INTRODUCTION TO MULTIAGENT SYSTEMS

Emergency Response Problem

Task 2 - Coordination Design

[1]

Joan Caballero Castro
Oriol Miró López-Feliu
Marc Gonzalez Vidal
Julia Amenós Dien
Víctor Carballo Araruna

Master in Artificial Intelligence
Universitat Politècnica de Catalunya (UPC)
Universitat Rovira i Virgili (URV)

Quadrimester 1, 2024/25

Contents

1 Introduction

This document details the design of a city's emergency response problem, developed for the Introduction to Multiagent Systems (IMAS) course within the AI Master's program. The objective is to coordinate a group of autonomous agents to create a plan to effectively manage and mitigate fire incidents. The design follows the standard multiagent system architecture principles [2], and draws insights from agent-based control systems for complex engineering applications [3].

In this second task, we define the cooperation and coordination mechanisms necessary to address the emergency response problem. We outline the processes each individual crew will follow and the sequence of tasks; the Pydantic outputs for tasks that benefit from structured outputs; and the interactions between different agent crews using pipelines and routers.

For this task, following the professor's guidance, we modified the structure of the agent crews compared to the previous submission. Specifically, the Material Expert, Ambulance Manager, and Hospital Manager agents were too complex. To simplify, we decided to split each into separate agents, allowing each to handle more specific and manageable tasks. The final design of our agents and crews is illustrated in Figure 5.

2 Process Definition

In this section, we discuss how the process is structured as sequential or hierarchical, as well as who is managing other agents in the case of hierarchical crews.

2.1 Emergency Crew

Let us begin with the emergency services crew, depicted in Figure 1. The tasks that need to be performed are summarizing the key points from the input data in natural language and dividing the work between the firefighter and Medical Services Crews. These tasks are fully dependent on each other and thus this process needs to be carried out sequentially, one task per agent, adhering to well-established principles of multiagent coordination [4].

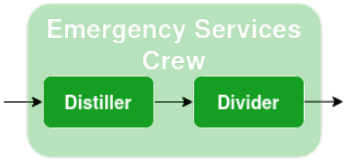

Figure 1: Diagram of the Emergency Crew.

There is the possibility of adding planning to enhance the performance of the crews, even though this would require an extra planning LLM, not included in the diagram.

2.2 Firefighter Crew

The Firefighter Crew, shown in Figure 2, is much more complex than the emergency services crew, with different parallel workflows and tasks that depend on outputs from several agents.

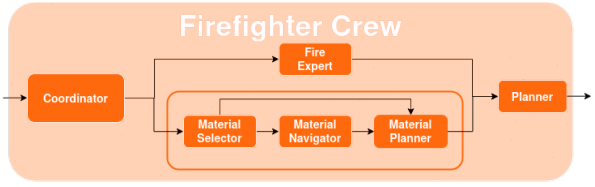

Figure 2: Diagram of the Firefighter Crew.

The structure of this crew is hybrid, blending hierarchical and sequential elements to address the complexity of firefighting tasks effectively, as recommended for structuring complex agent teams in distributed problem-solving [3].

It benefits from a hierarchical organization, with the controller agent acting as the manager. The controller decomposes the complex set of tasks that the Firefighter Crew handles into smaller, more manageable sub-tasks for the rest of the agents it manages. Examples of such tasks include selecting the correct team of firefighters or materials depending on the emergency conditions and the type of fire.

In addition, the structure of this crew enables some degree of parallelization. The tasks of the fire expert, on one hand, and the material selector, navigator, and planner agents, on the other, can be performed independently of each other.

Nevertheless, the tasks of the material selector, navigator, and planner agents need to be performed one after the other, in that order. Additionally, the firefighter planner needs to complete its task once the fire expert and material planner have finished their tasks, so there is no more room for parallelization on that side. All these dependencies introduce sequential properties to the crew organization.

2.3 Medical Services Crew

The medical services crew, depicted in Figure 3, is in a similar situation to the previously discussed Firefighter Crew. The complexity of the tasks that this crew has to accomplish, combined with the number of parallel workflows that depend on one another, makes this crew suitable for a hybrid process.[5, 6]

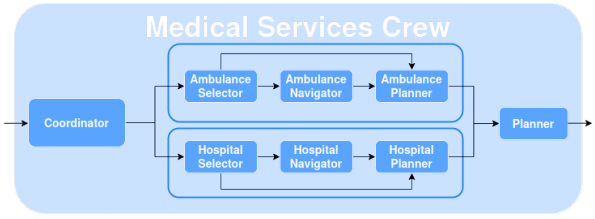

Figure 3: Diagram of the Medical Services Crew.

