Tecnológico de Monterrey

Guadalajara campus

Evidence 2. Final document

José Yael Varela García - A01645324

Faisal Alali- A01830963

Sebastian Certuche - A01644942

Sebastián Alejandro Soria Piñuela - A01645849

Danilo Paolo Tato Velázquez - A01644630

Modeling of multi-agent systems with computers graphics

Group 301

December 05, 2025

Index

Final UML & AUML diagrams.

Agent Class Diagrams

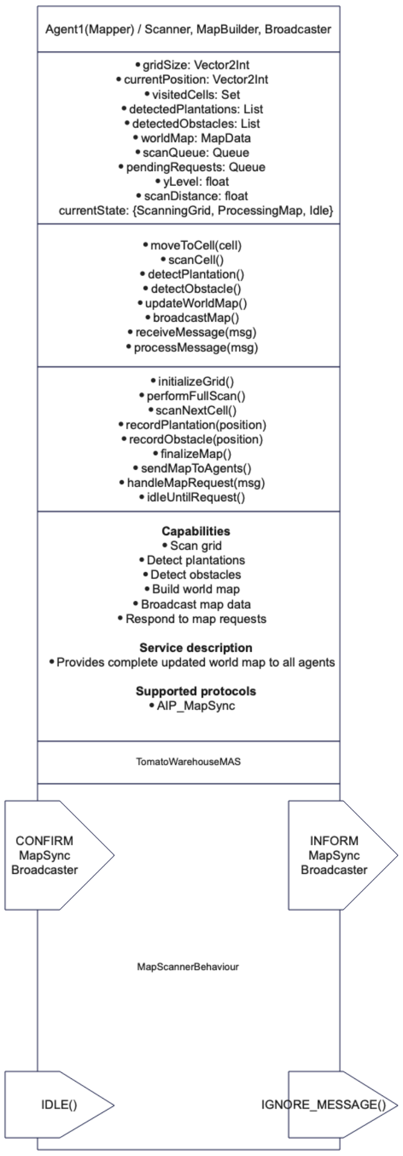

Fig : Agent 1 (Mapper) Class Diagram

Agent 1 is responsible for fully exploring the warehouse and generating the global map of the environment. It moves through the grid, identifies the position of each tomato plantation, and detects relevant obstacles. Once the initial exploration is completed, it processes the information and shares it with the other agents. Afterward, it remains in an idle state until it receives an update request, at which point it scans the necessary areas again and synchronizes the shared map.

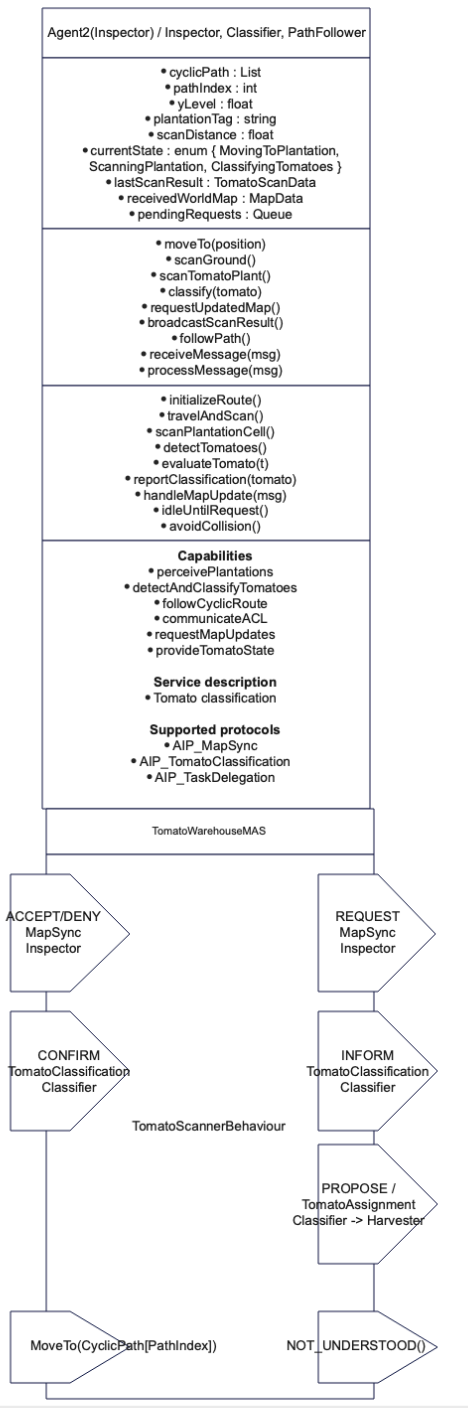

Fig : Agent 2 (Inspector) Class Diagram

Agent 2 is responsible for traversing all the plantations following the route generated by Agent 1. At each point of interest, they inspect the plantation, detect the tomatoes present, and classify their condition, determining whether they are ripe, healthy, or infected. After classification, they send the corresponding information to the collecting agents (Agents 3), assigning which tomatoes should be removed and by whom.

