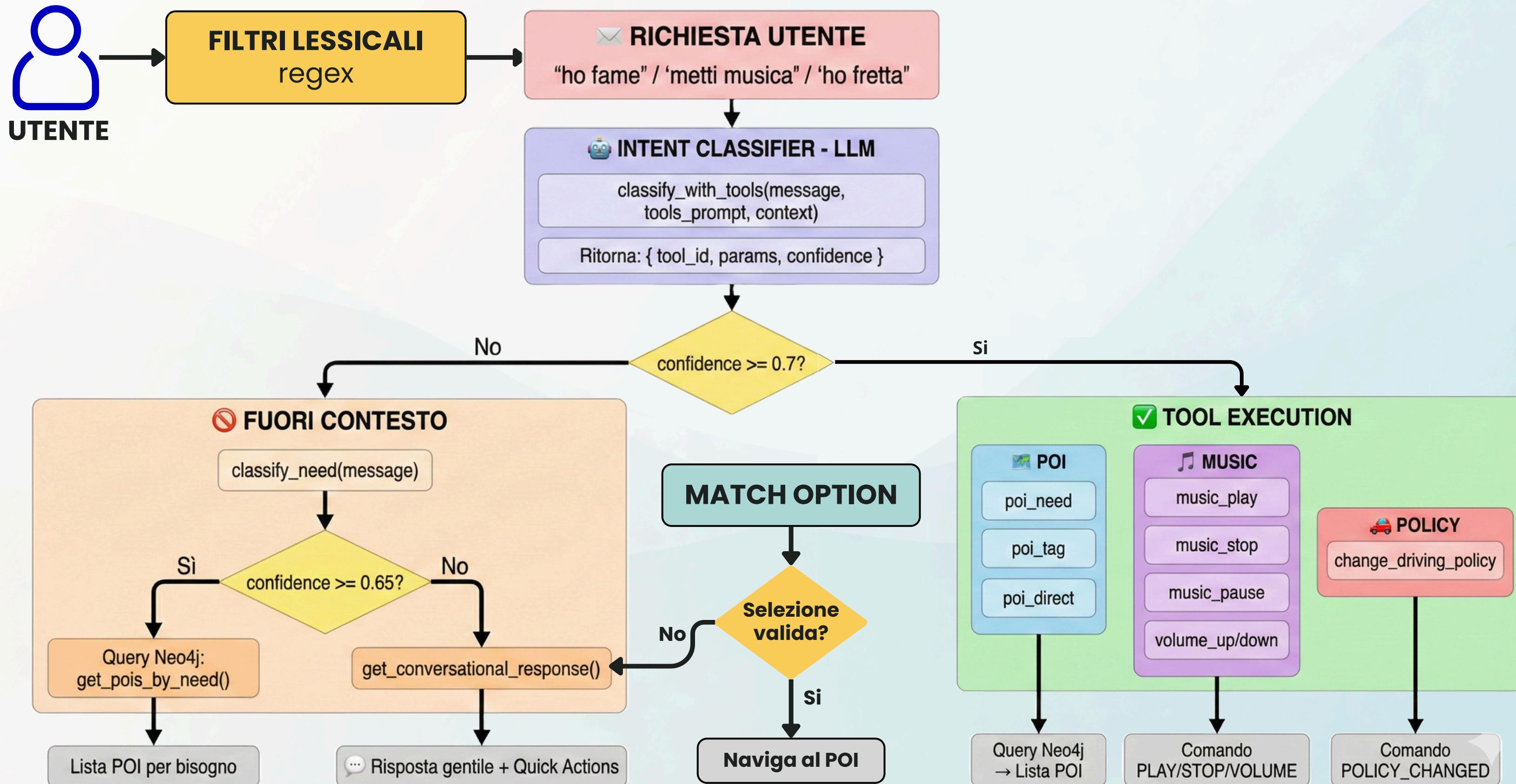
Policy tool:

- Seleziona policy

🔍 Caso d'uso: Classificazione del Bisogno (POI)

- ① L'utente comunica un esigenza in linguaggio naturale:
➢ "Vorrei mangiare qualcosa"
- ② L'LLM classifica il tool **POI_NEED** e il tipo di bisogno **"Fame"**
- ③ Il backend esegue il tool **POI_NEED("Fame")**
- ④ Viene fatta inferenza alla base di conoscenza:
➢ Bisogno → CATEGORIA → POI
Restituisce una lista di POI: Ristorante, Pizzeria, FastFood...
- ⑤ Personalizzazione tramite preferenze utente:
score = like_boost * visit_boost * rating * priority
- ⑥ Ranking e proposta finale

Diagramma di Flusso: Classificazione & Tools



Explainability



- **La chat è anche interfaccia di explainability**
- **Niente black box:**
 - l'LLM serve solo per classificare la richiesta
 - le **azioni sono deterministiche** (regole + A* + KB).
- **Le scelte critiche vengono spiegate** (meteo, policy, barriere, batteria...).

Richiesta fine corsa: mi fermo al punto sicuro più vicino.

🌧 Pioggia in arrivo! Ho ricalcolato il percorso per evitare strade scivolose.

⌚ Nuovo tempo stimato: 13 min

🚧 Ho ricalcolato il percorso per evitare un blocco stradale.