The manager of this hierarchical planning is again the controller agent of the crew. This agent is in charge of delegating the complex task of organizing the medical planning and selecting the appropriate hospital and ambulances into smaller and more manageable sub-tasks. These sub-tasks are then distributed among specialized agents within the crew.

There are two completely parallel workflows: the ambulance selection, navigation, and planner on one hand, and the hospital selector, navigator, and planner on the other. These can be done simultaneously, as there are no dependencies until the final planner agent.

However, there are strong dependencies within the ambulance and hospital groups, so the three tasks that compose each group have to be performed sequentially. Then, the final planner agent, who is in charge of combining the work of the previous agents, has to wait until the ambulance and hospital planners finish their tasks before starting its own.

2.4 Reporter Crew

The last crew is the Reporter Crew, shown in Figure 4. It is responsible for taking the plans produced by the medical and Firefighter Crews and transforming them into a readable, ready-to-deploy report in Markdown format.

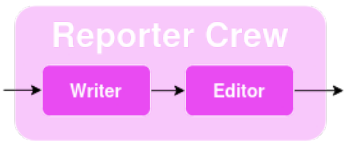

Figure 4: Diagram of the Reporter Crew.

Its structure is similar to the Emergency Crew. It has two agents: one in charge of writing the report and another whose task is to act as an editor for the produced plan. The flow of tasks is completely sequential; the second agent needs the output of the first to start its task, so we opt for this type of crew structure.

As was the case with the Emergency Crew, there exists the possibility of adding an extra planning LLM whose role is to provide a step-by-step plan to complete the task, even though it is not added in the diagram.

3 Pydantic Outputs

In this section, we outline the specific agents for which we plan to use Pydantic schemas, as incorporating structured data enhances the efficiency and organization of our system.

3.1 Emergency Crew

3.1.1 Distiller

The task of this agent is to transform raw input from emergency calls into structured data containing the key information. Using Pydantic schemas guarantees that subsequent agents will receive consistent and reliable data, thereby facilitating decision-making. The output format must include the following fields:

- `Location`: Coordinates where the fire originated.
- `Victims`: Estimation of the number of individuals affected by the fire, including those who may be trapped, injured, or at risk.
- `Time of the Call`: The time when the emergency call was received, which is essential for computing the total response time for the plan.
- `Description of the Emergency`: Additional information describing the incident.

3.1.2 Divider

The Divider agent is responsible for communicating emergency information to both the Firefighter and Medical Services Crews. It analyzes the information from the Distiller to determine which information is relevant for each crew's planning. Note that the first three sections were defined by the Distiller.

- `Location`: Coordinates where the fire originated.
- `Victims`: Estimation of the number of individuals affected by the fire, including those who may be trapped, injured, or at risk.
- `Time of the Call`: The time when the emergency call was received, which is essential for computing the total response time for the plan.
- `Firefighter Emergency Information`: Details of the emergency necessary for the Firefighter Crew.
- `Medical Service Emergency Information`: Details of the emergency necessary for the Medical Service crew.

3.2 Firefighter Crew

3.2.1 Coordinator

This agent reads information about the crew resources and communicates all the knowledge to the specialized agents. Since all agents in the crew require the same information, no further division is necessary. Additionally, the Coordinator verifies that the outputs from those agents are valid. Therefore, no specific output is required for the Coordinator.

3.2.2 Fire Expert, Material Selector, and Material Planner

These agents focus on resolving their specialized tasks. Due to the simplicity of their outputs, no specific structured format is necessary.

3.2.3 Firefighter Planner

The Firefighter Planner aggregates resources to formulate a comprehensive fire response plan. The structured output is crucial for generating coherent response plans without missing critical information:

- `Personnel`: Information about the number of units needed for each type of firefighter.
- `Vehicles`: Description of the number and types of vehicles required for the plan.
- `Material`: Additional materials that must be carried to assess the fire (for example, a ladder).
- `Route`: Information about the route from the firefighter station to the fire incident. As all units depart from the same location, a single route will be provided.

3.3 Medical Services Crew

3.3.1 Coordinator

Following the same reasoning applied to the firefighter crew, the Coordinator does not have any specific output requirements.

3.3.2 Hospital Selector, Hospital Navigator and Hospital Planner

These agents focus on resolving their specialized tasks. Due to the simplicity of their outputs, no specific structured format is necessary.