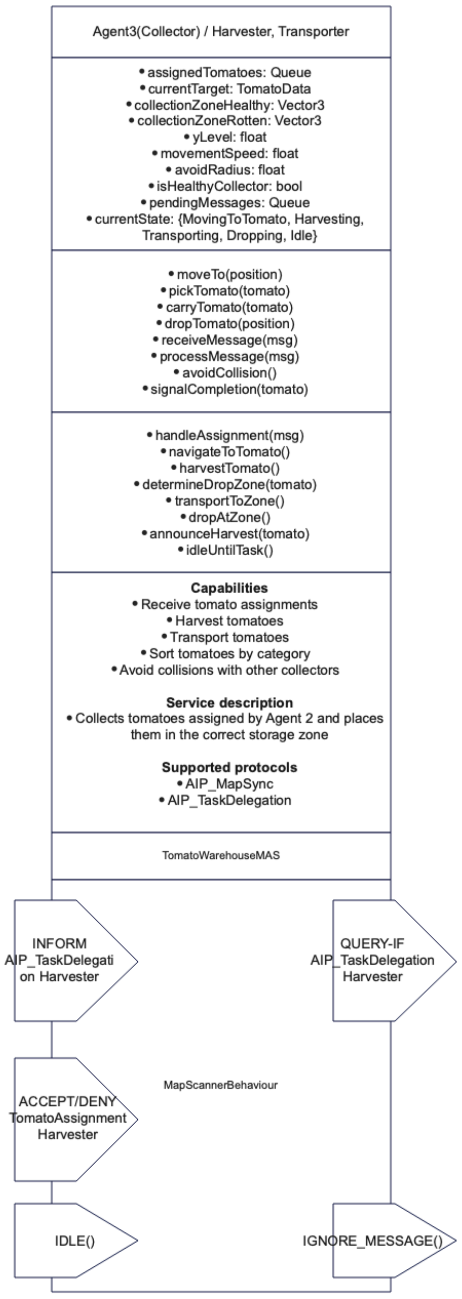
Agent3(Collector) / Harvester, Transporter
• assignedTomatoes: Queue
• currentTarget: TomatoData
• collectionZoneHealthy: Vector3
• collectionZoneRotten: Vector3
• yLevel: float
• movementSpeed: float
• avoidRadius: float
• isHealthyCollector: bool
• pendingMessages: Queue
• currentState: {MovingToTomato, Harvesting, Transporting, Dropping, Idle}
• moveTo(position)
• pickTomato(tomato)
• carryTomato(tomato)
• dropTomato(position)
• receiveMessage(msg)
• processMessage(msg)
• avoidCollision()
• signalCompletion(tomato)
• handleAssignment(msg)
• navigateToTomato()
• harvestTomato()
• determineDropZone(tomato)
• transportToZone()
• dropAtZone()
• announceHarvest(tomato)
• idleUntilTask()
Capabilities
• Receive tomato assignments
• Harvest tomatoes
• Transport tomatoes
• Sort tomatoes by category
• Avoid collisions with other collectors
Service description
• Collects tomatoes assigned by Agent 2 and places them in the correct storage zone
Supported protocols
• AIP_MapSync
• AIP_TaskDelegation
TomatoWarehouseMAS
INFORM AIP_TaskDelegation Harvester
QUERY-IF AIP_TaskDelegation Harvester
MapScannerBehaviour
ACCEPT/DENY TomatoAssignment Harvester
IDLE()
IGNORE_MESSAGE()

Fig : Agent 3 (Collector) Class Diagram

Agent 3 receives collection tasks from Agent 2, indicating which tomatoes need to be picked and to which pile they should be transported based on their condition (healthy or infected). Each instance of the agent is specialized in a type of tomato to prevent cross-contamination, so they coordinate with each other to avoid interference or collisions during their movements. The agent navigates to the indicated location, harvests the assigned tomato, and transports it to its corresponding pile. Additionally, it can consult with other collecting agents to avoid task duplication and ensure that no tomato is attended to by more than one agent at the same time.

Protocol Diagrams (AIP)

