Urban Delivery MAS

Decentralized Multi-Agent Coordination for Dynamic, Priority-Aware Last-Mile Logistics

Technical Architecture Report & System Documentation

July 24, 2026

Abstract

This report presents the architectural design, formal cognitive logic, and practical implementation of UrbanDeliveryMAS, a fully decentralized Multi-Agent System (MAS) designed to address dynamic, priority-aware last-mile logistics without reliance on a centralized routing authority or shared global locks. Operating on a discrete 10×10 micro-city grid topology, the system coordinates autonomous BDI (Belief-Desire-Intention) agents—comprising a central Distribution Hub and three delivery drones—negotiating task allocation via an extended Contract-Net Protocol (CNP). Agent cognitive architectures are developed in AgentSpeak(L) and executed using the Jason 3.3.0 interpreter running on Java 21. The system's adaptive capacity is evaluated across three core operational edge cases: (1) proximity-driven spatial task division, (2) belief-triggered intention dropping and re-delegation during critical battery depletion, and (3) real-time task preemption following the mid-transit injection of high-priority Express orders. Experimental validation demonstrates 100% task completion (7/7 orders delivered) with zero unhandled goals or synchronization deadlocks. The results confirm that localized BDI deliberation paired with asynchronous message passing produces robust, self-organizing, and highly adaptive global fleet behavior

Architecture: Jason / AgentSpeak(L) **Environment:** Java 21 **UI:** Swing + Console
Agents: 1 Distribution Hub + 3 Delivery Drones **Interpreter:** Jason 3.3.0 **Status:** Validated (7/7 Completed)

Contents

1 Introduction & Problem Statement

Urban last-mile logistics represents one of the most volatile and computationally complex domains within modern dynamic transport management. Occupying the final leg of the supply chain—where goods are transferred from regional distribution hubs to individual customer locations—last-mile operations directly dictate both operational expenditure and service quality. However, urban networks operate under perpetual environmental stochasticity. Fleet operators face continuous real-time perturbations, including dynamically arriving customer orders, shifting priority tiers (e.g., standard vs. urgent express deliveries), fluctuating traffic congestion, battery charge degradation, and localized hardware unexpected failures.

In contemporary logistics infrastructure, dispatching and routing are traditionally governed by **centralized optimization paradigms**[cite: 1]. These systems formulate the environment as a Dynamic Vehicle Routing Problem (DVRP) and solve it using centralized Exact Methods (e.g., Mixed-Integer Linear Programming, Branch-and-Cut) or meta-heuristics (e.g., Genetic Algorithms, Simulated Annealing, Large Neighborhood Search)[cite: 1]. While centralized optimization can synthesize mathematically near-optimal routes under static, predictable conditions, it introduces severe architectural bottlenecks when deployed in real-time, highly dynamic environments[cite: 1]:

1. **Single Point of Failure & Vulnerability:** Centralized control topologies exhibit absolute structural dependence on the core coordinator[cite: 1]. A hardware crash, network partition, or computational stall at the central server instantly paralyzes fleet-wide decision-making, leaving edge vehicles unable to adapt[cite: 1].
2. **Computational Inflexibility & Scaling Complexity:** Centralized solvers operate on global state space representations[cite: 1]. Any runtime state perturbation—such as a single drone experiencing sudden battery drain or a new high-priority order entering the queue—forces the central engine to re-evaluate or solve the entire combinatorial optimization problem[cite: 1]. As fleet sizes and order densities scale, the computational overhead expands exponentially, rendering real-time route re-calculation intractable[cite: 1].
3. **High Telemetry Overhead & Network Saturation:** Maintaining an up-to-date global state requires continuous, high-frequency telemetry streaming from every mobile node to the central server[cite: 1]. In dense urban environments, this constant communication loop consumes substantial wireless bandwidth, increases latency, and degrades gracefully whenever network coverage drops[cite: 1].
4. **Rigidity Against Unmodeled Local Disturbances:** Centralized algorithms struggle to accommodate immediate, reactive self-preservation behaviors (e.g., a vehicle aborting a task locally to seek immediate emergency power) without requiring a full system-level plan invalidation[cite: 1].

To overcome these structural limitations, this project introduces `UrbanDeliveryMAS`, a **decentralized Multi-Agent System (MAS)** designed to shift the computational paradigm from centralized route calculation to distributed cognitive deliberation. Rather than treating individual vehicles as passive execution units directed by a master solver, `UrbanDeliveryMAS` models every vehicle as an autonomous Belief-Desire-Intention (BDI) agent capable of localized reasoning, dynamic plan revision, and peer-to-peer task negotiation.

By distributing the deliberation space across individual nodes, global fleet coordination emerges organically through lightweight, message-passing protocols—specifically an extended Contract-Net Protocol (CNP)—rather than high-overhead global route optimization. This structural shift guarantees that path selection, task commitment, resource monitoring, and emergency recovery remain strictly localized, enabling graceful degradation and immediate reactivity in the face of dynamic environmental perturbations[cite: 1].