3.3.3 Medical Planner

The **Medical Planner** is responsible for ensuring that all information is provided in a structured format. This is critical to guarantee that all required details are included in the message. Failure to include these elements will prevent the **Reporter Crew** from generating the plan, ultimately causing the system to fail in its task.

The structured output must include the following components:

- `Personnel Employed`: All personnel required in the ambulance, determined based on the number of injured individuals and the severity of their injuries.
- `Ambulances Employed`: The ambulances required for rescuing and treating individuals, whether for transporting them from the fire scene to the hospitals or for attending to those with minor injuries at the scene.
- `Materials Required`: The medical supplies and equipment needed in the ambulances to provide effective treatment.
- `Assigned Hospitals`: The designated hospitals to which ambulances will transport patients if necessary.
- `Rooms Needed in the Hospitals`: Reserved rooms or beds in each hospital to accommodate patients transported from the fire scene.
- `Route from Ambulances to Fire Scene`: The planned route for ambulances to reach the fire scene from their current location.
- `Route from Fire Scene to Hospitals`: The route from the fire scene to the assigned hospitals, in case patients require transportation for further care.
- `Total Response Time`: The time taken for ambulances to travel from their starting location to the fire scene.

3.4 Reporter Crew

3.4.1 Writer

- `Fire Location`: Specifies the coordinates of the fire.
- `Resources Used by Firefighter Crew`: Provides a detailed list of all resources and personnel utilized in the plan, including firefighters, fire trucks, and materials.
- `Firefighter Plan`: Outlines the plan that allocates personnel, vehicles, and materials, establishing their respective roles and relationships.
- `Resources Used by Medical Crew`: Lists all resources and personnel deployed in the plan, such as ambulances, nurses, drivers, doctors, medical supplies, beds, and rooms, specifying the hospital of origin.
- `Medical Crew Plan`: Details the plan that allocates and organizes all resources, defining the relationships between them.
- `Total Response Time for Firefighter Crew`: Indicates the time required for the Firefighter Crew to arrive at the fire scene.
- `Total Response Time for Medical Crew`: Indicates the time required for the Medical Services Crew to arrive at the fire scene.

4 Agent Interaction

In this section, we define how the different agent crews interact utilizing **flows**, **pipelines**, and **routers**, as introduced in Lab 8. These concepts enable the creation of event-driven, structured workflows that react to asynchronous triggers, handle complex control flows, and maintain stateful data across multiple agents.

A **flow**, as presented in Lab 8, serves as the overarching framework to coordinate the execution of multiple agents and tasks. It uses decorators such as `@start` and `@listen` to define the sequence of execution. Within the same flow, **routers** allow branching logic based on the outputs of certain agents. By applying these tools, we can implement conditional logic (e.g., route selection based on data) and parallel execution of distinct tasks, ultimately merging their results into a coherent whole.

The **pipeline** metaphor is used to describe how outputs from one agent flow into subsequent agents or groups of agents. Meanwhile, **routers** serve as decision points within these pipelines, directing information to one or more downstream branches based on the current state or the data produced by previous agents. This approach maintains clarity in complex scenarios, like coordinating the Firefighter and Medical Services Crews in parallel, and later merging their outputs for reporting.

Figure 5 illustrates the flow of information and interaction between the agents and crews.

