Team Software Engineering Project

Progress Report

Dispatcher Agent

01286990 Team Software Engineering Project

Software Engineering Program

Department of Computer Engineering, KMITL

By

66011564 Panisara Yimcharoen

66011147 Phatdanai Khemanukul

66011288 Vichaya Roongsiripornphol

1. Project Description

1.1 Problem Statement

Emergency response systems must handle multiple incidents simultaneously while operating under limited resources such as police cars, ambulances, and fire trucks. Traditional dispatch systems rely on fixed rules or human operators, which can lead to inefficient decisions when incidents compete for resources, urgency changes over time, or unexpected emergencies occur.

There is a need for an intelligent system that can reason dynamically about emergencies, resource availability, urgency escalation, and future system risks in order to make effective dispatch decisions in real time.

1.2 Project Objectives

The objectives of this project are:

- To design a **knowledge-based emergency dispatcher** using Prolog.
- To represent emergency incidents, services, units, and environment knowledge as facts.
- To reason about **urgency, resource availability, and system pressure**.
- To dynamically **dispatch, delay, or reassign emergency units** based on priority.
- To support **non-monotonic reasoning**, allowing decisions to be revised when new incidents occur.
- To provide **explainable reasoning** for every decision made by the system.

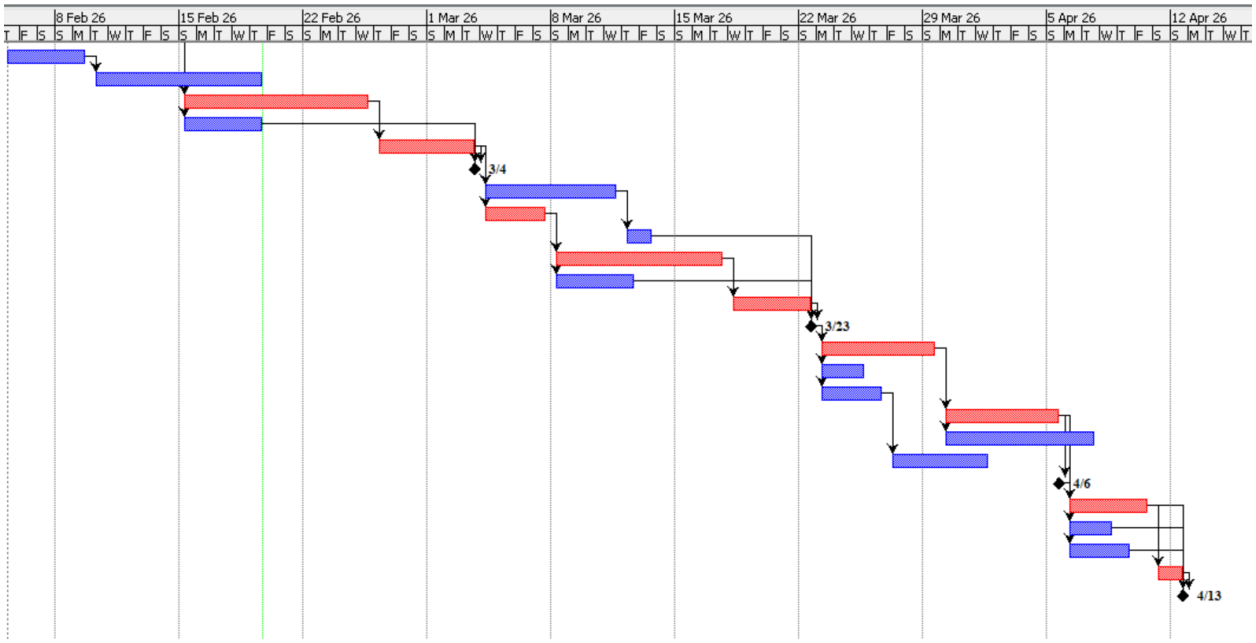
1.3 Project Plan

The project is developed in the following stages:

1. Knowledge representation design (incidents, units, services).
2. Rule development for service selection and unit assignment.

3. Temporal reasoning for urgency escalation.
4. Risk-aware reasoning to prevent system overload.

5. Creating a city environment and travel logic using graph.
6. Integration with a web-based dashboard for visualization.
7. Testing with multiple simulated emergency scenarios.



Name	Start	Finish	Duration	Predecess
Define Incidents & Units Structure	2/5/26 8:00 AM	2/9/26 5:00 PM	5 days	
Define Services & Environment Facts	2/10/26 8:00 AM	2/19/26 5:00 PM	10 days	2
Rule Development for Service Selection & Unit Assignment	2/15/26 8:00 AM	2/25/26 5:00 PM	11 days	1
Service Selection Rules	2/15/26 8:00 AM	2/19/26 5:00 PM	5 days	1
Unit Assignment Rules	2/26/26 8:00 AM	3/3/26 5:00 PM	6 days	4
Milestone: Core Knowledge & Rules Completed	3/3/26 5:00 PM	3/3/26 5:00 PM	0 days	5;6
Temporal Reasoning for Urgency Escalation	3/4/26 8:00 AM	3/11/26 5:00 PM	8 days	6
Urgency Evaluation Logic	3/4/26 8:00 AM	3/7/26 5:00 PM	4 days	6
Urgency Escalation Over Time	3/12/26 8:00 AM	3/13/26 5:00 PM	2 days	8
Risk-Aware Reasoning (System Overload Prevention)	3/8/26 8:00 AM	3/17/26 5:00 PM	10 days	9
System Load Analysis Logic	3/8/26 8:00 AM	3/12/26 5:00 PM	5 days	9
Reassignment & Risk Prevention Rules	3/18/26 8:00 AM	3/22/26 5:00 PM	5 days	11
Milestone: Intelligent Dispatcher Logic Completed	3/22/26 5:00 PM	3/22/26 5:00 PM	0 days	10;12;13
City Environment & Graph Travel Logic	3/23/26 8:00 AM	3/29/26 5:00 PM	7 days	14
Map Representation	3/23/26 8:00 AM	3/25/26 5:00 PM	3 days	14
Graph-Based Travel & Distance Logic	3/23/26 8:00 AM	3/26/26 5:00 PM	4 days	14
Web Dashboard Integration	3/30/26 8:00 AM	4/5/26 5:00 PM	7 days	15
Backend–Prolog Integration	3/30/26 8:00 AM	4/7/26 5:00 PM	9 days	15
Frontend Visualization	3/27/26 8:00 AM	4/1/26 5:00 PM	6 days	17
Milestone: System Integration Completed	4/5/26 5:00 PM	4/5/26 5:00 PM	0 days	18
Simulated Emergency Scenario Testing	4/6/26 8:00 AM	4/10/26 5:00 PM	5 days	18
Single Incident Testing	4/6/26 8:00 AM	4/8/26 5:00 PM	3 days	18
Multiple Concurrent Incident Testing	4/6/26 8:00 AM	4/9/26 5:00 PM	4 days	21
System Optimization & Bug Fixing	4/11/26 8:00 AM	4/12/26 5:00 PM	2 days	22
Milestone: Project Completed	4/12/26 5:00 PM	4/12/26 5:00 PM	0 days	22;23;24;25

2. Software Requirements

2.1 User Functional Requirements

1. The user must be able to report a new emergency incident by selecting a location on the map.

2. The user must be able to specify the type of emergency incident (e.g., accident, fire, medical).

3. The user must be able to specify incident details including severity and initial urgency.
4. The user must be able to submit an incident to the system for automated handling.