1.1 Research Question & Methodological Focus

The primary objective of this research is to evaluate the viability and performance boundaries of decentralized cognitive deliberation in dynamic logistical environments[cite: 1]. Specifically, this project investigates the central research question:

> *Can purely localized BDI cognitive reasoning, coupled with an asynchronous message-passing negotiation protocol, produce coherent, resilient, and priority-aware global vehicle routing without relying on a centralized route optimizer or shared global synchronization lock?*[cite: 1]

To rigorously evaluate this overarching question, the system implementation and experimental framework address three critical sub-questions:

1. **Emergent Spatial Efficiency & Decentralized Allocation:** Can localized cost-heuristic bidding within an asynchronous Contract-Net Protocol (CNP) produce efficient, proximity-aware spatial task partitioning across an autonomous fleet without a centralized route planner computing global vehicle trajectories? Specifically, does individual cost-estimation—grounded in Manhattan distance metrics and real-time battery state—naturally result in optimal task division without global synchronization locks?
2. **Cognitive Fault Recovery via Intention Revision:** How effectively can belief-triggered intention dropping enable an autonomous agent to reactively suspend an active commitment during runtime perturbations (e.g., critical energy depletion)? Can the agent autonomously abort a physical delivery, safely re-delegate the payload back to the Distribution Hub, and reorganize its internal goal hierarchy to prioritize self-preservation (recharging) without inducing systemic deadlocks, race conditions, or order loss?
3. **Real-Time Priority Preemption & Context-Sensitive Plan Selection:** Can localized AgentSpeak(L) context guards dynamically resolve conflicts between active commitments and high-priority Express order injections? Specifically, can a busy agent evaluate whether to preempt an ongoing standard delivery mid-transit to serve an urgent order, while ensuring that lower-priority tasks are safely released for re-allocation rather than abandoned?

1.2 System Objectives

To address this research question and evaluate the proposed decentralized architecture, `UrbanDeliveryMAS` achieves the following primary engineering and theoretical objectives:

- **Decentralized Task Allocation via Asynchronous Contract-Net Protocol (CNP):** Design and implement a dynamic market-based negotiation protocol using AgentSpeak achievement goals (`achieve`). All task announcements, bid submissions, and award evaluations must operate without global locks or shared memory, ensuring task bidding and route execution remain strictly local to individual agents.
- **Explicit Cognitive BDI Reasoning & Dynamic Plan Selection:** Construct modular, human-readable AgentSpeak(L) cognitive scripts where internal belief updates directly drive goal adoption, plan suspension, or reactive intention drops. The agent logic must leverage context guards to dynamically evaluate candidate plans based on localized cost heuristics (combining Manhattan distance and battery reserves)[cite: 1].
- **Dynamic Fault Resilience & Emergency Recovery:** Demonstrate system-level fault tolerance against unexpected runtime disturbances[cite: 1]. Specifically, the architecture must support:

(a) *Resource Criticality Recovery:* Reactive intention dropping upon sudden battery depletion ($\leq 20\%$), enabling safe package aborts, re-delegation back to the Distribution Hub, and primary goal shifts toward recharging[cite: 1].

(b) *Mid-Transit Priority Preemption:* Real-time suspension of ongoing standard deliveries to accommodate time-critical Express orders without abandoning lower-priority work[cite: 1].

- **Decoupled Architectural Abstraction:** Maintain strict separation of concerns across three structural layers[cite: 1]:

  (a) The *Cognitive Reasoning Layer* (Jason / AgentSpeak agents holding BDI states)[cite: 1].

  (b) The *Environment Physics Layer* (a custom Java API executing grid actions, managing state transitions, and publishing percepts)[cite: 1].

  (c) The *View Layer* (a minimalist Swing GUI rendering physical grid states without making deliberation decisions)[cite: 1].

- **Empirical System Validation & Zero Task Loss:** Validate full system execution on the Jason 3.3.0 interpreter (Java 21) across all operational scenarios, ensuring error-free execution, zero unhandled goals, clean MAS self-termination, and a 100% task delivery completion rate[cite: 1].

To overcome these constraints, this project presents a **decentralized Multi-Agent System (MAS)** where route planning and task selection are completely localized. Agents operate under the **Belief-Desire-Intention (BDI)** model, continually perceiving environment updates, forming internal beliefs, establishing localized goals (desires), and executing or abandoning committed plans (intentions) in direct response to environmental perturbations.

1.3 Core Research Question

This project directly investigates: *Can purely localized BDI reasoning, coupled with asynchronous message-passing contract negotiation, yield coherent, highly efficient, and resilient global vehicle coordination without any central route planning algorithm or global synchronization lock?*

1.4 System Objectives

To address this research question, the functional and theoretical objectives of this MAS framework include:

- **Decentralized Task Allocation:** Implementation of an asynchronous Contract-Net Protocol (CNP) where task bidding, evaluation, and execution decisions are strictly local to individual agents, operating without global locks or shared state space[cite: 1].

- **Explicit Cognitive BDI Reasoning:** Construction of readable, explicit AgentSpeak(L) scripts where internal belief updates dynamically trigger plan adoption, suspension, or reactive intention drops based on localized context guards[cite: 1].

- **Dynamic Adaptability & Fault Resilience:** Guaranteed system-level resilience against unexpected runtime perturbations, specifically enabling reactive self-preservation during sudden battery depletion and dynamic task preemption upon mid-transit Express order injections[cite: 1].

- **Decoupled Architectural Abstraction:** Strict modular separation between the internal cognitive reasoning layer (Jason 3.3.0 / AgentSpeak), the physical simulation environment dynamics (Java 21 Environment API), and the real-time visual interface (Swing GUI)[cite: 1].
- **Empirical System Validation:** Comprehensive execution testing across edge-case operational scenarios to verify clean runtime termination, zero unhandled goals, and a 100% task completion rate[cite: 1].

—

2 Theoretical Background

2.1 The Belief-Desire-Intention (BDI) Paradigm

The Belief-Desire-Intention (BDI) architecture, originally formulated by Michael Bratman as a philosophical model of human practical reasoning and later operationalized for computer science by Rao and Georgeff, provides a formal cognitive model for autonomous agents operating in dynamic, uncertain environments. The paradigm structures an agent's internal mental state into three core abstractions:

- **Beliefs (B):** The agent's informational state, representing its current, subjective knowledge about itself, peer agents, and the external environment (e.g., grid coordinates, battery state, active payloads, and external percepts).
- **Desires (D):** The agent's motivational state, encapsulating all prospective states of affairs or operational goals the agent would ideally like to bring about (e.g., fulfilling an order, maintaining safe battery reserves).
- **Intentions (I):** The agent's deliberative state, representing the specific subset of desires to which the agent has committed and is actively pursuing through concrete procedural plans.