Activation protocol

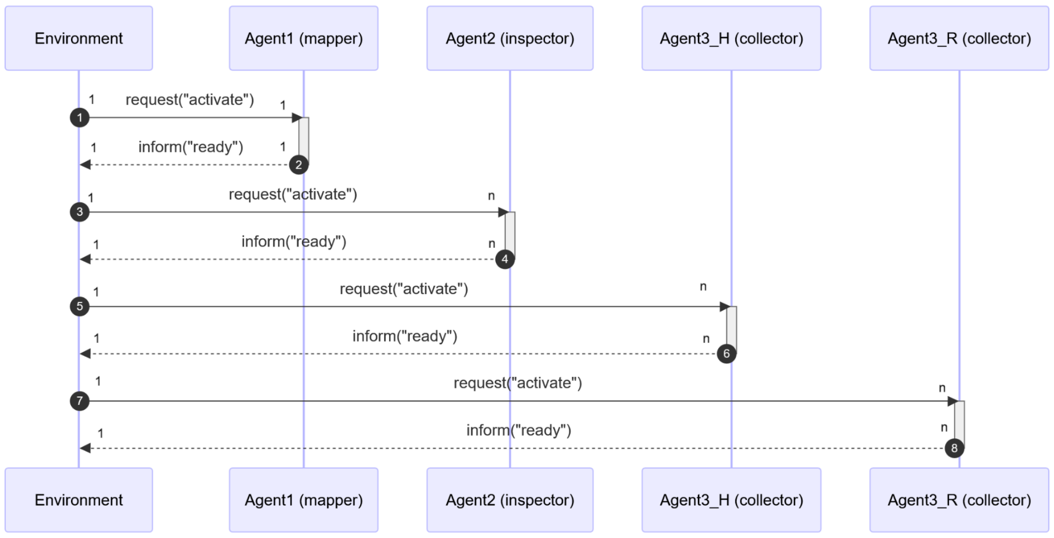

Fig : AIP Diagram of the Activation protocol.

This interaction diagram represents how the Environment component initializes the multi-agent system by sending activation requests to all the agents. Each agent replies to the activation with a readiness message, indicating that it has completed its initialization process and is prepared to execute its assigned task in the system.

Mapping protocol

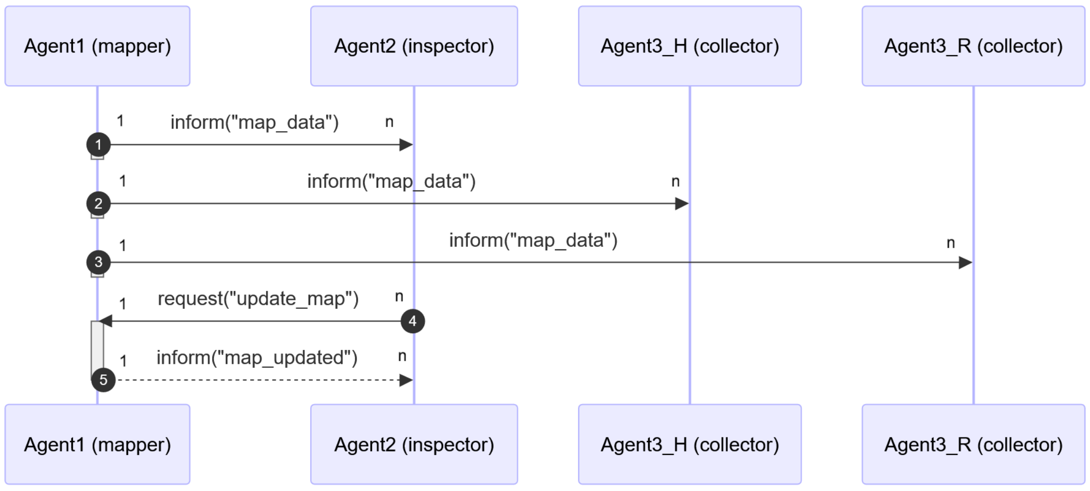

Fig : AIP Diagram of the mapping protocol.

This interaction diagram represents how Agent 1 (The mapping agent), acts as the system's mapper, and with that shares the updated greenhouse information with all the other agents after scanning the environment, and upon receiving a direct update request, performs a rescan and informs the requester that the map has been refreshed.

Inspection protocol

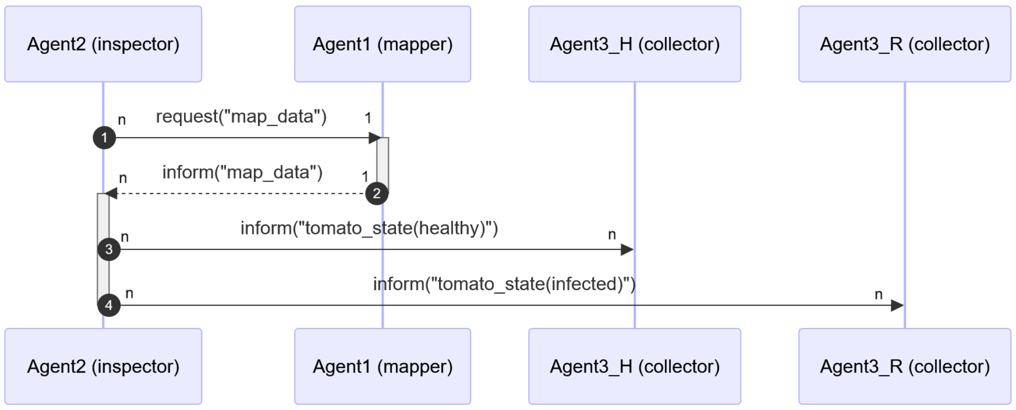

Fig : AIP Diagram of the inspection protocol.

This interaction diagram represents how Agent 2 (the tomato inspector, which is the one that classifies infected and healthy tomatoes) requests the latest map data from Agent 1 (the mapping agent) before beginning its inspection pathing routine. After receiving the information, the Inspector analyzes the tomato plants, classifies each tomato as healthy or infected, and sends state reports to the correct collector agents (Healthy or infected collector agent), enabling them to prepare for cutting and harvesting tasks.