5. The user must be able to view which emergency service is assigned to an incident.

6. The user must be able to monitor the status of emergency units (available, dispatched, busy).

7. The user must be able to observe changes in urgency over time.

8. The user must be able to see when an incident is delayed due to low priority.

9. The user must be able to see when a unit is reassigned to a higher-priority incident.

10. The user must be able to view system-generated explanations for dispatch decisions.

2.2 User Non-Functional Requirements

1. The system shall provide the user interface and instructions in English.

2. The system shall be accessible through modern web browsers on desktop devices.

3. The system shall respond to user actions, such as reporting incidents, within an acceptable response time (less than 2 seconds under normal load).

4. The system shall display all time-related information, such as incident creation time and urgency updates, using the local system time.

5. The system shall update incident and unit status in real time to reflect current system conditions.

6. The system shall maintain stable operation during continuous usage without unexpected crashes.

7. The system shall remain usable under multiple simultaneous incidents without significant performance degradation.

8. The system shall ensure that information presented to the user accurately reflects the current knowledge base state.

9. The system shall be available for user interaction at all times.

10. The system shall present information in a clear and readable manner to minimize user interpretation errors.

2.3 System Functional Requirements

1. The system shall maintain a knowledge base containing emergency incidents, units, services, and environment data.

2. The system shall automatically register new emergency incidents submitted by the user.

3. The system shall infer the required emergency services for each incident based on incident attributes.

4. The system shall evaluate incident urgency and update it dynamically over time.

5. The system shall determine unit availability based on current assignments and workload.

6. The system shall select appropriate emergency units using knowledge-based reasoning.

7. The system shall dispatch selected units and update their operational status accordingly.

8. The system shall re-evaluate dispatch decisions when new higher-priority incidents occur.

9. The system shall update the system state continuously to reflect unit movement and incident handling.

10. The system shall generate structured explanations describing the reasoning behind dispatch decisions.

2.4 System Non-Functional Requirements

1. The system must implement the core reasoning logic using Prolog for knowledge representation and inference.

2. The system must be integrated with a Python-based backend to manage system state and communication with the user interface.

3. The system must use a web-based frontend that is compatible with modern browsers such as Google Chrome.

4. The system must support real-time data updates using WebSocket communication.

5. The system must maintain the knowledge base in memory during runtime to support fast reasoning and decision making.

6. The system must be version controlled locally using Git to track development changes.

7. The system must use GitHub as the remote version control repository to support collaboration and backup.

8. The system must be designed using a modular architecture separating reasoning, system control, and presentation layers.

9. The system must be developed and tested on a display resolution of 1920×1080 (16:9) to ensure consistent user interface layout.

10. The system must be documented using UML diagrams such as use case and class diagrams to support system understanding and maintenance.

3. System Architecture

Architectural Overview

The system is designed using a **layered architecture** that separates user interaction, system control, and knowledge-based reasoning. This separation ensures that the emergency dispatcher agent can reason autonomously while allowing users to interact with the system through a web-based interface.

At a high level, the system consists of three main components:

1. **User Interface Layer**
2. **System Control Layer**
3. **Knowledge-Based Reasoning Layer**

Each component has a clearly defined responsibility, enabling modular development and future extensibility.

3.1 User Interface Layer

The User Interface Layer is responsible for all interactions between the user and the system.

Its main responsibilities include:

- Displaying a graph-based city environment consisting of road intersections (nodes), road segments (edges), and buildings connected to the road network.
- Rendering the road network using node coordinates and edge geometry, and animating emergency units along computed shortest paths in real time.
- Allowing the user to report new emergency incidents by selecting a location and specifying incident details
- Visualizing active incidents, emergency units, and their statuses
- Presenting system-generated explanations for dispatch decisions
- Displaying system load and resource availability information

The user interface visualizes the geometric representation of the city using node coordinates, while path computation and travel time calculations are handled by the backend system.

This layer does not contain any decision-making logic. All information displayed to the user is derived from the system's internal state and reasoning outcomes.

3.2 System Control Layer

The System Control Layer acts as the intermediary between the user interface and the reasoning engine.

Its responsibilities include:

- Receiving incident data submitted by the user
- Managing the current system state, including incidents and unit statuses
- Communicating with the knowledge-based reasoning layer
- Triggering re-evaluation of decisions when system conditions change
- Maintaining a graph data structure representing intersections, buildings, and road segments, including edge weights corresponding to travel distance.
- Synchronizing real-time updates between the backend and frontend

This layer ensures that user input is properly translated into system facts and that reasoning results are consistently reflected in the user interface.

3.3 Knowledge-Based Reasoning Layer

The Knowledge-Based Reasoning Layer is the core intelligence of the system.

It is responsible for:

- Representing incidents, emergency units, services, and environment data as facts
- Reasoning about incident urgency and required services

- Selecting appropriate emergency units using knowledge-based rules
- Performing temporal reasoning for urgency escalation
- Supporting non-monotonic reasoning to allow reassignment of units
- Generating structured explanations for dispatch decisions

This layer operates independently of the user interface and focuses entirely on reasoning and decision-making.

3.4 Data and Communication Flow

The interaction between system components follows this flow:

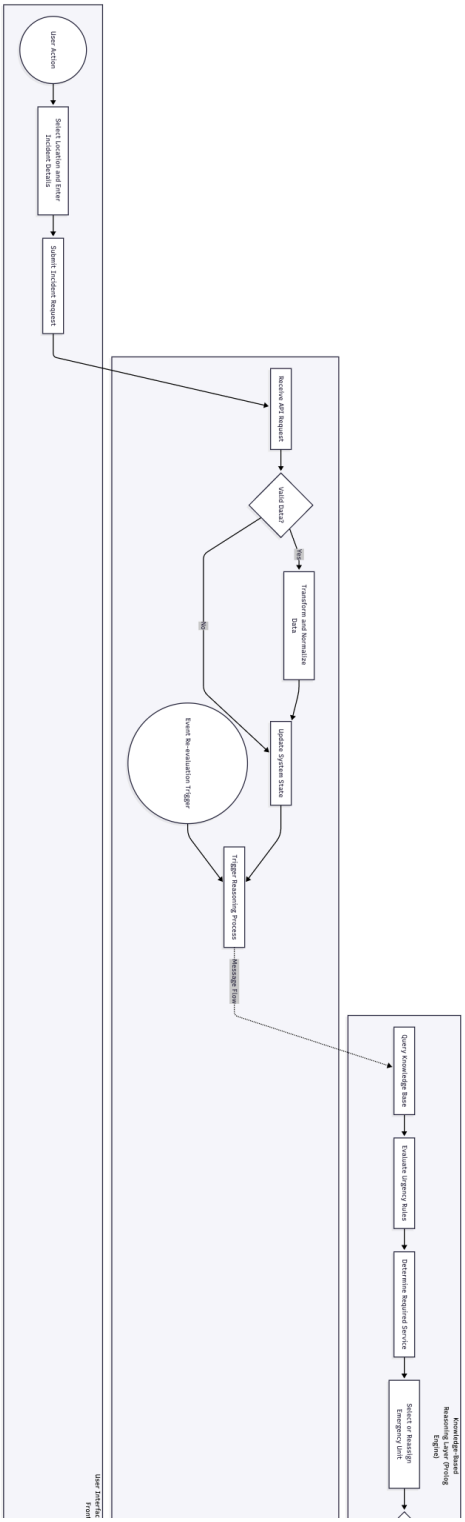
1. The user reports an incident through the user interface.
2. The system control layer processes the input and updates the system state.
3. The knowledge-based reasoning layer evaluates the situation and determines dispatch decisions.
4. The reasoning results and explanations are returned to the system control layer.
5. The user interface is updated in real time to reflect the new system state.