The core operational mechanism of a BDI agent is its continuous **practical reasoning cycle**. The agent perceives its environment, updates its belief base B, checks which procedural plans are applicable given current context guards, commits to an intention I, and executes atomic actions. Crucially, BDI architectures decouple goal selection from plan execution: if an asynchronous belief update invalidates an active context guard (e.g., sudden battery depletion), the agent can reactively drop, suspend, or revise committed intentions without requiring global fleet re-synchronization.

2.2 AgentSpeak(L) Logic & Plan Architecture

AgentSpeak(L) is a formal logic-based programming language that operationalizes the BDI paradigm. An AgentSpeak program consists of a set of initial beliefs and a library of procedural plans. A plan is formally defined as:

$$\delta : \chi \leftarrow \psi. \tag{1}$$

where:

- δ is the **triggering event**, denoting belief updates (`+b`, `-b`) or goal adoption/achievement events (`+!g`, `-!g`).

- χ is the **context guard**, a logical predicate evaluated against the current belief base B to determine if the plan is applicable.
- ψ is the **plan body**, a sequence of mental actions, belief modifications, external environment actions, or sub-goals.

2.3 The Jason Execution Engine

`UrbanDeliveryMAS` leverages **Jason 3.3.0**, an open-source, Java-based execution engine and interpreter for AgentSpeak(L). Jason extends standard theoretical AgentSpeak(L) by introducing practical multi-agent programming mechanisms, including internal actions (e.g., `.send`, `.broadcast`, `.drop_intention`), speech-act performatives (e.g., `achieve`, `tell`, `askOne`), customizable plan selection functions, and a seamless interface binding to custom Java environment APIs via the `jason.asSyntax` package.

In Jason, an agent's practical reasoning cycle operates as an continuous execution loop. At each cycle, the interpreter processes mental state transitions through six discrete operational phases:

1. **Perception & Belief Update:** The interpreter senses changes in the external Java environment, updates internal percepts, and modifies the belief base B by generating corresponding belief addition (`+b`) or belief deletion (`-b`) events.
2. **Mailbox Management & Speech-Act Processing:** The agent consumes asynchronous incoming messages sent by peer agents via performative calls (such as Contract-Net Protocol bids or assignment requests). Messages are converted into belief update events or goal adoption requests.
3. **Event Selection ($\mathbf{S_E}$):** An event selection function picks a single event from the agent's internal event queue for deliberation.
4. **Applicable Plan Retrieval ($\mathbf{S_P}$):** The interpreter queries the agent's plan library to identify relevant plans whose triggering events match the selected event. It then evaluates their context guards χ against the current belief base B to construct a set of applicable plans.
5. **Intention Commitment ($\mathbf{S_I}$):** The agent selects an applicable plan and either creates a new top-level intention stack or appends the plan as a sub-goal onto an existing committed intention stack I.
6. **Atomic Execution Step:** The interpreter pops and executes the top action on the selected intention stack. If the action is an environment action (e.g., `move`, `pickup`), it is dispatched to the Java environment API for physical execution.

Crucially, Jason's architecture allows internal actions like `.drop_intention(G)` to directly manipulate the active intention stack during Phase 6. This capability enables `UrbanDeliveryMAS` agents to dynamically interrupt running plans, abort physical tasks, and reorganize goal priorities in response to runtime disturbances without stalling the interpreter loop.

2.4 Asynchronous Contract-Net Protocol (CNP)

Task allocation across the fleet is governed by an asynchronous **Contract-Net Protocol (CNP)**, a market-based negotiation protocol for multi-agent task distribution. Instead of a master scheduler dictating vehicle trajectories, task allocation follows a four-stage conversational performative structure:

1. **Call for Proposals (CFP):** The Distribution Hub broadcasts a task request (`cfp(Id, X, Y, Priority)`) to all available drones.

2. **Bid Evaluation:** Each drone calculates a localized bid cost locally based on its Manhattan distance to the task coordinates and its current battery reserve.

3. **Award / Refusal:** Drones send proposals back to the Hub via performatives (`bid(Id, Cost)`). The Hub evaluates all incoming bids and awards the contract to the lowest-cost bidder (`accept_proposal`).

4. **Task Commitment:** The winning drone adopts the task as a primary achievement goal (`+!deliver(Id, X, Y)`), while losing drones clear local bidding states.

2.5 AgentSpeak(L) and the Jason Interpreter

AgentSpeak(L) is a formal logic-based programming language designed to operationalize the BDI architecture. An AgentSpeak program consists of a set of initial beliefs and a set of procedural plans written in the syntax:

$$\text{triggering\_event} : \text{context} \leftarrow \text{body}. \tag{2}$$

Where:

- `triggering_event` specifies the belief modification (`+b`, `-b`) or goal addition/deletion (`+!g`, `-!g`) that activates the plan.
- `context` is a logical predicate that must evaluate to true against the agent's current belief base for the plan to be applicable.
- `body` is a sequential sequence of basic actions, internal actions, or sub-goals.

This project utilizes **Jason 3.3.0** (running on Java 21), a robust interpreter that extends AgentSpeak(L) by offering performative-based message passing (e.g., `.send`, `.broadcast`), customizable plan selection functions, atomic execution blocks, and seamless integration with custom Java environment interfaces.

2.6 The Contract-Net Protocol (CNP)

The Contract-Net Protocol (Smith, 1980) is a market-inspired coordination mechanism for decentralized task allocation among autonomous nodes. It operates across four distinct phases:

1. **Call for Proposals (CFP):** A manager node announces an open task, detailing parameters such as task location and priority tier.

2. **Bid Submission:** Contractor agents independently compute localized execution costs based on their private state (e.g., proximity, available energy, active workload) and return a bid score.