⌚ Nuovo tempo stimato: 17 min

☀️ Meteo migliorato! Ho ricalcolato il percorso per ottimizzare i tempi.

⌚ Nuovo tempo stimato: 13 min



⚠️ Per la tua sicurezza, utilizziamo la modalità **Comfort** ⚡️

La modalità **Comfort** non può essere cambiata per garantire la tua sicurezza.

Batteria insufficiente per la policy attuale.
Passo a ECO per completare la corsa.

⌚ Nuovo tempo stimato: 6 min

✓ Modalità Sport attivata con successo. Il percorso è stato ricalcolato sulla base della policy scelta e potrebbe aver subito variazioni.

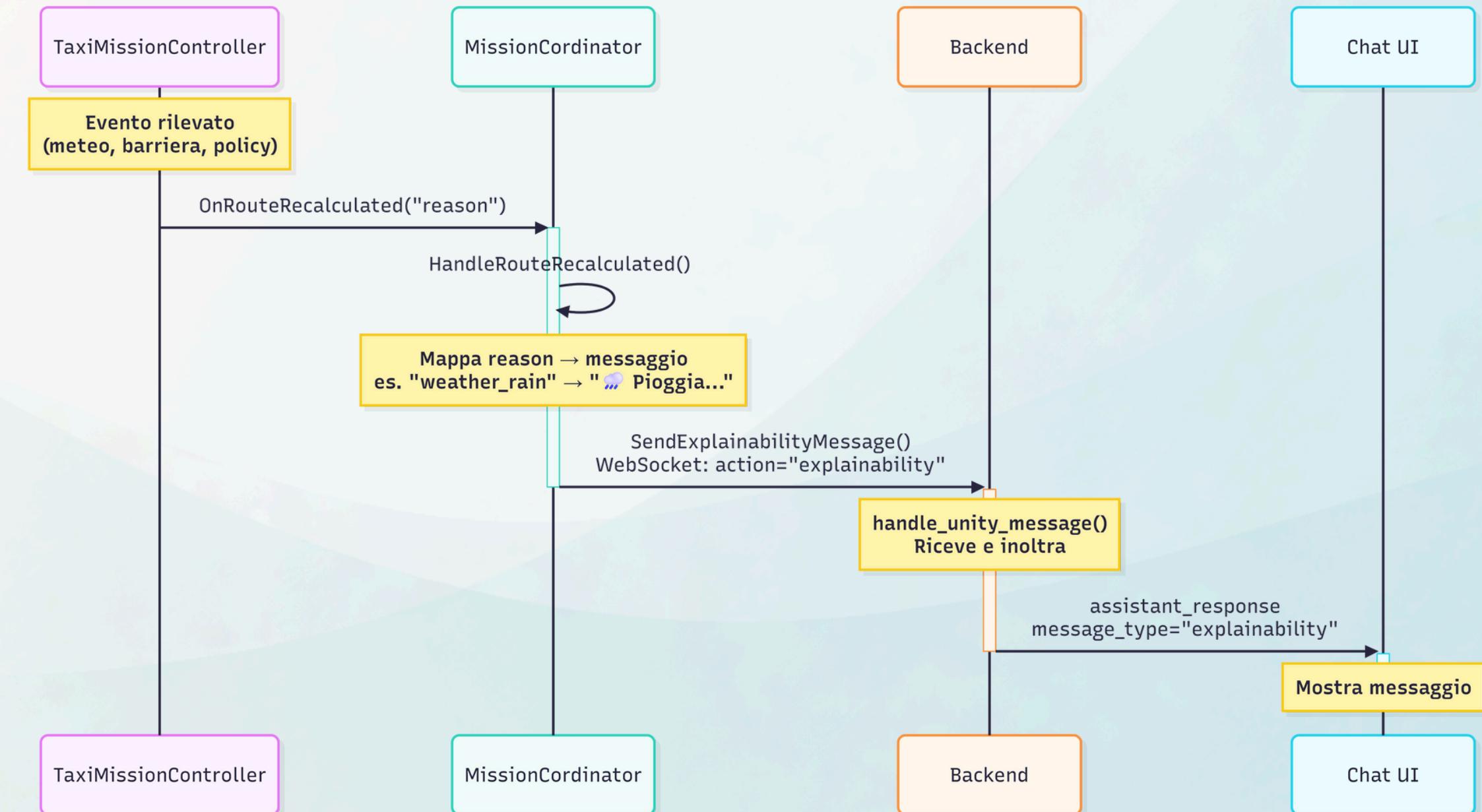
⌚ Nuovo tempo stimato: 7 min

Sequence Diagram Explainability



Caso d'uso: Pioggia + Ricalcolo del percorso

- **Explainability end-to-end:** Unity rileva un **evento** (meteo, barriera, cambio policy)
- **TaxiMissionController** emette il "**ragionamento**"
- **MissionCoordinator** mappa il ragionamento → **messaggio** comprensibile per l'utente
- Il **Backend** riceve via WebSocket il messaggio **action="explainability"** e la inoltra alla chat
- La **Chat UI** riceve **assistant_response** con **message_type="explainability"** " e **mostra il messaggio all'utente**



Grazie per l'attenzione 