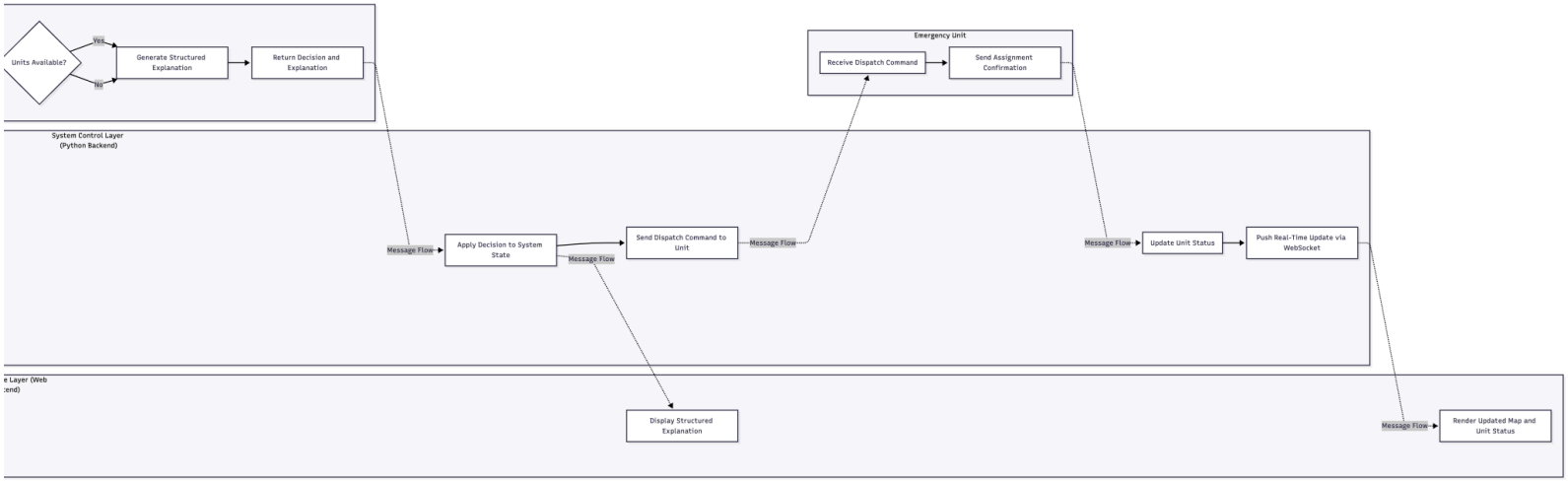
This architecture ensures that all decisions are driven by inferred knowledge rather than hardcoded behavior.

Architectural Justification

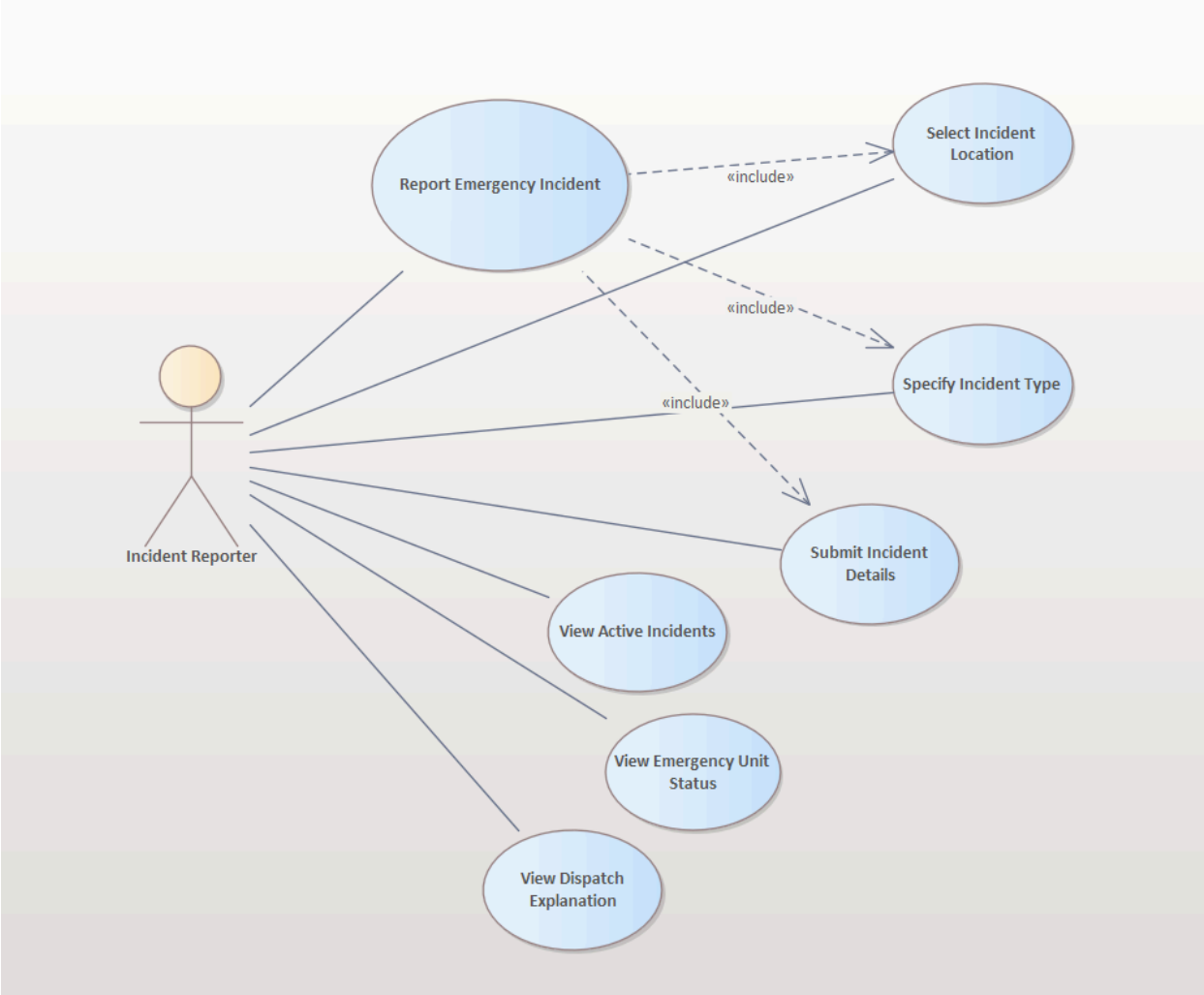
The selected system architecture is designed to clearly separate user interaction, system control, and knowledge-based reasoning. This separation of concerns allows each component to be developed, tested, and modified independently, supporting modular system development. By isolating the reasoning logic from the presentation layer, the dispatcher agent can focus entirely on intelligent decision-making without being affected by user interface changes. The architecture also enables explainable and transparent reasoning, as all dispatch decisions are generated by the reasoning layer and communicated explicitly to the user interface. In addition,