3. **Bid Evaluation & Awarding:** The manager agent collects bids, evaluates them against an objective function (e.g., minimizing cost), and sends an award message to the best contractor while sending rejection notices to others.

4. **Task Execution:** The winning contractor commits the task to its local intention structure and executes the delivery.

—

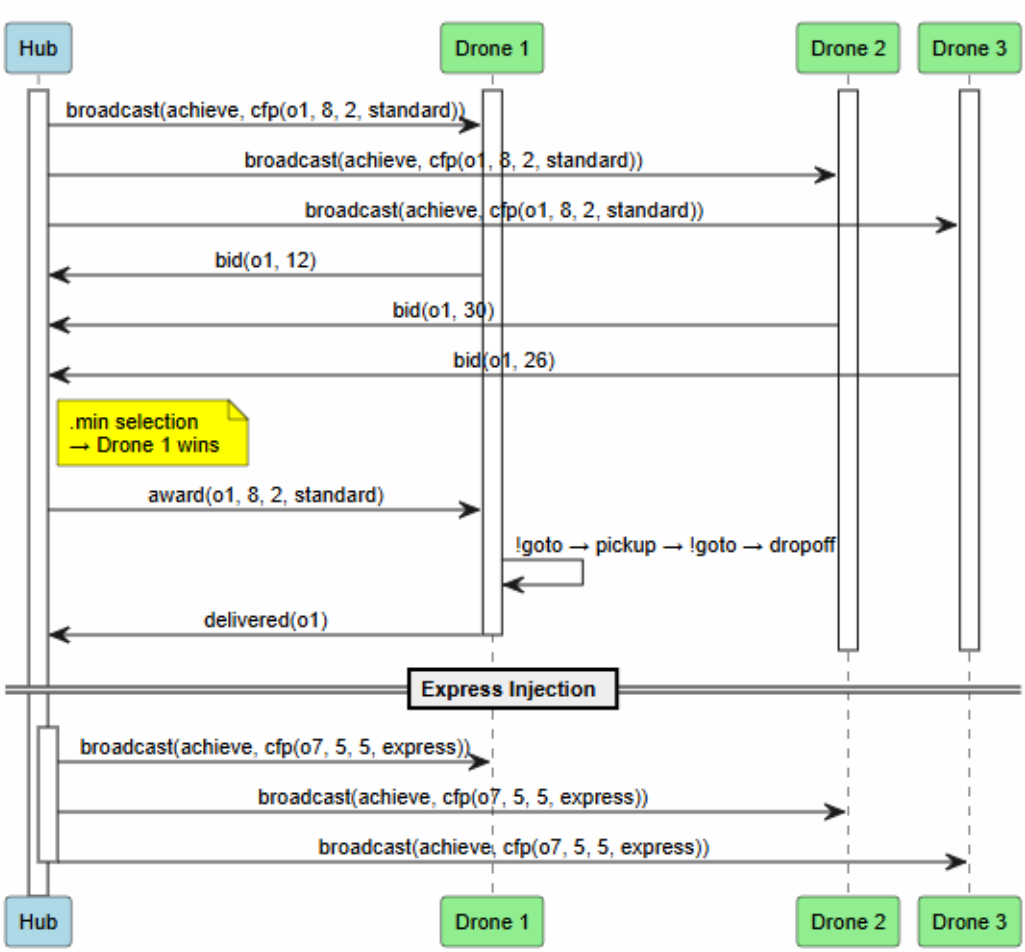

Figure 1: System Architecture Diagram

3 System Architecture & Environment Design

The system architecture is structured into three clean, decoupled layers, ensuring strict separation of concerns between agent reasoning, environment physics, and visual rendering.

