Tecnológico de Monterrey

Integration of Computer Security in Networks and Software Systems (Gpo 401)

Evidence 2. Review

Juan Pablo Buenrostro | A01645981

Joaquín Hiroki Campos Kishi | A01639134

Esteban Camilo Muñóz Rosero | A01644609

Fernanda Yaltzin Jiménez Ramos | A01645485

November 29, 2025.

Updated Work Plan

This document presents the visual modeling used in the development of the **Detection and Management of Vegetation with Anomalies – Multi-Agent System (Traffic Jam)**. The diagrams included here define the structure, communication, reasoning flow, and behaviors of the autonomous agents operating inside a simulated greenhouse environment.

Team Composition and roles

Fernanda Jimenez	Environment modeling, Unity integration, diagrams
Juan Pablo Buenrostro	MAS logic, Unity
Joaquin Hiroki	Metrics, logging, documentation
Esteban Muñoz	Unity agents, movement and visualization

Problem Statement

In this review phase, we focused on bridging the gap between our conceptual Multi-Agent System models and a functional Unity prototype. Building on the global and sequence diagrams defined in our previous deliverable, we implemented the core logic for detection and treatment within the virtual greenhouse.

Key achievements included:

- Unity Scene Setup: We built the virtual greenhouse environment, including the terrain, aisles, and crop rows, integrating all necessary assets and prefabs for the simulation.
- Plant Health Model: We implemented the PlantHealth logic, defining states for Healthy, Detected, Infected, and Treated. Each state is visually represented by a specific color. We also used coroutines to handle state transitions, allowing treated plants to revert to a healthy state after a set interval.
- Infection Simulation: We created a script (InfectRandomPlants) to generate random infections at the start of the simulation. This allows us to test agent behaviors dynamically without manual interference.
- Response Agent (Ground Robot): We implemented the autonomous movement for the ground agent (Healer). It now follows predefined waypoints and includes logic to "heal" infected plants within its action radius.
- Monitoring Agent (Drone): We developed the reconnaissance movement for the drone. It scans for healthy plants, marks them as Detected, and simulates the progression to Infected if they are not treated in time.

Effort and Timeline Adjustments: Initially, we estimated about 10 to 12 hours to transition from modeling to a playable prototype. However, realistically, reaching this stage took approximately 16 hours. This increase was due to the learning curve associated with Unity and the troubleshooting required to align our code with the conceptual diagrams (specifically debugging collisions and state flows).

Diagrams

Agent Architecture

Detection Agent

A reactive agent responsible for continuously scanning the greenhouse environment. It perceives plant conditions through its sensors, detects potential anomalies, and immediately sends anomaly information to the Analysis Agent without internal deliberation.

• Analysis Agent

A reasoning agent that processes anomaly reports using a Belief–Desire–Intention model. It validates symptoms, classifies anomaly types, updates the internal anomaly map, and forwards validated information to the Decision Agent for further action planning.

• Decision Agent

A hybrid agent combining reactive responses with deliberative decision-making. It integrates the information received from multiple agents, evaluates possible actions, prioritizes them, and dispatches the most appropriate tasks to the Action Agents.

• Action Agent

A physical executor responsible for carrying out the tasks assigned by the Decision Agent. It navigates the greenhouse, performs treatments or inspections, interacts with plants, and reports completion status back to the system.

The diagrams included are:

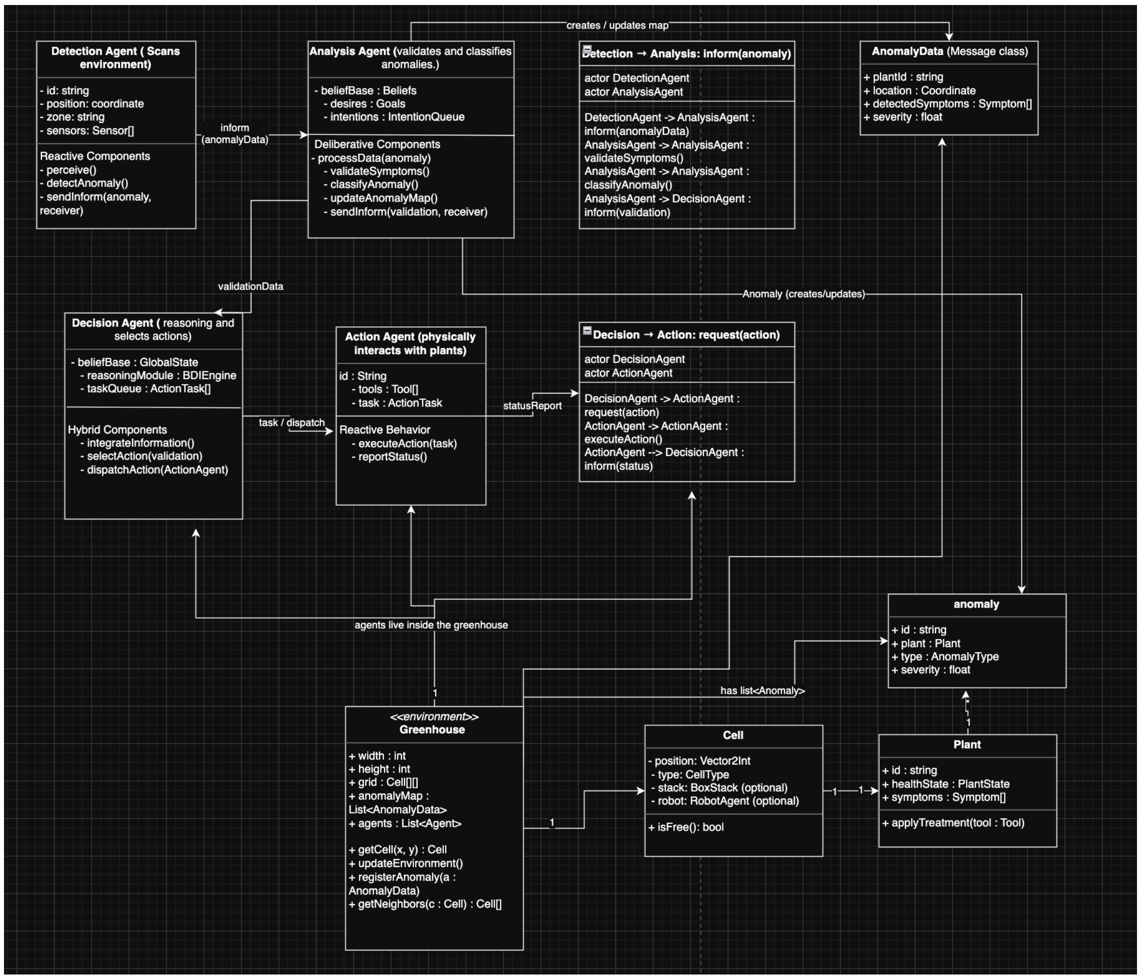
1. **Global MAS Class Diagram**
   Shows all agents, the greenhouse environment, plants, anomalies, messages, and their relationships.

2. **AUML Sequence Diagram**
   Describes the full communication flow between agents during an anomaly event.

3. **Agent State Machines** (Detection, Analysis, Decision, Action)
   Represent the internal behavior and transitions of each agent during operation.

These diagrams serve as a foundational reference for understanding, implementing, and validating the Multi-Agent System, ensuring consistency across modeling, Unity development, and team communication.