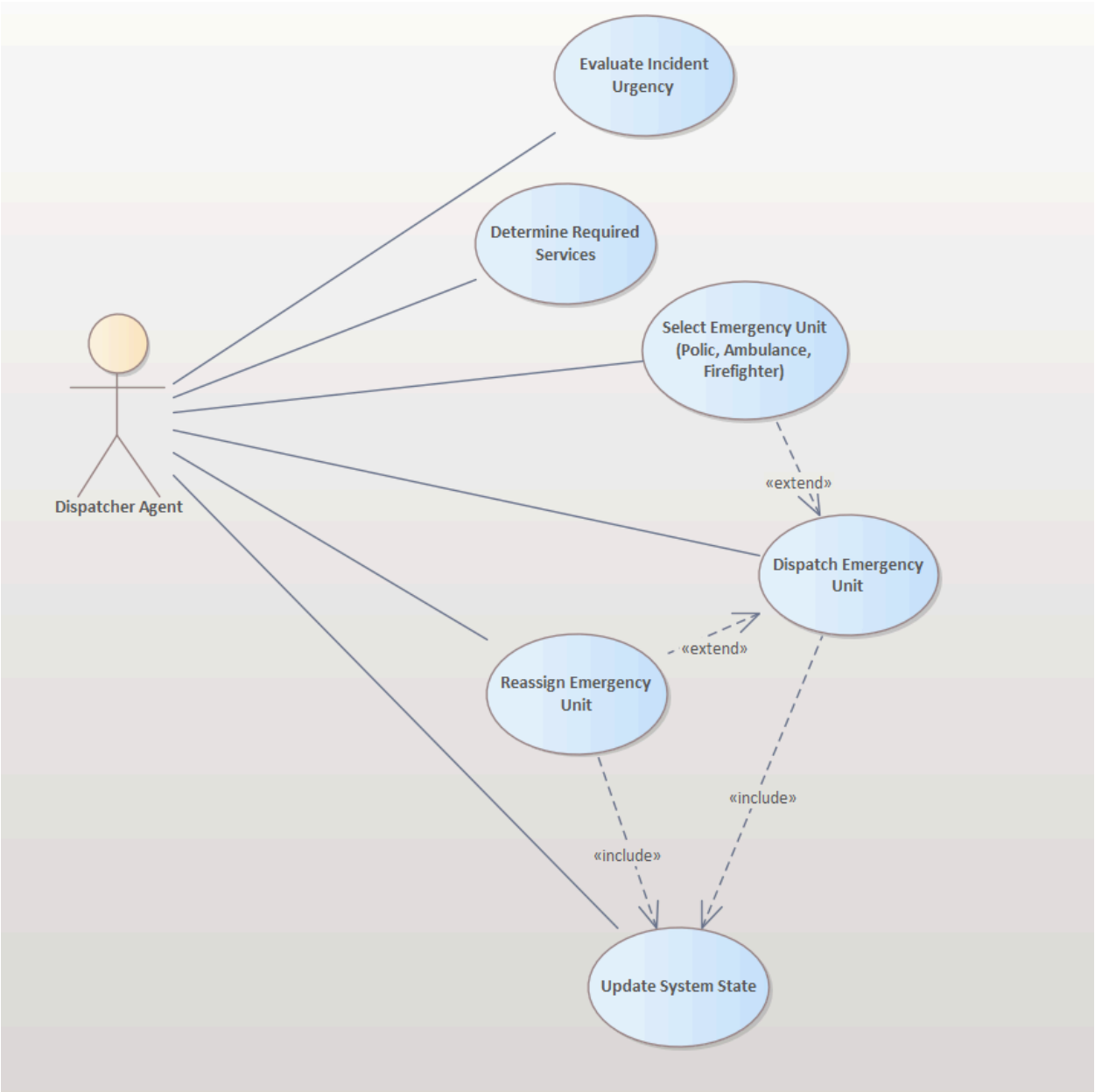
the layered design supports real-time responsiveness by allowing system state updates and reasoning processes to be handled efficiently. Finally, the modular structure allows future extensions, such as additional emergency services or new reasoning rules, to be incorporated with minimal impact on existing system components.




Units Available?
Yes
No
Generate Structured Explanation
Return Decision and Explanation
Emergency Unit
Receive Dispatch Command
Send Assignment Confirmation
System Control Layer (Python Backend)
Message Flow
Apply Decision to System State
Message Flow
Send Dispatch Command to Unit
Message Flow
Message Flow
Update Unit Status
Push Real-Time Update via WebSocket
e Layer (Web :end)
Display Structured Explanation
Message Flow
Render Updated Map and Unit Status

4. Use Case Analysis




Evaluate Incident Urgency
Determine Required Services
Select Emergency Unit (Polic, Ambulance, Firefighter)
Dispatcher Agent
«extend»
Dispatch Emergency Unit
«extend»
Reassign Emergency Unit
«include»
«include»
Update System State

Use Case 1: Evaluate Incident Urgency

Primary Actor

Dispatcher Agent

Description

This use case describes how the dispatcher agent evaluates the urgency level of a reported emergency incident based on its type, initial severity, and elapsed time.

Basic Course of Events

1. A new emergency incident is registered in the system.
2. The dispatcher agent retrieves incident details such as type, location, and reported severity.
3. The dispatcher agent applies urgency evaluation rules.
4. The system assigns an initial urgency level to the incident.
5. The evaluated urgency is stored in the system state.

Alternative Courses of Events

A: Urgency Escalation Over Time

1. The incident remains unresolved for a period of time.
2. The dispatcher agent re-evaluates the incident urgency.
3. The urgency level is increased based on predefined temporal rules.
4. The updated urgency is saved in the system state.

Exceptional Courses of Events

E: Incomplete Incident Information

1. Required incident details are missing or invalid.
2. The dispatcher agent assigns a default urgency level.
3. The system flags the incident for further review.