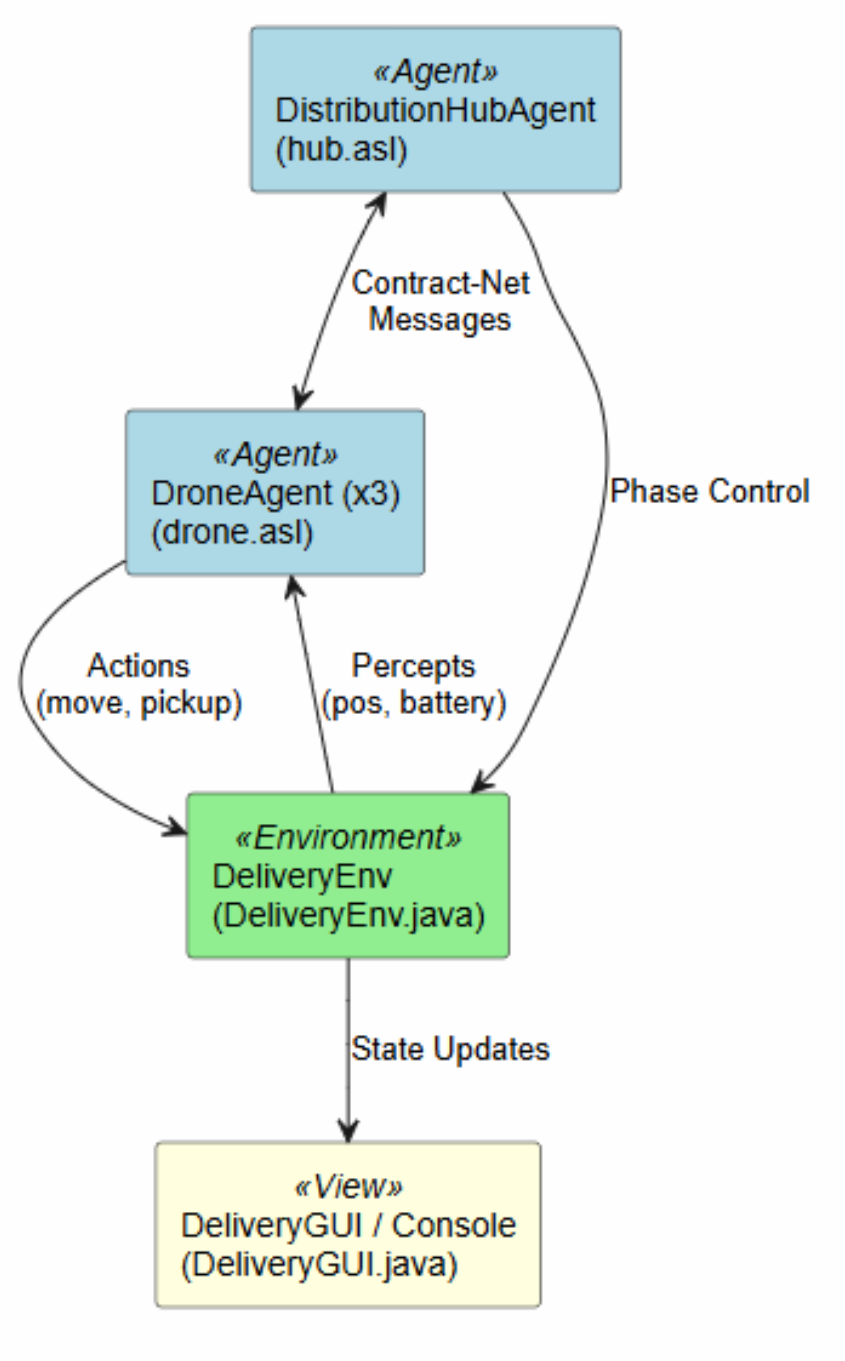

Figure 2: Contract-Net Sequence Diagram.

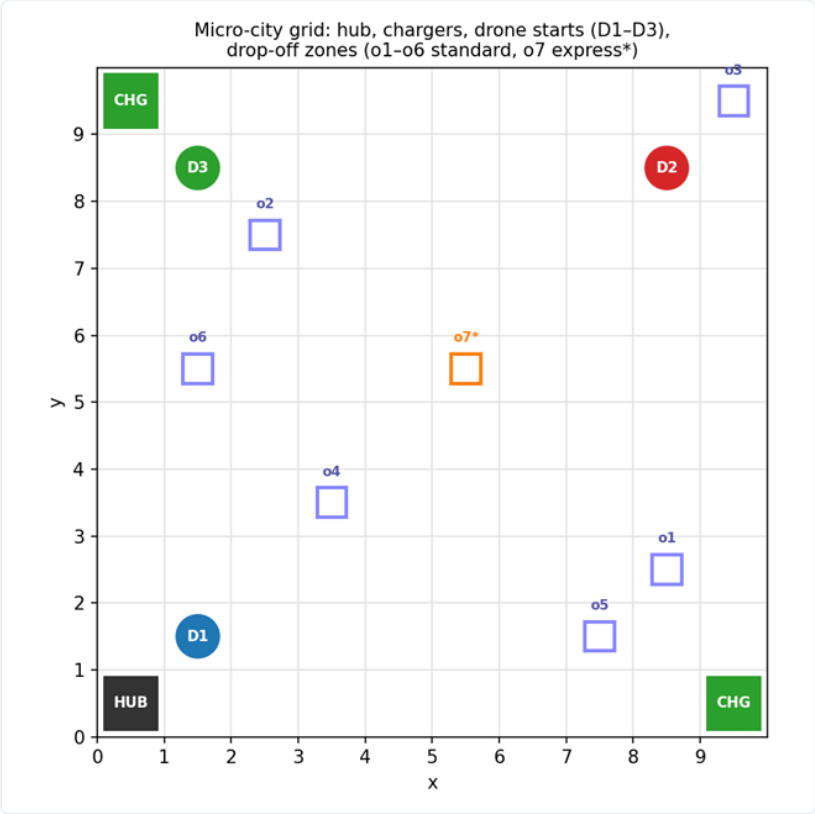

Figure 3: The 10×10 micro-city. Hub at (0,0); charging docks at (9,0) and (0,9); drone start depots D1–D3; standard drop-offs o1–o6 and the injected Express order o7 (orange).

Table 1: System Architecture Component Responsibilities

Component	Instances	Responsibility
`DistributionHubAgent` (`hub.asl`)	1	Broadcasts CFPs, collects and evaluates agent bids, awards tasks, triggers phase progressions, and manages re-delegation. Maintains zero global control over drone actions.
`DroneAgent` (`drone.asl`)	3 (`drone1..3`)	Computes context-sensitive bids based on local state, navigates grid, manages battery cycles, handles task preemption, and reactively drops intentions when critical.
`DeliveryEnv` (`DeliveryEnv.java`)	1	Maintains physical grid representation (10×10), executes atomic agent actions, publishes updated percepts, and introduces dynamic disturbances.
`DeliveryGUI` (`DeliveryGUI.java`)	1	Real-time minimalist 2D Swing visualization showing spatial grid positions, drone paths, battery indicators, and drop-off zone states.

3.1 The Micro-City Grid Representation

The physical environment is modeled as a discrete 10×10 coordinate grid ($X, Y \in [0, 9]$). Key topological coordinates are structured as follows:

- **Central Hub:** Positioned at $(0, 0)$, acting as the central collection point for all packages.

- **Charging Docks (`CHG`):** Located at opposite grid corners $(9, 0)$ and $(0, 9)$ to test spatial energy planning.
- **Drone Initial Depots:** `drone1` at $(1, 1)$, `drone2` at $(8, 8)$, and `drone3` at $(1, 8)$.
- **Drop-Off Zones ($o1$–$o6$):** Standard priority locations scattered across the grid.
- **Express Drop-Off Zone ($o7^*$):** High-priority injected task location positioned at $(5, 5)$.

3.2 Environment Action API & Percept Model

Agents do not access grid data structures directly. Instead, they interact via atomic action execution calls provided by the Java `DeliveryEnv` class, which in turn returns localized percepts.