Collection protocol

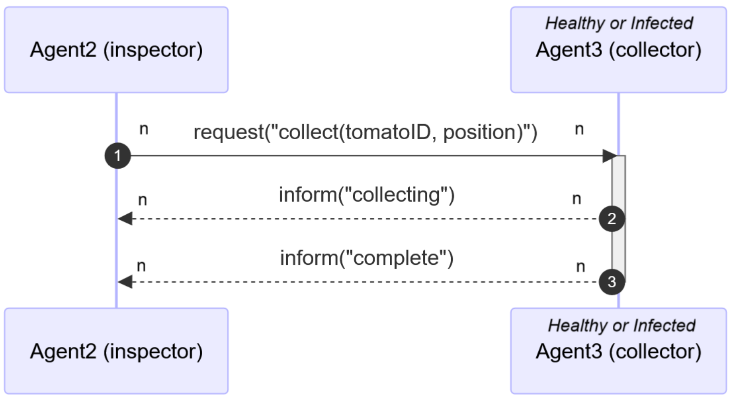

Fig : AIP Diagram of the Collection protocol.

This AIP diagram represents how Agent 2 (the inspector agent) assigns a collection task to a specific Agent 3 collector (healthy or infected collectors) by sending a request that includes the target tomato's ID and location. The collector acknowledges this task, begins the harvesting process, and after that sends a completion notification back to the inspector to indicate that the tomato has been successfully collected.

Work distribution protocol

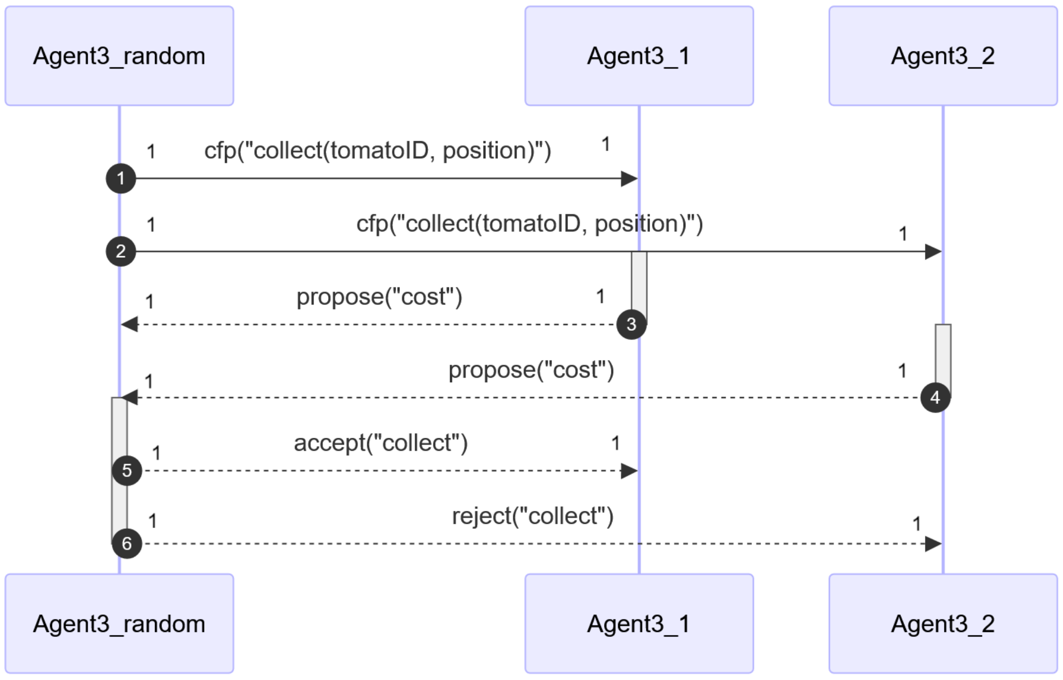

Fig : AIP Diagram of the Work distribution protocol, in this diagram we represented a leader and two workers to simplify the logic of accepting and rejecting offers, but in the project there are going to be more collector agents.

This AIP diagram represents how one of the Agent 3 (Collector) units assumes the role of a leader and announces a new task for collection of a respective tomato to the rest of the Collector agents. The leader broadcasts a call for proposals (cfp in the diagram) containing the tomato information that needs to be collected (such as the ID and the coordinates). Each collector evaluates its own distance and workload, then they reply with a proposal indicating its estimated cost. The leader compares the proposals, selects the best collector, based on the cost, by sending an acceptance message rejecting the other collector agents.