Use Case 2: Dispatch Emergency Unit

Primary Actor

Dispatcher Agent

Description

This use case describes how the dispatcher agent selects and dispatches an appropriate emergency unit to handle an incident.

Basic Course of Events

1. The dispatcher agent identifies an incident requiring response.
2. The dispatcher agent determines the required emergency service.
3. Available emergency units are retrieved from the system.
4. The dispatcher agent selects the most suitable unit.
5. The selected unit is dispatched to the incident location.
6. The system updates the unit status to “dispatched”.

Alternative Courses of Events

A: Low-Priority Incident

1. The incident urgency is determined to be low.
2. The dispatcher agent delays dispatch.
3. The incident remains in the active incident list.

Exceptional Courses of Events

E: No Available Emergency Units

1. No suitable units are available.
2. The dispatcher agent does not dispatch any unit.
3. The system records the incident as pending.

Use Case 3: Reassign Emergency Unit

Primary Actor

Dispatcher Agent

Description

This use case describes how the dispatcher agent reassigns an emergency unit when a higher-priority incident occurs.

Basic Course of Events

1. A new high-priority incident is detected.

2. The dispatcher agent evaluates current unit assignments.
3. A unit assigned to a lower-priority incident is identified.
4. The unit is reassigned to the higher-priority incident.
5. The system updates the status of both incidents.

Alternative Courses of Events

A: Partial Reassignment

1. Only part of the required service can be reassigned.
2. The dispatcher agent reallocates available capacity.
3. The system updates unit workload information.

Exceptional Courses of Events

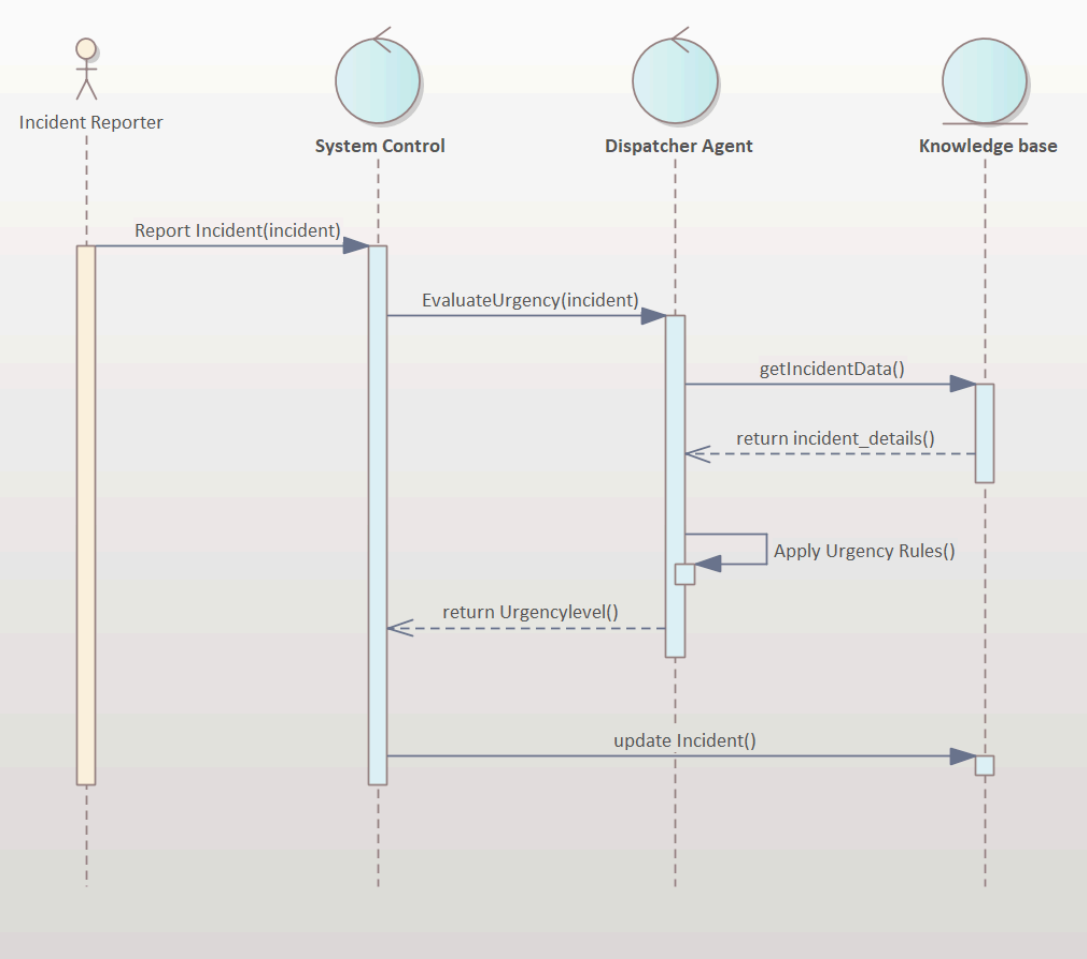
E: Reassignment Not Possible

1. All units are handling equal or higher-priority incidents or reassigning a unit would increase system risk.
2. No reassignment occurs.
3. The high-priority incident is marked as waiting.