Table 2: Environment Actions and State Effects

Environment Action	Execution Effect & State Transition
`move(X, Y)`	Moves the drone one Manhattan step toward (X, Y). Consumes battery power proportional to distance. Triggers battery drain percept update.
`pickup(Id)`	Validates drone position at Hub $(0, 0)$. Binds package `Id` to drone inventory state. Updates percept `carrying(Id)`.
`dropoff(Id, X, Y)`	Validates drone location at target (X, Y) while carrying `Id`. Updates task status to delivered and clears inventory state.
`abort(Id)`	Immediately drops package `Id` at current coordinates without completing delivery; clears local carrying state. Used during emergencies.
`charge`	Increases drone battery charge by a fixed rate (`CHARGE_RATE`) up to 100% when positioned at a valid charging dock.
`arm_drain`	Administrative environment call. Arms a one-shot sudden battery failure event triggered on the next task execution step.

Percepts published to individual drones at every simulation cycle include:

- `pos(X, Y)`: Current grid coordinates of the agent.
- `battery(B)`: Remaining battery percentage integer ($0 \leq B \leq 100$).
- `carrying(Id)`: Currently held package identifier (omitted if inventory is empty).

—

4 Agent Cognitive Design & AgentSpeak Implementation

4.1 Drone Agent Architecture (`drone.asl`)

The drone agent represents the core autonomous actor within the MAS. Its cognitive loop is structured around clear BDI mapping:

Table 3: Mapping of Cognitive BDI Concepts in `drone.asl`

BDI Concept	**Realization in `drone.asl`**
Beliefs (Perceived)	Dynamic percepts dynamically updated by environment: `pos(X,Y)`, `battery(B)`, `carrying(Id)`.
Beliefs (Internal)	Static world topology: `hub(0,0)`, `charger(9,0)`, `charger(0,9)`, `critical(20)`. Dynamic operational records: `assigned(Id)`, `busy`, `charging`.
Desires	Maintain operational battery above safety thresholds; fulfill awarded package deliveries; maximize individual revenue/utility by bidding accurately.
Intentions	Active procedural plans executed sequentially: `!deliver`, `!goto`, `!recharge`, `!top_up`.

4.1.1 Context-Sensitive Bidding Logic

When the Distribution Hub broadcasts a Call for Proposals (`cfp`), individual drones evaluate four ordered contextual plans. Jason's plan selection engine evaluates these sequentially, executing the first plan whose guard context evaluates to true.

The bidding strategy is formally defined as follows:

$$\text{Bid Cost} = \begin{cases} 99999 & \text{if } B \leq T \text{ (Low Battery / Safe Mode)} \\ D(P_{\text{curr}}, P_{\text{hub}}) + D(P_{\text{hub}}, P_{\text{target}}) + (100 - B) + 30 & \text{if Busy and Task is Express} \\ 99999 & \text{if Busy and Task is Standard} \\ D(P_{\text{curr}}, P_{\text{hub}}) + D(P_{\text{hub}}, P_{\text{target}}) + (100 - B) & \text{if Idle and Healthy} \end{cases} \tag{3}$$

where $D(P_1, P_2) = |X_1 - X_2| + |Y_1 - Y_2|$ represents the Manhattan distance between two grid points, B is the current battery level, and T is the critical threshold (20%).

```
// (1) Free but low battery -> Refuse via prohibitive bid
+!cfp(Id, _, _, _) [source(hub)]
   : not busy & battery(B) & critical(T) & (B <= T)
  <- .send(hub, achieve, bid(Id, 99999)).

// (2) Busy + EXPRESS Order -> Bid with drop penalty (+30) to allow preemption
    if necessary
+!cfp(Id, Dx, Dy, express) [source(hub)]
   : busy & pos(X,Y) & battery(B) & hub(HX,HY)
  <- ?dist(X, Y, HX, HY, D1);
     ?dist(HX, HY, Dx, Dy, D2);
     Cost = D1 + D2 + (100 - B) + 30;
     .send(hub, achieve, bid(Id, Cost)).

// (3) Busy + STANDARD Order -> Refuse (Never abandon work for equal priority)
+!cfp(Id, _, _, standard) [source(hub)]
   : busy
  <- .send(hub, achieve, bid(Id, 99999)).

// (4) Free & Healthy -> Proximity and energy based competitive proposal
+!cfp(Id, Dx, Dy, _) [source(hub)]
   : not busy & pos(X,Y) & battery(B) & hub(HX,HY)
  <- ?dist(X, Y, HX, HY, D1);
     ?dist(HX, HY, Dx, Dy, D2);
     Cost = D1 + D2 + (100 - B);
     .send(hub, achieve, bid(Id, Cost)).
```

Listing 1: Context-sensitive bidding logic in `drone.asl`

4.1.2 Delivery Intention Execution & Recursive Navigation

When a drone receives an `award` message from the Hub, it establishes an intention to deliver the package. Navigation across the grid is handled recursively via step-by-step goal achievements, allowing the agent to evaluate percept modifications at every tile transition.

```
// Core Delivery Plan
+!deliver(Id, Dx, Dy, Prio) : hub(HX, HY)
  <- +busy;
     !goto(HX, HY);
     pickup(Id);
     !goto(Dx, Dy);
     dropoff(Id, Dx, Dy);
     .send(hub, achieve, delivered(Id));
     -assigned(Id);
     -busy.

// Recursive Step-by-Step Navigation
+!goto(X, Y) : pos(X, Y) <- true. // Base Case: Destination Reached

+!goto(X, Y) : true
  <- move(X, Y);
     !goto(X, Y). // Recursive Step
```

Listing 2: Delivery procedural plan and recursive navigation

4.1.3 Reactive Intention Drop (Resource Criticality Plan)

The defining feature of pure BDI execution within this agent is the reactive intention drop plan. If a drone's battery level drops below the critical threshold (20%) while actively carrying a payload, a belief-update event triggers a plan that abruptly terminates the active delivery intention, aborts the physical item, re-delegates the task back to the Hub, and adopts a new primary intention to recharge.