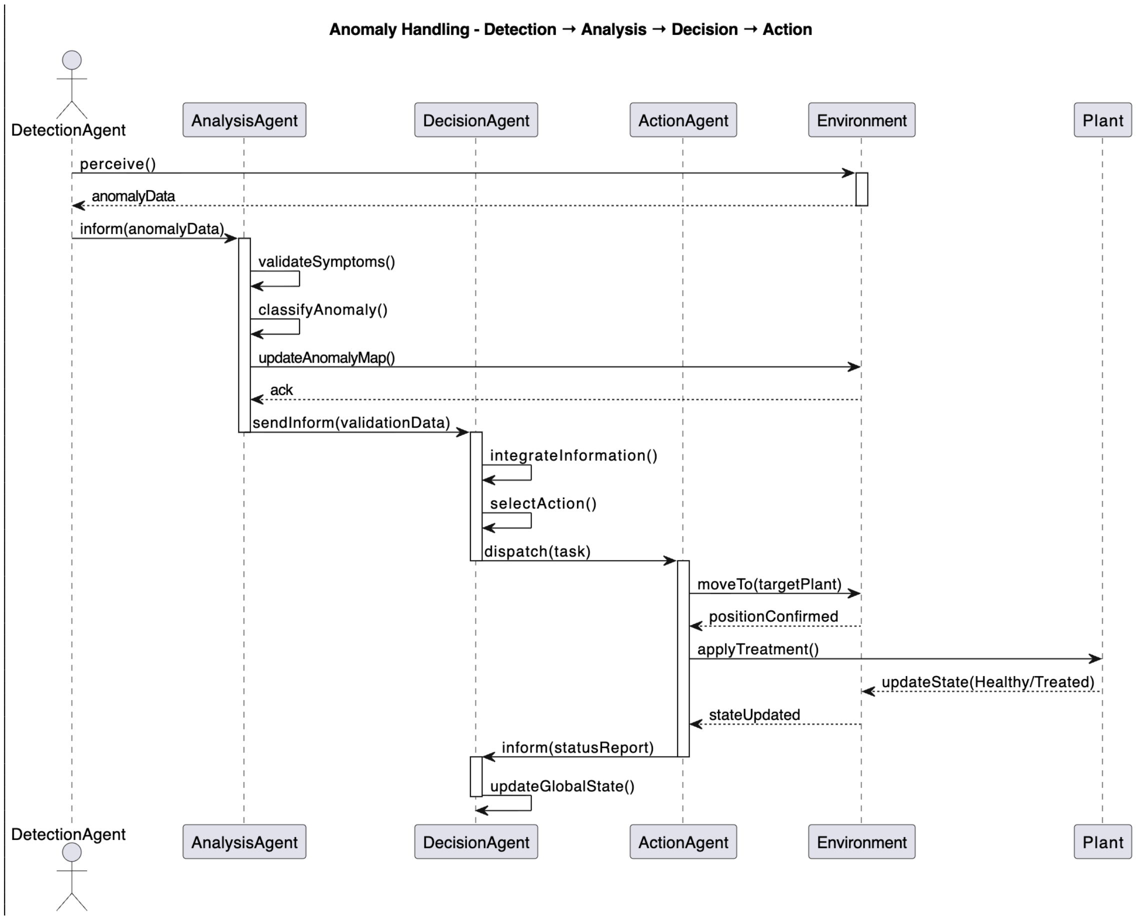
Global MAS Class Diagram



creates / updates map
Detection Agent (Scans environment)
- id: string
- position: coordinate
- zone: string
- sensors: Sensor[]
Reactive Components
- perceive()
- detectAnomaly()
- sendInform(anomaly, receiver)
inform (anomalyData)
Analysis Agent (validates and classifies anomalies.)
- beliefBase : Beliefs
- desires : Goals
- intentions : IntentionQueue
Deliberative Components
- processData(anomaly)
- validateSymptoms()
- classifyAnomaly()
- updateAnomalyMap()
- sendInform(validation, receiver)
Detection → Analysis: inform(anomaly)
actor DetectionAgent
actor AnalysisAgent
DetectionAgent -> AnalysisAgent : inform(anomalyData)
AnalysisAgent -> AnalysisAgent : validateSymptoms()
AnalysisAgent -> AnalysisAgent : classifyAnomaly()
AnalysisAgent -> DecisionAgent : inform(validation)
AnomalyData (Message class)
+ plantId : string
+ location : Coordinate
+ detectedSymptoms : Symptom[]
+ severity : float
validationData
Anomaly (creates/updates)
Decision Agent (reasoning and selects actions)
- beliefBase : GlobalState
- reasoningModule : BDIEngine
- taskQueue : ActionTask[]
Hybrid Components
- integrateInformation()
- selectAction(validation)