5. Sequence Diagram

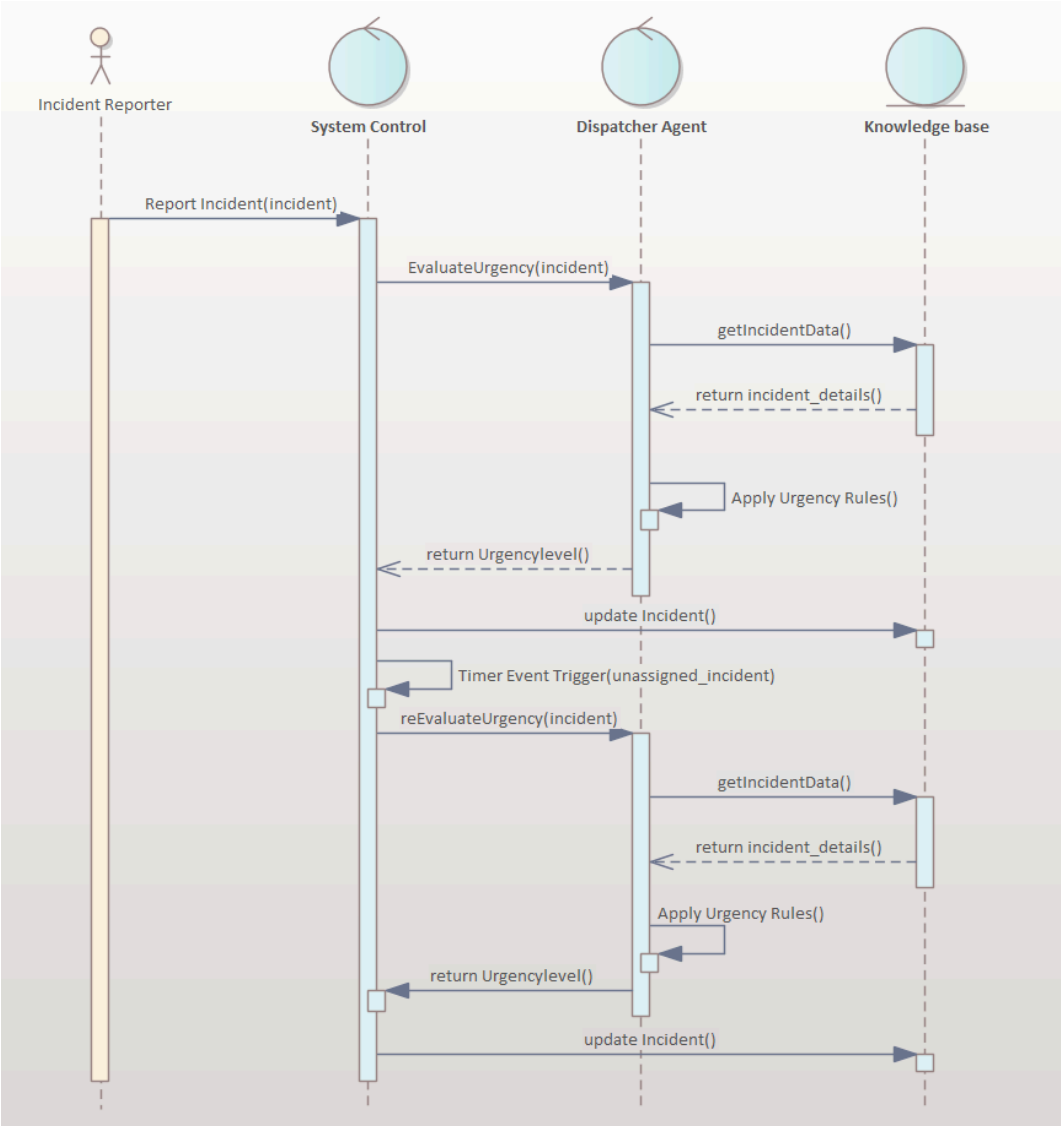
5.1 Evaluate Incident Urgency

5.1.1 Evaluate Incident Urgency: Basic



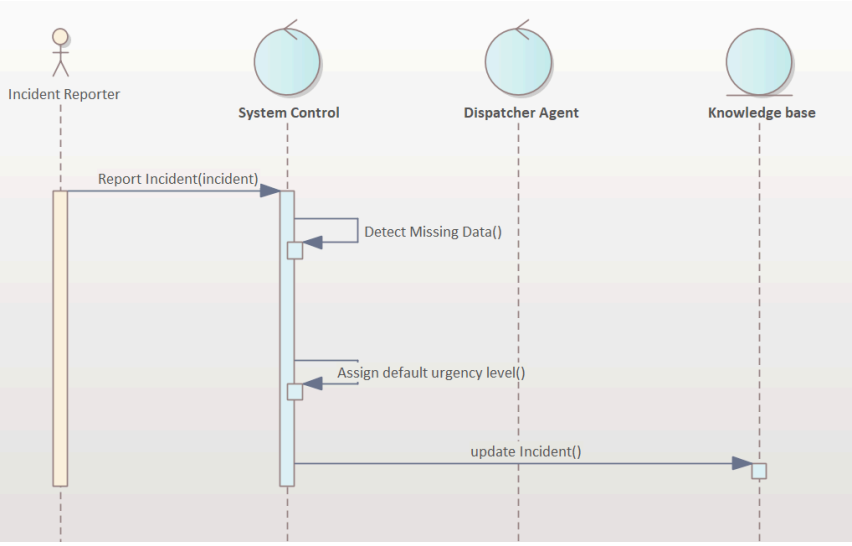
5.1.2 Evaluate Incident Urgency : Alternative

Urgency Escalation Over Time



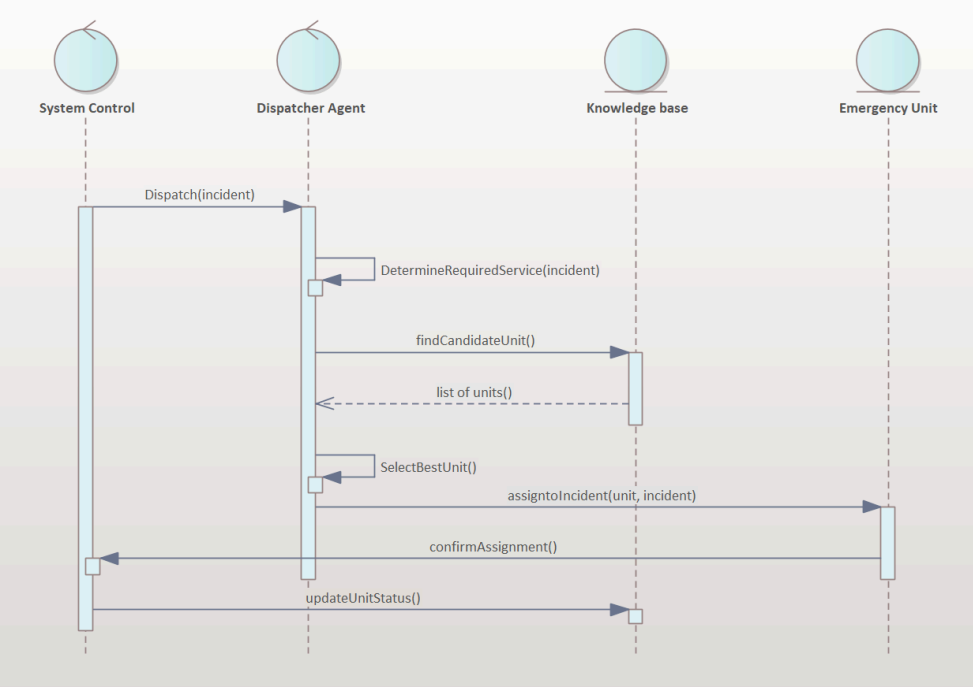
5.1.3 Evaluate Incident Urgency : Exceptional