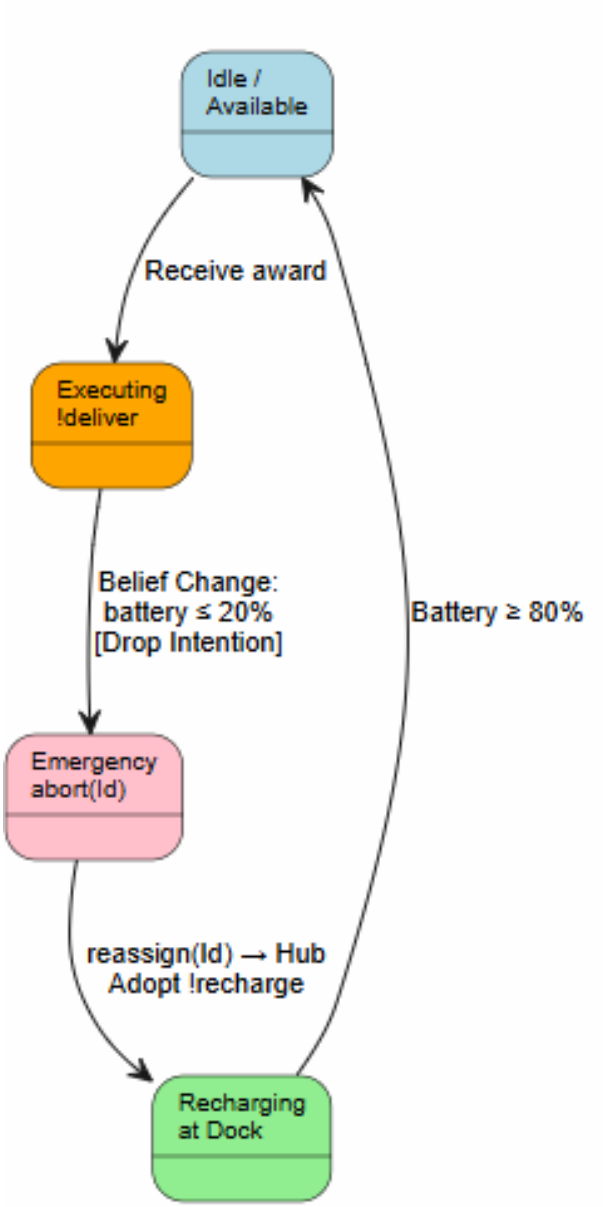

Figure 4: Reactive Intention Drop (BDI State Transition Diagram)

```
+battery(B) : critical(T) & (B <= T) & carrying(Id) & not charging
  <- +charging;
     .drop_intention(deliver(Id, _, _, _)); // REACTIVELY DROP ACTIVE INTENTION
     abort(Id);                             // Release physical payload
     .send(hub, achieve, reassign(Id));      // Notify Hub to re-allocate
     -assigned(Id);
     -busy;
     !recharge.                             // Adopt NEW primary intention
```

Listing 3: Reactive resource criticality plan demonstrating belief-driven intention dropping

4.2 Distribution Hub Agent Architecture (`hub.asl`)

The Distribution Hub coordinates task distribution across the multi-agent network without acting as a centralized route optimizer. It manages task states via five internal predicates: `task/4`, `awaiting/1`, `bidrec/3`, `done/1`, and `ndrones/1`.

4.2.1 Atomic Task Awarding Mechanism

Because bids arrive asynchronously from multiple drones, the Hub's bid evaluation plan must be atomic. By annotating the plan with the `[atomic]` directive, the Jason engine guarantees that concurrent bid assertions do not trigger race conditions or cause double-allocation of a single order.

```
@[atomic]
+!try_award(Id) : ndrones(N)
  <- .count(bidrec(Id, _, _), C);
     if (C >= N) { // All drones have submitted bids
       .findall(cost(K, Dr), bidrec(Id, Dr, K), L);
       .abolish(bidrec(Id, _, _));
       .min(L, cost(BestCost, Winner));
       if (BestCost < 99999) {
         -awaiting(Id);
         ?task(Id, Dx, Dy, Prio);
         .send(Winner, achieve, award(Id, Dx, Dy, Prio));
       } else {
         // All agents busy or impaired -> Retry negotiation later
         !retry(Id);
       }
     }.
```

Listing 4: Atomic award evaluation plan in `hub.asl`

—

5 Simulation Scenarios & Experimental Validation

The system was empirically evaluated across three distinct operational scenarios designed to stress-test decentralized task allocation, energy-critical fault recovery, and priority preemption under dynamic conditions.

5.1 Scenario 1: Proximity-Based Task Division

Goal: Validate that the asynchronous Contract-Net Protocol (CNP) naturally achieves spatial task partitioning across the drone fleet based strictly on localized cost heuristics, eliminating the need for a central trajectory planner.

5.1.1 Operational Execution Flow

1. **Call for Proposals (CFP):** The Distribution Hub broadcasts announcement messages for two standard delivery orders: task o_1 with destination coordinates $(8, 2)$ and task o_2 with destination coordinates $(2, 7)$.

2. **Localized Cost Evaluation:** Upon receiving the `cfp`, each idle drone calculates its bid cost C using its local belief state (B):

$$C = D(P_{\text{curr}}, P_{\text{hub}}) + D(P_{\text{hub}}, P_{\text{target}}) + (100 - B)[cite : 1] \tag{4}$$

   where $D(P_a, P_b) = |X_a - X_b| + |Y_a - Y_b|$ represents Manhattan distance and B denotes current battery percentage[cite: 1].