- dispatchAction(ActionAgent)
task / dispatch
Action Agent (physically interacts with plants)
id : String
- tools : Tool[]
- task : ActionTask
Reactive Behavior
- executeAction(task)
- reportStatus()
statusReport
Decision → Action: request(action)
actor DecisionAgent
actor ActionAgent
DecisionAgent -> ActionAgent : request(action)
ActionAgent -> ActionAgent : executeAction()
ActionAgent --> DecisionAgent : inform(status)
agents live inside the greenhouse
1
anomaly
+ id : string
+ plant : Plant
+ type : AnomalyType
+ severity : float
has list<Anomaly>
1
<<environment>>
Greenhouse
+ width : int
+ height : int
+ grid : Cell[][]
+ anomalyMap : List<AnomalyData>
+ agents : List<Agent>
+ getCell(x, y) : Cell
+ updateEnvironment()
+ registerAnomaly(a : AnomalyData)
+ getNeighbors(c : Cell) : Cell[]
1
Cell
- position: Vector2Int
- type: CellType
- stack: BoxStack (optional)
- robot: RobotAgent (optional)
+ isFree(): bool
1
1
Plant
+ id : string
+ healthState : PlantState
+ symptoms : Symptom[]
+ applyTreatment(tool : Tool)

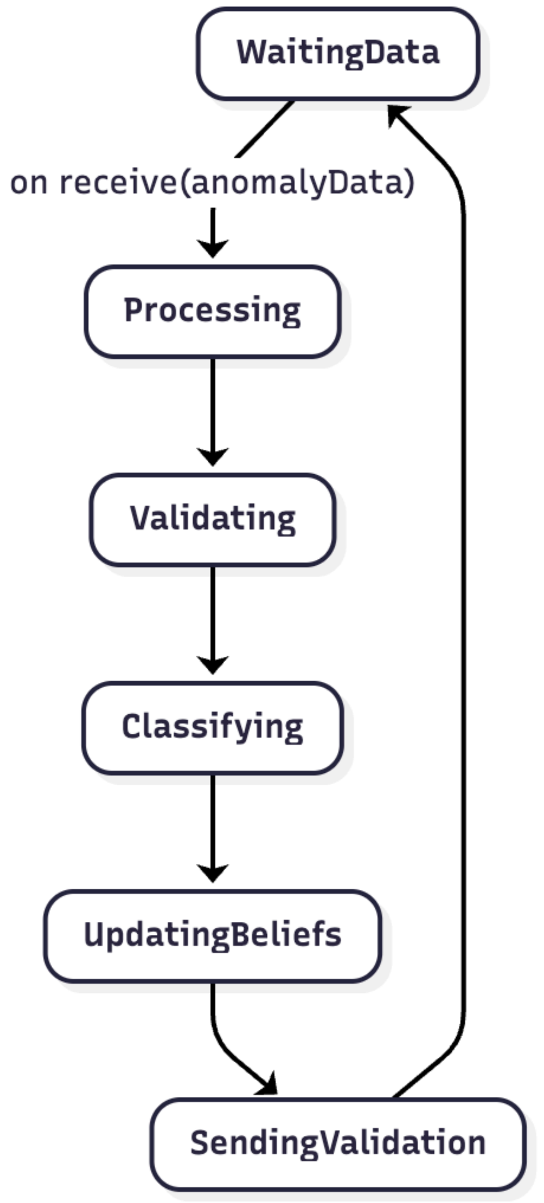
Link to view the complete Diagram:
https://app.diagrams.net/?title=Global%20Diagram&lightbox=1&page-id=YlH-tHGrtLMmND71Ih9b&client=1