Incomplete Incident Information

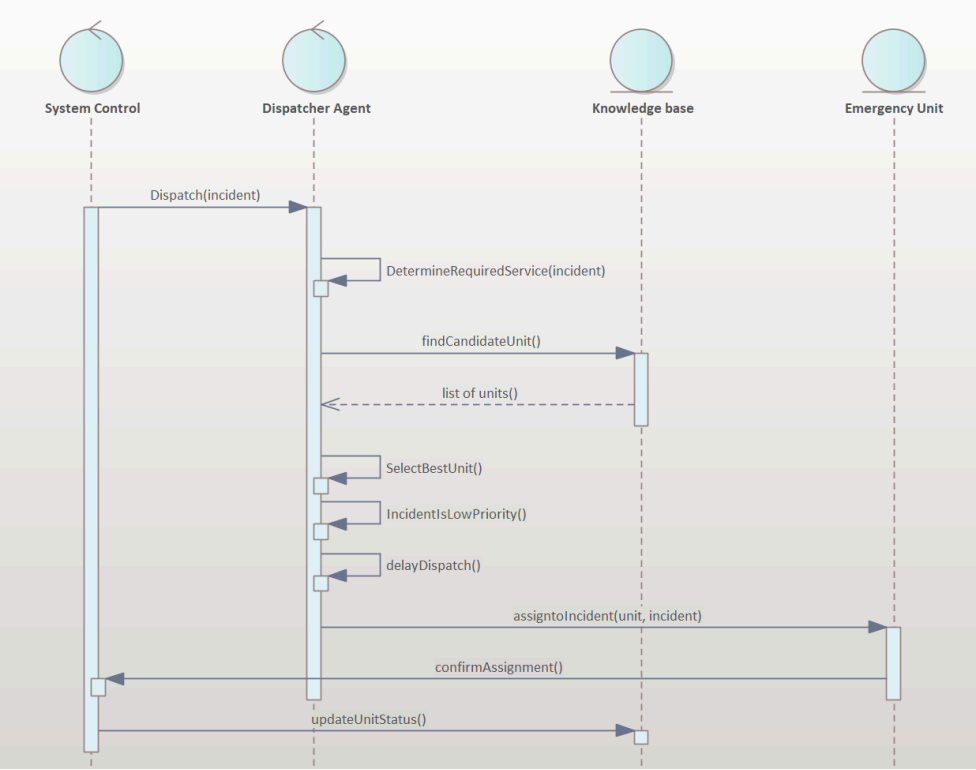


5.2 Dispatch Emergency Unit

5.2.1 Dispatch Emergency Unit: Basic

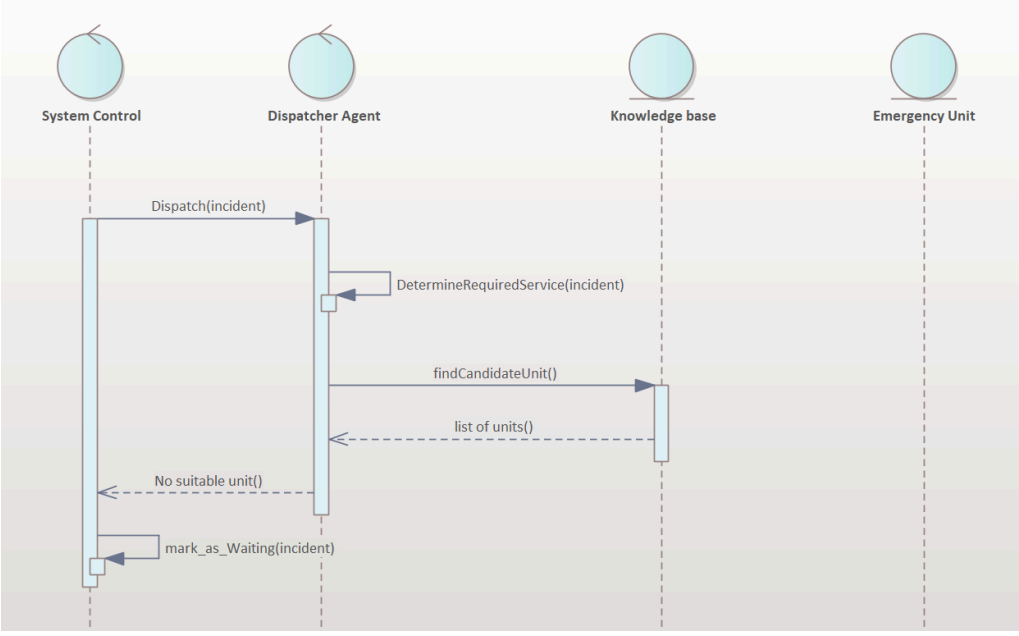


5.2.2 Dispatch Emergency Unit: Alternative



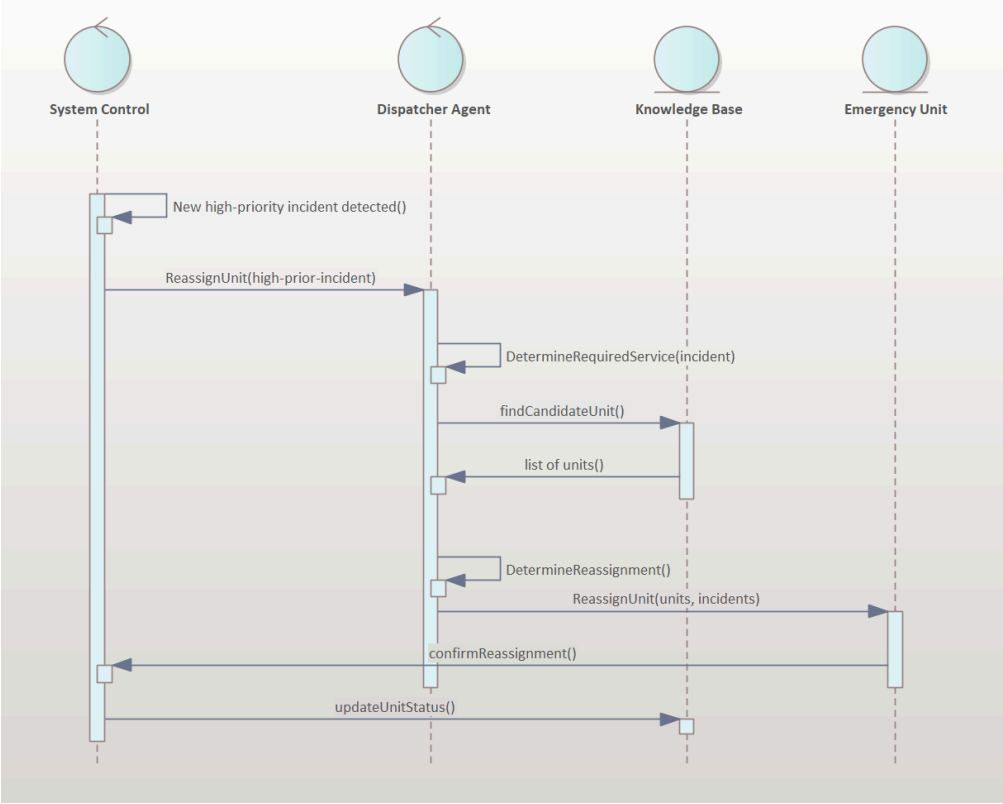
5.2.3 Dispatch Emergency Unit: Exceptional