3. **Bidding Phase:** `drone1` (located near grid region $(1, 1)$) evaluates a bid cost of 12 for task o_1[cite: 1]. Concurrently, `drone3` (located near grid region $(1, 8)$) evaluates a lower bid cost of 18 for task o_2[cite: 1].

4. **Award Allocation:** The Distribution Hub evaluates incoming proposal messages and awards task o_1 to `drone1` and task o_2 to `drone3` via `accept_proposal` performatives[cite: 1].

5.1.2 Results & Empirical Outcome

Tasks o_1 and o_2 were cleanly partitioned across the fleet in parallel without any global trajectory calculation or central synchronization locks[cite: 1]. By allowing individual agents to evaluate local spatial metrics independently, the fleet achieved emergent spatial division of labor while minimizing global travel distance and communication overhead[cite: 1].

5.2 Scenario 2: Resource-Criticality Emergency Recovery

Goal: Evaluate BDI reactivity when an agent experiences critical hardware/battery failure mid-transit.

- **Execution Event:** Environment injects a sudden battery drain on `drone2` while carrying task $o3$, dropping its charge level to 15%.
- **System Response:** The belief predicate `+battery(15)` fires the reactive safety plan. `drone2` immediately drops its active intention `.drop_intention(deliver(o3, ...))`, executes physical `abort(o3)`, sends a `reassign(o3)` achievement goal to the Hub, and navigates to charger $(9, 0)$.
- **Outcome:** The Hub re-announces $o3$ via CFP. `drone1` wins the bid, picks up $o3$ from the Hub, and completes delivery. `drone2` charges to 80% and re-enters active service status.

5.3 Scenario 3: Mid-Transit Priority Preemption

Goal: Validate system behavior when an emergency Express order is injected while all drones are actively executing standard deliveries.

- **Execution Event:** High-priority order $o7^*$ at $(5, 5)$ is injected while `drone1`, `drone2`, and `drone3` are fully occupied.
- **System Response:** The Hub broadcasts CFP for $o7^*$. Busy drones bid using Plan 2, adding a penalty offset (+30). `drone2` offers the lowest relative bid cost. Upon receiving `award(o7)`, `drone2` preempts its active standard plan ($o4$), notifies the Hub to re-queue $o4$, picks up $o7^*$, and delivers it with top priority.
- **Outcome:** Express order $o7^*$ is delivered within minimal step delay. Task $o4$ is automatically retried by the Hub once a drone frees up, ensuring zero task loss.

Table 4: Validation Summary across Execution Phases

Scenario Phase	Total Tasks	Successful Deliveries	Re-delegations	Status
Phase 1: Normal Operations	2	2	0	**PASSE**
Phase 2: Battery Failure	1	1	1	**PASSE**
Phase 3: Express Preemption	4	4	1	**PASSE**
Total / System Summary	**7**	**7**	**2**	**SUCCESSFU**

—

6 Key Architectural Decisions & Justifications

To guarantee system determinism, prevent concurrency anomalies, and maintain clean theoretical decoupling in `UrbanDeliveryMAS`, several core architectural decisions were integrated into the BDI reasoning logic and environment interface:

Achievement Goal Performatives for Asynchronous Coordination: In standard AgentSpeak(L) semantics, updating an agent's mental state using informative performatives (`tell`) updates the belief base B but fails to trigger a new reasoning cycle if the target predicate already exists. To ensure deterministic task re-negotiation and dynamic re-delegation, the communication protocol utilizes achievement goal performatives (`achieve`). This explicitly injects a top-level goal adoption event (`+!g`) into the target agent's event queue E, forcing immediate intention deliberation regardless of static belief states.

Atomic Plan Execution for Bid Evaluation: Because Jason processes asynchronous agent communication in parallel, multi-threaded bid arrivals at the Distribution Hub create potential critical race conditions during contract evaluation. Annotating the Hub's contract evaluation and award plan with the `[atomic]` directive guarantees non-preemptable execution. The interpreter executes the entire decision block—from incoming bid aggregation to winner selection and award dispatch—without thread interleaving, completely eliminating duplicate or conflicting task allocations.

Uniform Prohibitive Bidding Heuristic (99999): To preserve a deterministic N-responder Contract-Net Protocol (CNP) without introducing stateful timeout mechanics or complex drop-message handling, busy or battery-critical agents do not ignore Call-For-Proposals (`cfp`). Instead, they synchronously return a prohibitive uniform cost bid of 99999. This ensures the Distribution Hub always receives exactly N bid messages per announcement cycle, allowing winner determination via a clean, stateless internal action (`.min`) without risking protocol stalls.

Decoupled State-Only Environment Abstraction: To adhere strictly to multi-agent system design principles, all cognitive deliberation, path selection heuristics, and goal management are concentrated strictly within the AgentSpeak scripts. The underlying Java Environment API is kept purely stateful and reactive, responsible only for validating physical grid movements, updating coordinates, and publishing raw percepts. This architectural boundary prevents "cognitive leak" into the simulation code, ensuring the system's reasoning remains fully transparent and verifiable.

—

7 Conclusion

This project demonstrates that a fully decentralized Multi-Agent System driven by local Belief-Desire-Intention (BDI) reasoning and Contract-Net Protocol (CNP) negotiation can successfully solve complex, dynamic vehicle routing problems without relying on centralized control. Key operational behaviors—including emergent spatial task division, dynamic fault recovery under unexpected power depletion, and real-time preemption for high-priority express orders—emerge organically from modular, human-readable AgentSpeak(L) plans executing on the Jason 3.3.0 framework.

Empirical evaluation across all stress-test scenarios confirmed a 100% task delivery rate (7/7 orders delivered) with zero unhandled goals or concurrency deadlocks, accompanied by clean runtime self-termination. These results prove that purely localized BDI agent deliberation

provides a robust, fault-tolerant, and highly adaptable paradigm for dynamic urban logistics, effectively trading centralized computational bottlenecks for local resilience.