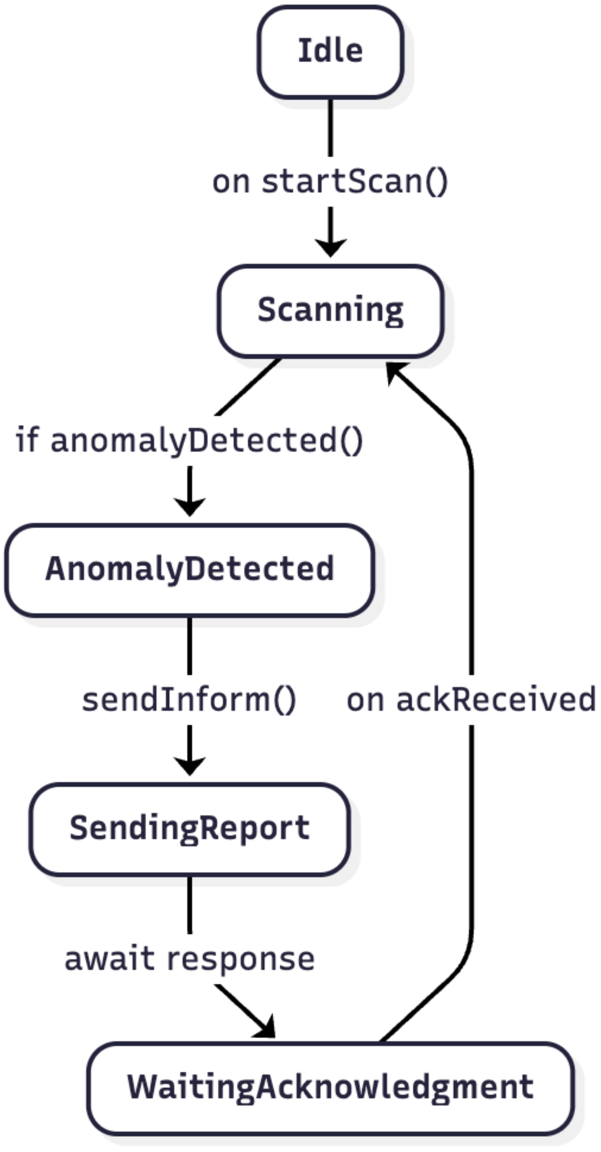
AUML Sequence Diagram

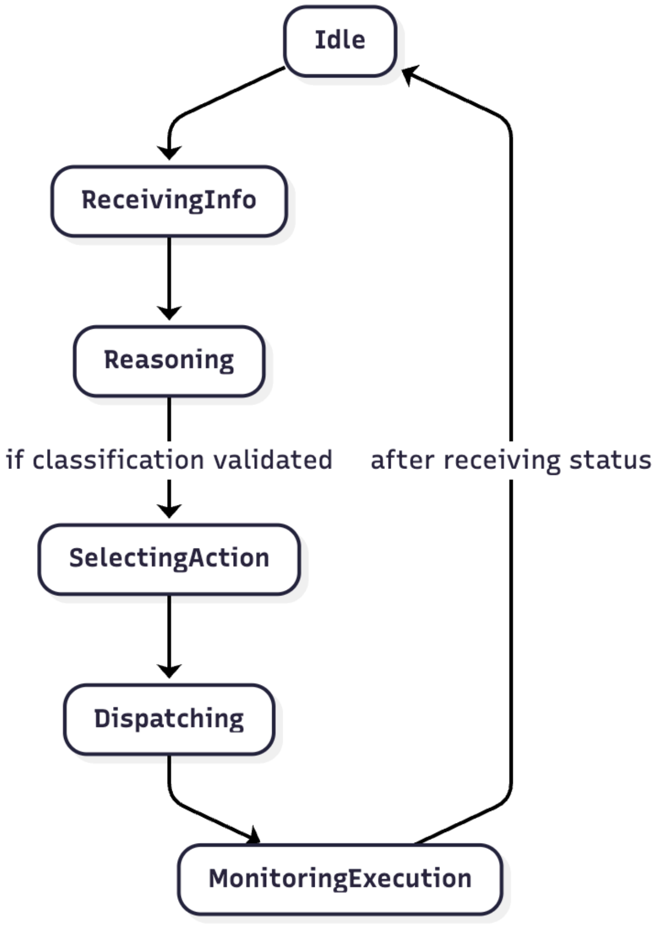


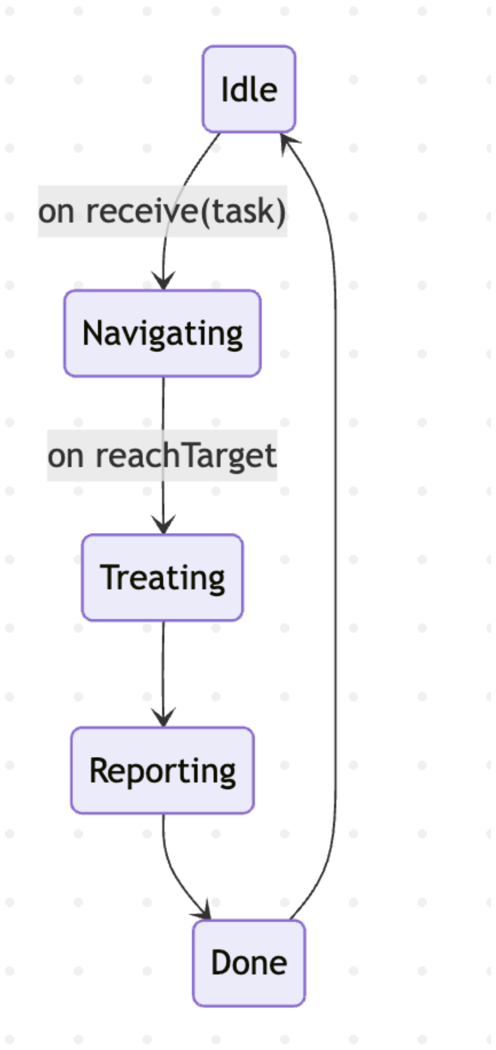
Link to the Diagram:

https://app.diagrams.net/?title=AUML&lightbox=1&page-id=fbwLEcjqp9B_76X9By F5&client=1


WaitingData
on receive(anomalyData)
Processing
Validating
Classifying
UpdatingBeliefs
SendingValidation


Idle
on startScan()
Scanning
if anomalyDetected()
AnomalyDetected
sendInform()
on ackReceived
SendingReport
await response
WaitingAcknowledgment


Idle
ReceivingInfo
Reasoning
if classification validated
SelectingAction
Dispatching
MonitoringExecution
after receiving status



Pending Activities & Next Steps:

Analysis & Decision Logic: Complete the code implementation for the AnalysisAgent and DecisionAgent to fully reflect the AUML and class diagrams. Target: Next week.

Monitoring Dashboard: Integrate a simple in-game UI to track statistics (infected vs. treated plants). Target: One week before final delivery.

Visual Polish: Refine the user interface and capture the final demonstration video. Target: Final project week.

Final Documentation: Update the report with performance metrics (e.g., treatment success rate).

With this updated plan, the work submitted in this review serves as the functional foundation of the system. Future iterations will focus on refining behaviors and visualizing data.

Lessons Learned

One major takeaway from this review was realizing that translating theoretical diagrams into Unity code isn't a linear process. While the AUML and global diagrams provided a necessary structure, we had to iterate several times to get the agents to behave correctly within a 3D environment involving physics and real-time updates.

We also confirmed the importance of dividing responsibilities. While part of the team focused on refining the models and diagrams, the rest focused on the practical implementation. This parallel workflow was efficient but required constant communication; we found that several scripting errors (such as null references or state stagnation) stemmed from minor misunderstandings of the original agent models.

Technical & Process Insights:

Unity Mechanics: We gained a deeper understanding of using coroutines for state timing and Physics.OverlapSphere for handling the robot's area of influence when treating plants.

Workflow Integration: We learned to better align the infection-detection-action loop in code with the phases described in our diagrams.

Version Control: This review reinforced the need for strict Git discipline. Using branches, clear commit messages, and tagging this release as REVISION_3 allowed us to create a clean "snapshot" of the project that matches this week's deliverable.