Collision avoidance protocol

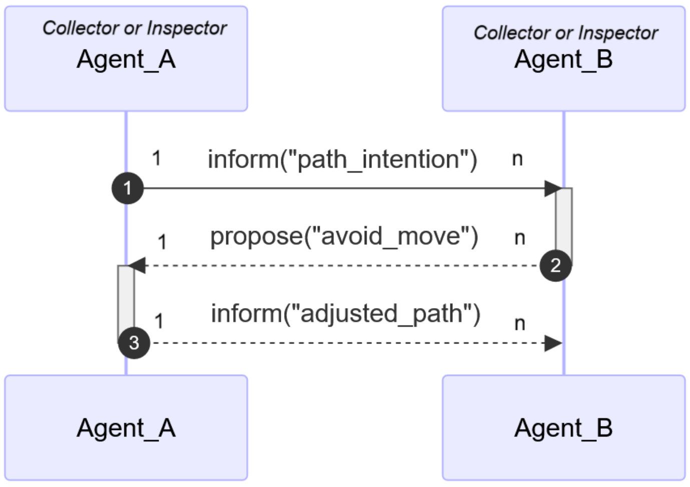

Fig : AIP Diagram of the Collision avoidance protocol for both the Inspector (Agent 2)and the Collector (Agent 3) agents .

This AIP diagram represents how the agents 2 and 3 (inspector and collector agents) coordinate to avoid collisions between them while navigating the greenhouse. Whenever an agent intends to move, it sends its planned path to nearby agents. Then the receiving agent analyzes the path, detects hypothetical conflicts that could result in a collision, and replies with an alternative path if needed. After receiving the possible suggestion, the originating agent adjusts its trajectory and with that avoids the collision.

