Ke Jia – Runzhou Zhu – Yang Yang – Qingyun Wang

Car Service Point

Gasoline station simulation

Metropolia University of Applied Sciences
Information and communication technologies
Software Engineering
Software project
13.12.2024

Table of contents

1 Introduction

This document serves as the foundational description of a simulation project focused on a car service point. The primary goal of this project is to create a model that simulates customer interactions with five key services: a gas station, car wash, drying station, a drive-through store and a cashier. This simulation focuses solely on modelling the queueing behaviour at each service point and does not include financial transactions. The cashier service serves as a unified exit point for all customers after completing their chosen services.

The simulation allows users to choose any individual service, with the exception that the drying station can only be selected after the car wash. Additionally, customers may leave the service point from any service's exit, reflecting the flexibility of real-world scenarios.

The purpose of this document is to provide a detailed overview of the simulation project, its vision, conceptual definitions, and the non-software-specific model to be implemented. It will also serve as a reference for understanding the key assumptions, design, and structure of the simulation.

The content of this document is structured as follows:

- **Visio**: Outlines the end product of the simulation project and the goals it aims to achieve.
- **Concepts, Definitions**: Defines the key terms and elements used throughout the document.
- **Conceptual Model**: Presents the objectives, assumptions, and components of the simulation model.

This document will be updated as the project progresses, reflecting refinements to the model and its implementation.

2 Visio

The goal of this project is to simulate a modular car service point where users can customize their journey based on their specific needs. The simulation replicates five distinct service modules: a gas station, a car wash, a drying station, a store, and a cashier. The cashier acts as a unified point where all customers finalize their service process before leaving the system.

Core features of the simulation include:

- Modular Service Selection: Users can choose any individual service. For example, some may only refuel, while others may opt for a car wash followed by drying.
- Unified Exit Point: All services culminate at the cashier module, ensuring an organized flow and a clear endpoint for the simulation.
- Realistic Constraint: Some dependencies exist, such as the drying station being accessible only after the car wash.

Service Modules Overview:

- Gas Station: Offers fast refuelling services.
- Car Wash: Includes automated vehicle washing.
- Drying Station: Optional service that follows a car wash.
- Drive-through Store / Drive-through: Allows customers to purchase items quickly without leaving their car.
- Cashier: A unified exit point for all customers to finalize their chosen services.

The simulation aims to provide insights into operational efficiency, customer satisfaction, and optimal service layouts. It also serves as a testing ground for various scenarios, such as peak service times, queue management, and resource allocation.

3 Concepts, definitions

This section defines the key concepts used in the simulation and document to ensure consistent terminology and understanding.

- **Key Concepts:**

1. Service Point
   A location offering specific car-related services. The simulated service point includes five distinct modules: gas station, car wash, drying station, drive-through store and cashier. Each module functions independently but allows sequential interactions.
2. Customer
   A user of the service point who selects one or more services based on their preferences and may exit the system at any module. Customers have varying arrival intervals and service requirements.
3. Queue
   The line of customers waiting for a specific service module. Each module has its own queue, which operates on a first-come, first-served basis. Queue length can impact customer satisfaction and service efficiency.
4. Service Time
   The time taken for a service module to complete a customer's request. This may vary depending on the service type and simulation parameters, often modeled using probability distributions such as exponential or normal distributions.
5. Simulation Time
   The total time span over which the simulation runs, during which customer arrivals, service completions, and queue dynamics are tracked.
6. Idle Time
   The period when a service module is not actively serving customers. Idle time impacts the overall utilization rate of a module.
7. Arrival Rate
   The frequency at which customers arrive at the service point. Typically expressed as the number of arrivals per unit of time. Arrival rates can be uniform or fluctuate over time.
8. System State

A snapshot of the system at any given time, including the number of customers in queues, the status of each service module (idle or busy), and ongoing service times.