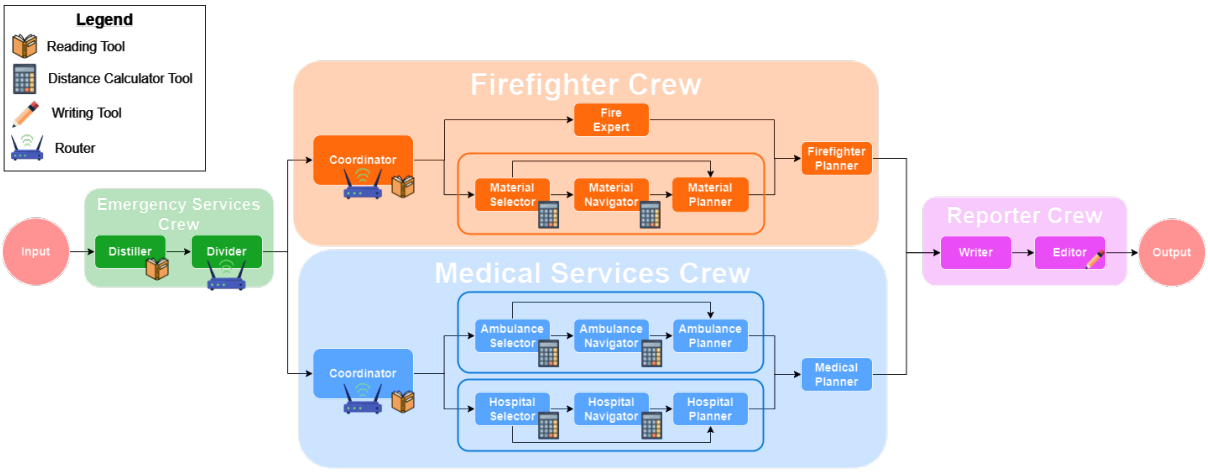

Figure 5: Agent Interaction Diagram

4.1 Agent Interaction Details

The system begins execution through a `@start`-annotated method within a dedicated **main flow**, which initiates the pipelines for the Emergency, Firefighter, Medical, and Reporter Crews. Using `@listen` decorators, subsequent tasks only begin after the agents they depend on have completed. **Routers** are introduced to conditionally direct the flow of information based on the state of the environment, severity of the incident, and crew requirements.

4.1.1 Emergency Crew

Pipeline (with a Router):

1. **Distiller Agent (Start of Pipeline)**: A `@start` method in the flow triggers the Distiller Agent to process the input call. This agent employs language processing to distill the raw input into structured, actionable information. The distilled output is then returned to the flow.

2. **Divider Agent (Router)**: The flow uses a `@router` method to determine how the Divider Agent distributes information. For example, if the processed call includes data about fires, injured individuals, or both, the router will direct fire-related details to the Firefighter Crew pipeline and medical-related details to the Medical Services Crew pipeline. This conditional logic ensures only relevant information flows downstream to each specialized crew.

4.1.2 Firefighter Crew

Pipeline with Parallel Branches:
Once the Divider Agent routes fire-related data to the Firefighter Crew, the flow triggers a `@listen` method connected to the Coordinator Agent. The Coordinator Agent itself functions as a router: it reads state information (e.g., from a JSON file) and determines how to proceed.

1. **Coordinator Agent (Router)**: The Coordinator Agent merges incoming fire-related data with internal state and routes distinct subsets of information down two parallel branches. For example, fire characteristics might be sent to a Fire Expert Agent, while logistical data regarding vehicles and equipment are sent to the Material Selection branch. This uses the router pattern to ensure that each branch receives precisely the information it needs.

2. **Parallel Branches:**

   (a) **Branch 1 (Fire Expertise)**:

   - **Fire Expert Agent**: Receives specialized fire data and classifies the fire, deciding how many and which type of firefighters to deploy. Once completed, a `@listen` decorator in the flow triggers the next step—passing its recommendations to the Firefighter Planner Agent.

   (b) **Branch 2 (Material Logistics)**:

   - **Material Selector Agent**: Takes inputs about fire severity and selects appropriate vehicles and equipment.
   - **Material Navigator Agent**: Once the Material Selector finishes, a `@listen` method begins the Material Navigator's tasks, calculating optimal routes for fire trucks.
   - **Material Planner Agent**: Finally, the Material Planner Agent aggregates the outputs from both Selector and Navigator Agents. The flow ensures this execution is sequential: the `@listen` decorators guarantee that the Material Planner runs only when both upstream tasks have completed.

   After both branches finalize their computations, an `@listen(and_(...))` approach (combining outputs from multiple parallel methods) can ensure that the Firefighter Planner Agent runs only when all necessary information is available.

3. **Firefighter Planner Agent**: Aggregates the entire pipeline's outcomes into a final, structured fire response plan. The planner's output is then passed to the downstream Writer Agent in the Reporter Crew pipeline.

4.1.3 Medical Services Crew

Pipeline with Parallel Branches:
A similar pattern applies to the Medical Services Crew. The Coordinator Agent for this crew, triggered by a `@listen` to the Divider Agent's router output, splits medical-related data into two parallel branches: Ambulances and Hospitals.

1. **Coordinator Agent (Router)**: Integrates state data (e.g., ambulance/hospital availability) with incident details and routes them accordingly.
2. **Parallel Branches:**
   (a) **Branch 1 (Ambulances)**:
   - **Ambulance Selector Agent**: Determines how many ambulances to send.
   - **Ambulance Navigator Agent**: Once selection is done, this agent plans optimal ambulance routes.
   - **Ambulance Planner Agent**: Listens to both Selector and Navigator outputs (using `@listen` and potentially `and_()` logic to ensure both are complete) and unifies them before sending the result onward.