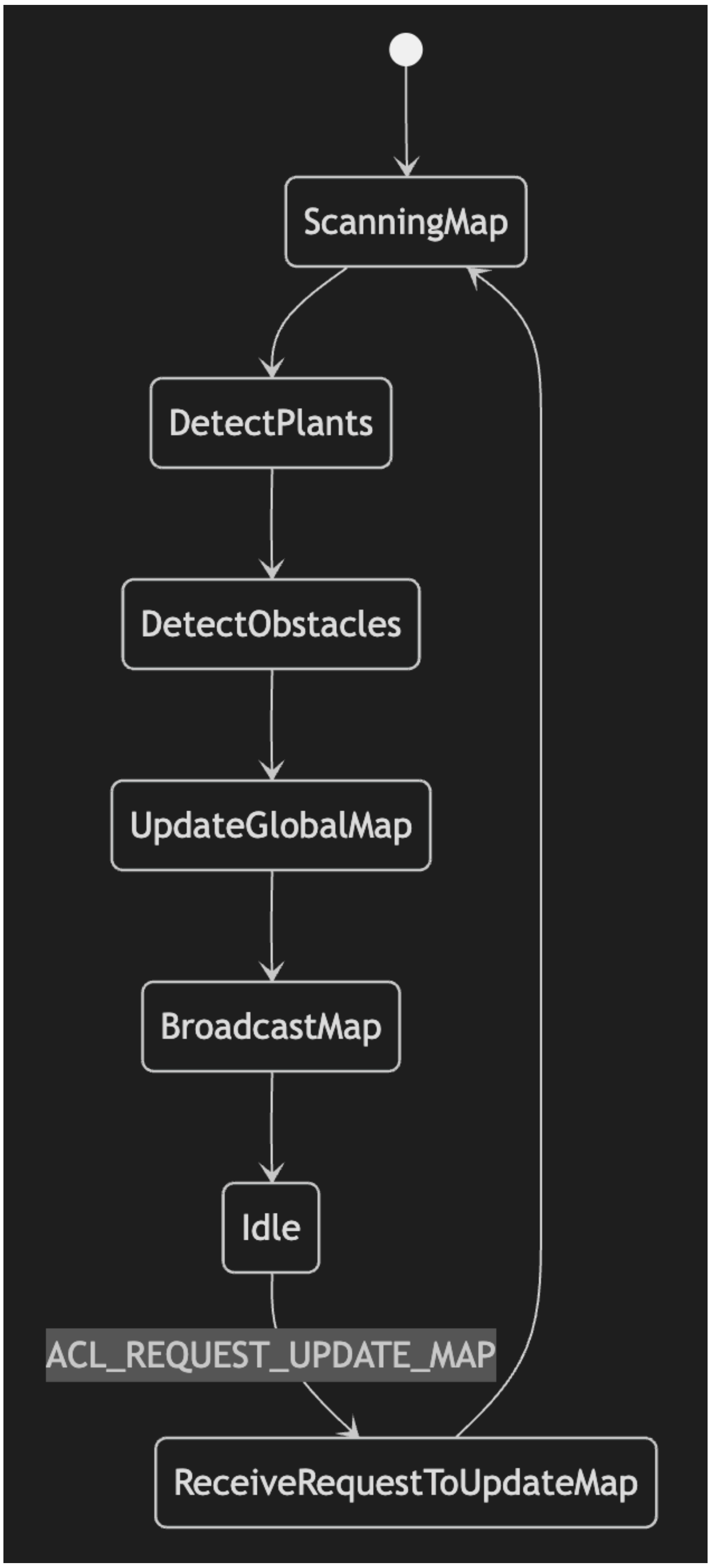

Fig : Activity Diagram that describes the life cycle of Agent1

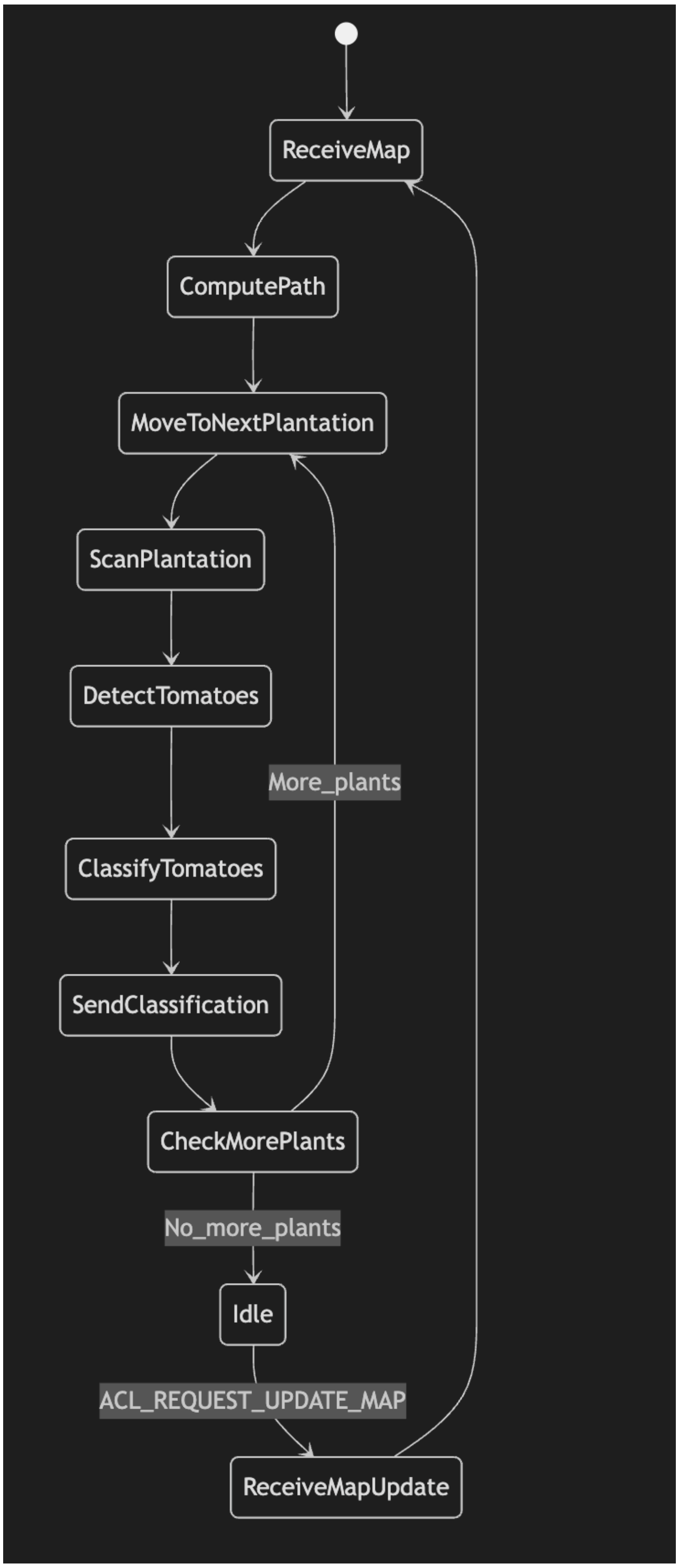

Fig : Activity Diagram that describes the life cycle of Agent2

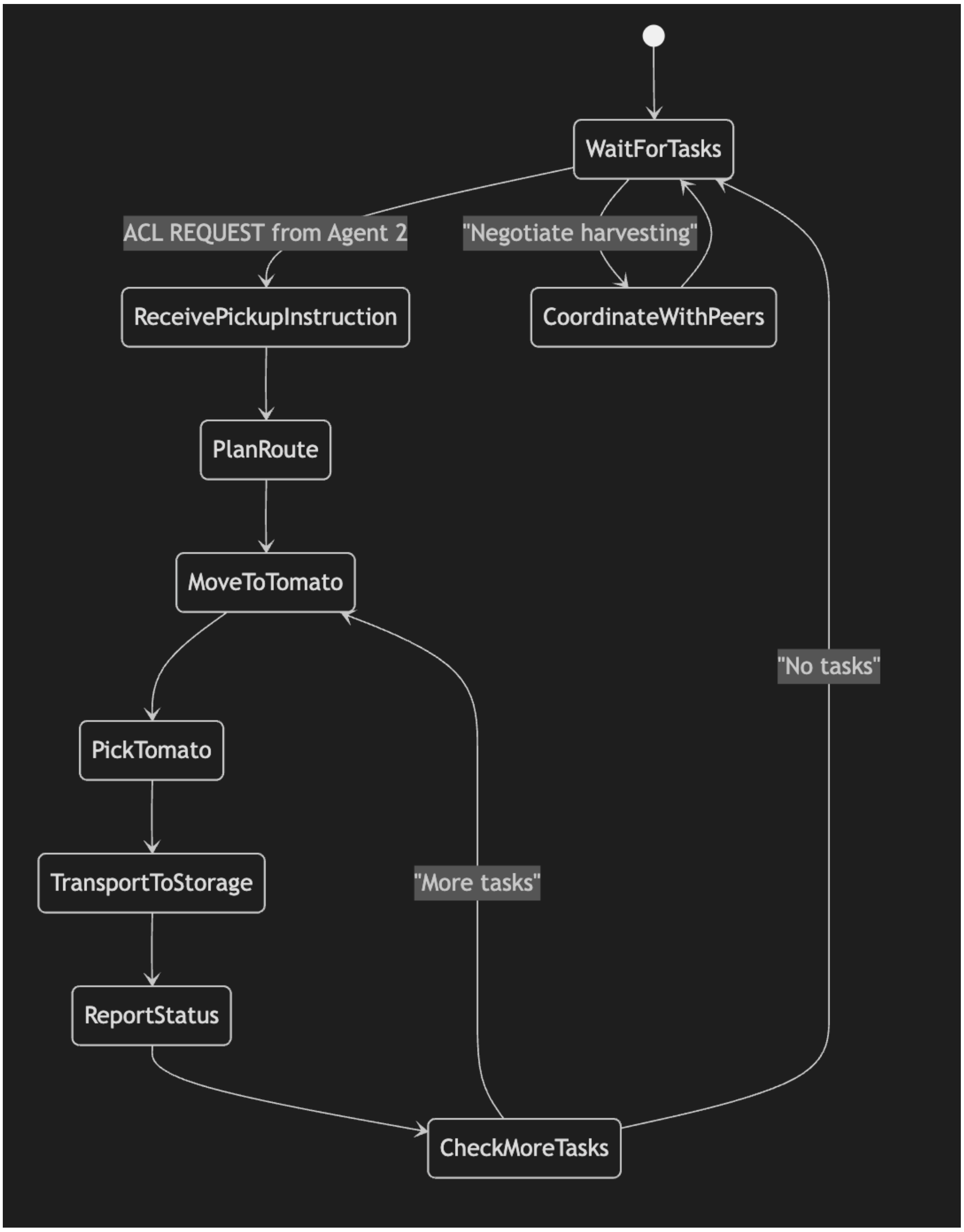

Fig : Activity Diagram that describes the life cycle of Agent3

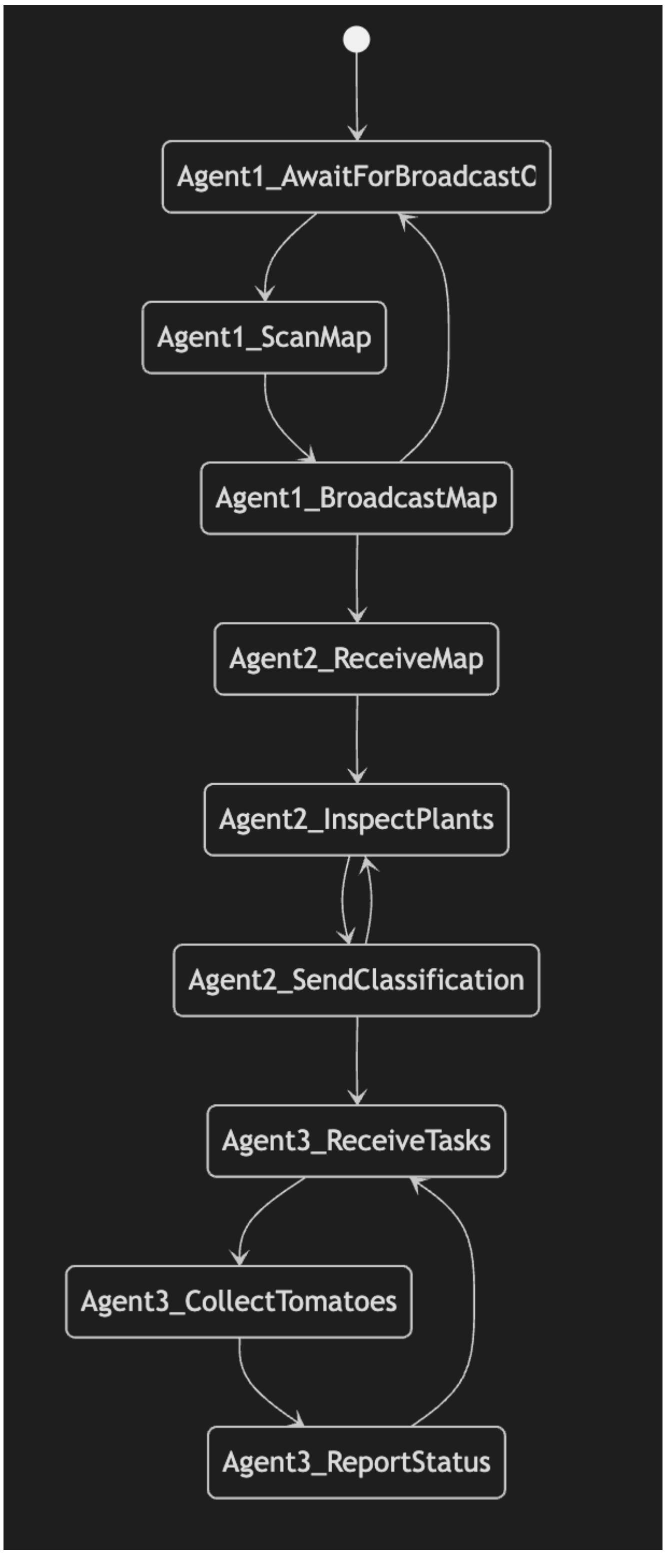

Fig : Activity Diagram that describes sequence of interactions between different agent classes