No Available Emergency Units



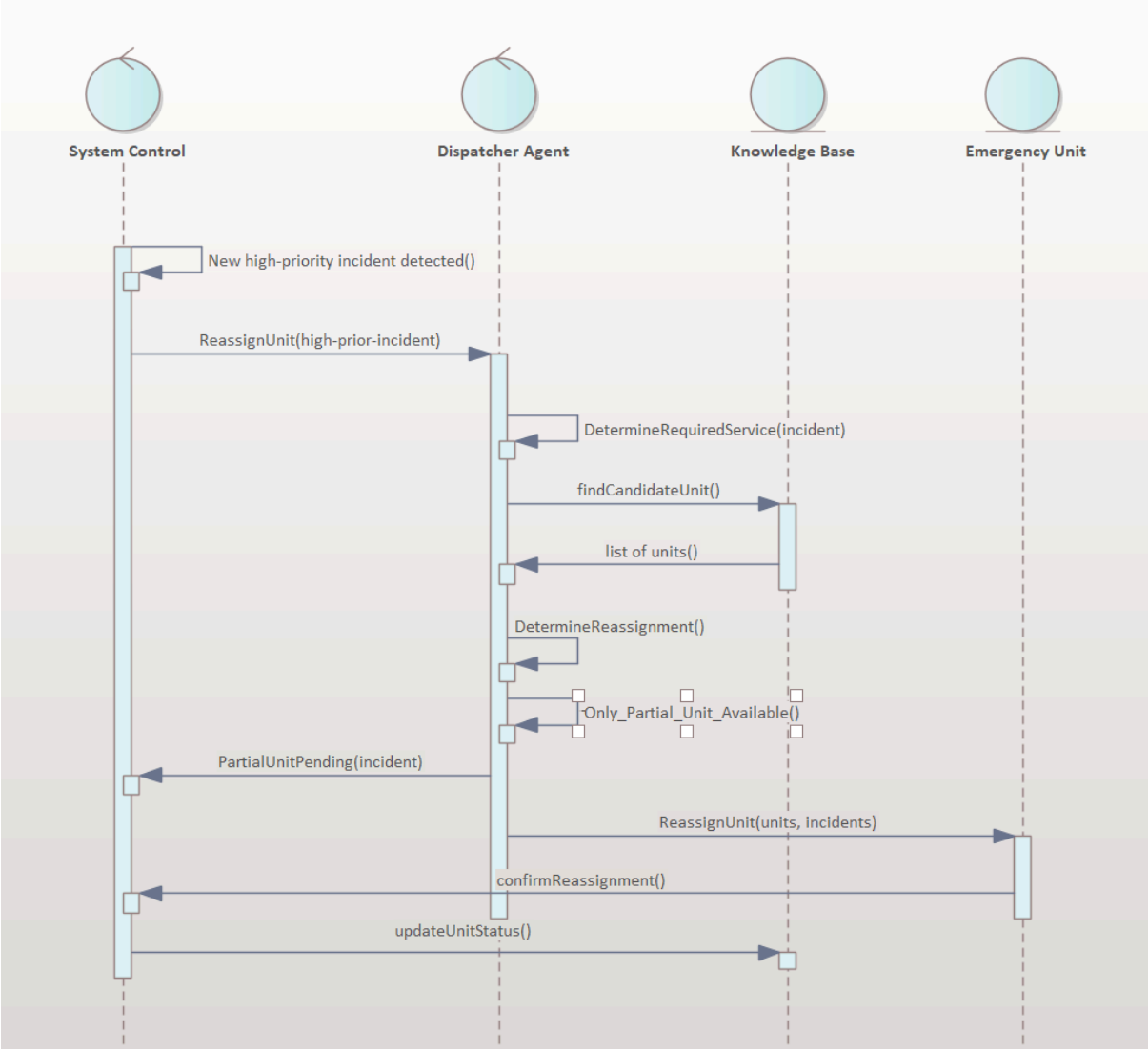
5.3 Reassign Emergency Unit

5.3.1 Reassign Emergency Unit: Basic



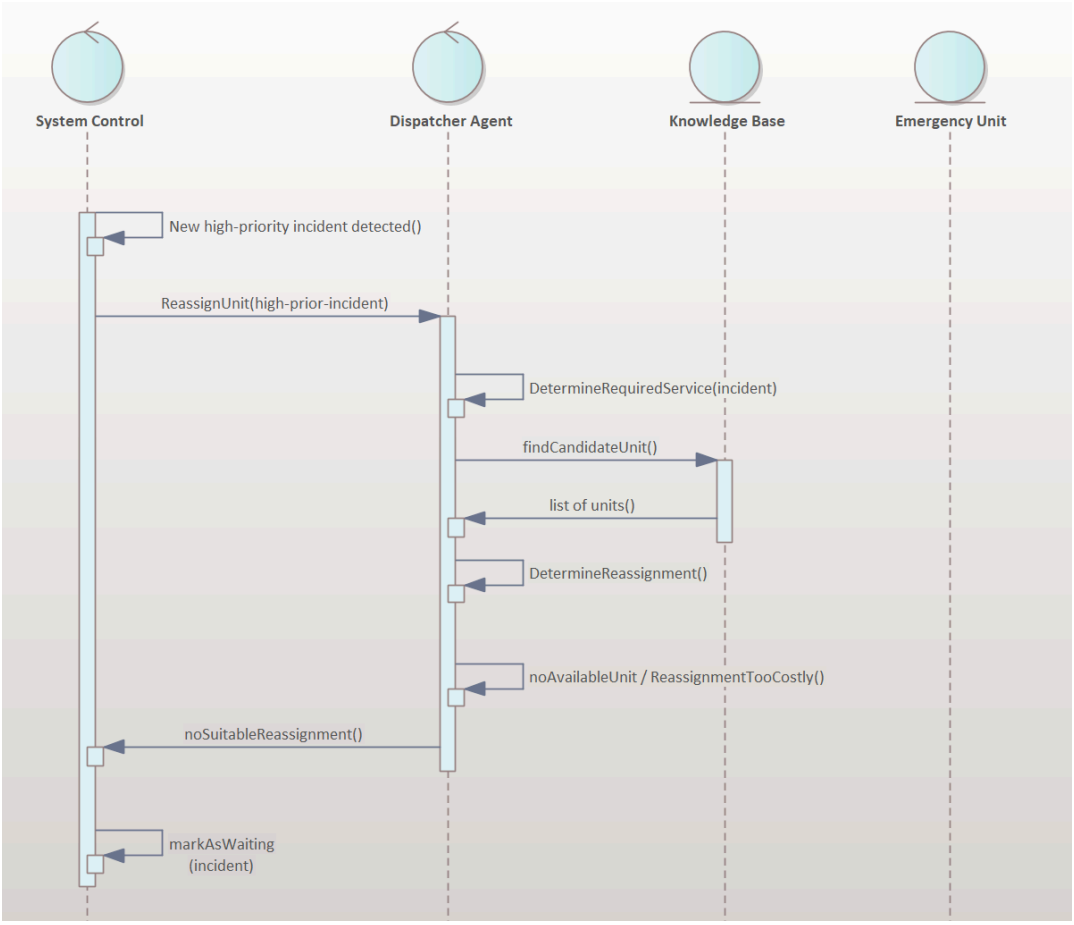
5.3.2 Reassign Emergency Unit: Alternative

Partial Reassignment



5.3.3 Reassign Emergency Unit: Exceptional

Reassignment not possible



6. Class Diagram