   (b) **Branch 2 (Hospitals)**:
   - **Hospital Selector Agent**: Picks appropriate hospitals based on capacity and distance.
   - **Hospital Navigator Agent**: Plans the best routes to reach the chosen hospitals.
   - **Hospital Planner Agent**: Integrates outputs from the Selector and Navigator Agents, similarly to the Ambulance branch.
3. **Medical Planner Agent**: Aggregates outputs from both Ambulance and Hospital branches. The flow ensures that this final planner runs only after all relevant methods have completed. Its final product is passed downstream to the Writer Agent.

4.1.4 Reporter Crew

Pipeline:
The Reporter Crew receives the merged outputs from both the Firefighter and Medical Services Crews. The flow ensures that the Writer Agent is only triggered (`@listen`) once all relevant information has been produced.

1. **Writer Agent**: Compiles the comprehensive plan from the Firefighter and Medical Services Crews into an integrated report.
2. **Editor Agent**: Once writing is complete, this final agent polishes and refines the textual outcome. The `@listen` decorator ensures that editing only begins when writing is finished, producing a final output ready for delivery.

4.2 Justification of Decisions

The use of **routers** and **flows** is not arbitrary. The complexity of our scenario—handling both fire emergencies and medical crises—naturally lends itself to branching logic and parallel execution.

- **Routers are needed for conditional logic:** The Divider Agent acts as a router to allocate data to the correct crew pipeline. Similarly, the Coordinator Agents within each crew decide how to split tasks into parallel branches. Without these routers, the workflow would become rigid and less adaptable to different types of incidents.
- **Parallel branches reflect real-world concurrency:** In an actual emergency, multiple tasks (e.g., selecting equipment and planning routes) occur simultaneously rather than sequentially. By modeling these as parallel branches in our flow, we accurately capture the concurrency and can later synchronize outputs when all tasks are done. The `@listen(and_(...))` functionality ensures that the final planning agents only proceed once their dependencies complete, maintaining coherence and data integrity.

- **Flows and states manage complexity:** Using flows, we can maintain internal state across agents and methods. This ensures that crucial data (e.g., severity of fire, number of injured individuals, availability of hospitals) are persistently accessible. The structured flow model simplifies debugging, visualization, and comprehension of the entire emergency response pipeline.

References

[1] OpenAI. This image was created with the assistance of dall·e 2. https://openai.com/dall-e-2, 2024. Accessed: 12-10-2024.

[2] Michael J. Wooldridge. *An Introduction to Multiagent Systems*. John Wiley & Sons, 2009.

[3] Nicholas R Jennings and Stefan Bussmann. Agent-based control systems: Why are they suited to engineering complex systems? *IEEE Control Systems Magazine*, 23(3):61–73, 2003.

[4] Gerhard Weiss. *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*. MIT Press, 1999.

[5] Yoav Shoham and Kevin Leyton-Brown. *Multiagent Systems: Algorithmic, Game-Theoretic, and Logical Foundations*. Cambridge University Press, 2009.

[6] Ana Mas. *Agentes software y sistemas multiagente: conceptos, arquitecturas y aplicaciones*. Pearson Prentice Hall, 2005.

[7] Ajuntament de Barcelona. Personal equipment — Firefighters of Barcelona. https://ajuntament.barcelona.cat/bombers/en/personal-equipment. Accessed: 2024-10-14.

[8] Ajuntament de Barcelona. Vehicles — Firefighters of Barcelona. https://ajuntament.barcelona.cat/bombers/en/vehicles. Accessed: 2024-10-14.

[9] Universitat Rovira i Virgili. Lab 05 - leveraging tasks, 2023. Master's Degree in Computer Security Engineering and Artificial Intelligence, Master's Degree in Artificial Intelligence, Multi-Agent Systems.

[10] Universitat Rovira i Virgili. Lab 06 - agent collaboration, 2023. Master's Degree in Computer Security Engineering and Artificial Intelligence, Master's Degree in Artificial Intelligence, Multi-Agent Systems.

[11] Universitat Rovira i Virgili. Lab 08 - flows and routers, 2023. Master's Degree in Computer Security Engineering and Artificial Intelligence, Master's Degree in Artificial Intelligence, Multi-Agent Systems.