Class Diagram

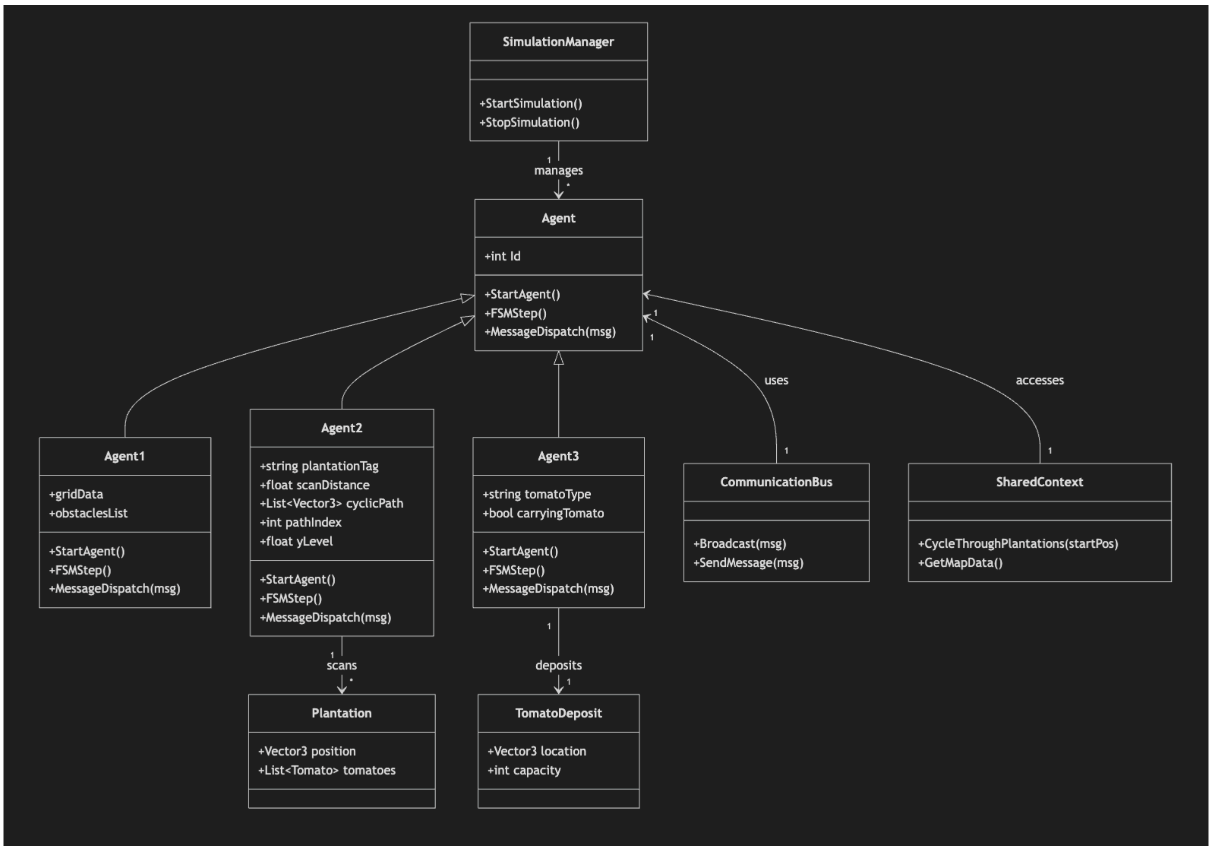

Fig : Inherency-based implementation of agents and their relationships with communication-relevant classes

The diagram shows key relationships. Agents use the communication bus and shared context; the TomatoesScanAgent interacts with plantations by scanning them, and CutterAgents deposit tomatoes in the appropriate TomatoDeposit locations. The SimulationManager coordinates all agents.

Installation process for Unity

The installation of our project requires no additional configuration. The user only needs to import the provided Unity package into a new Unity project. Once imported, Unity automatically loads all scenes, scripts, and assets. To run the simulation, the user simply opens the main scene and clicks the “Run” (Play) button. No extra setup or external dependencies are required.