- **Key Definitions**

1. Service Dependency
   A restriction where certain services can only be accessed after completing others. For example, the drying station is only available after the car wash.
2. Throughput
   The number of customers successfully served within the simulation time.
3. Utilization Rate
   The percentage of time a service module is actively serving customers during the simulation.
4. Queue Length
   The number of customers waiting in line for a service module at a given time.
5. Customer Satisfaction
   A qualitative measure of how well the service point meets customer expectations, often influenced by waiting times and service quality.
6. Event-Based Simulation
   A simulation approach where system changes are driven by discrete events such as customer arrivals or service completions.

4 Conceptual model

This chapter presents the non-software-specific description of the simulation model, detailing its objectives, inputs, outputs, assumptions, and simplifications. The conceptual model outlines the framework for the system that will be implemented in the simulation.

4.1 Objective

The main objective of this simulation is to model the operations of a car service point, where users can select from five service modules: a gas station, car wash, drying station, store, and cashier. The goal is to analyse queue dynamics, optimize service time, and improve customer satisfaction by simulating various scenarios such as varying customer arrival rates, queue lengths, and service times.

4.2 Feeds

The simulation receives various inputs that influence the system's operation. Key inputs include:

- Number of Service Points: The number of stations available for each service module (e.g., gas pumps, wash bays).
- Customer Arrival Rate: The rate at which customers arrive at the service point. It can vary depending on the time of day or external factors.
- Service Time Distributions: The time required to serve a customer at each service point, typically modelled using a probability distribution (e.g., exponential, normal).
- Queueing Rules: The method for managing queues at each service module, such as first-come, first-served (FCFS).
- Capacity of Service Points: The maximum number of customers that can be served simultaneously at each station.

4.3 Printouts

The outputs of the simulation provide valuable performance metrics that reflect how well the system is functioning. Key outputs include:

- Utilization Rate: The percentage of time each service point is in use during the simulation. This helps identify any underutilized or overburdened modules.
- Throughput: The total number of customers served by the system within a given simulation time.

- Queue Length: The average number of customers waiting in line at each service point. Long queues may indicate inefficiencies.
- Waiting Time: The average time customers spend in the queue before receiving service.
- Service Time: average service time in the service point.

4.4 Content

This model simulates the flow of customers through various service modules at the car service point. The model's boundaries are defined by the service modules that make up the service point, and the queues at each of these modules.

The limits of the model:

The model focuses on simulating customer interactions with the five service modules—gas station, car wash, drying station, drive-through store and cashier. It does not account for certain real-world issues such as:

- Staffing and resource allocation: The model does not simulate the effect of having insufficient staff or resources in certain modules, which could affect the speed and quality of service.
- Traffic flow and parking: The movement of cars within the service point area (e.g., how they park or how traffic congestion affects service time) is not included in the model.
- Real-world interruptions: Events like machine breakdowns, customer complaints, or external factors (e.g., weather) that could delay or stop services are not simulated.
- Environmental and safety factors: Real-world safety regulations, such as safety measures at the car wash or drying station, are not modelled.
- No monetary transactions: The project does not include real-world monetary transactions, which means the model cannot simulate overall revenue or financial performance.
- No direct customer satisfaction parameter: The model does not provide a direct metric for customer satisfaction based on factors like excessive waiting times, which limits its ability to suggest improvements to system users.

Model detail:

The model is designed to reflect the core operations of the service point, but it simplifies real-world components for ease of simulation. For example:

- Service time distributions: While real-world service times may vary due to various unpredictable factors, the model uses simplified probability distributions (e.g., exponential or normal distributions) to represent these times. This abstraction helps to simulate the flow without getting into the complexities of each individual service process.
- Queue management: The queue management system follows basic rules such as first-come, first-served (FCFS), whereas in reality, certain services may prioritize VIP customers or provide different queue systems (e.g., fast lanes).
- Service modules: Each service module is treated as a single unit with a fixed capacity, though in the real world, the service point may have additional complexities, such as variations in the number of service stations at peak times or during maintenance periods.

4.5 Assumptions and simplications

The model assumes several factors to simplify its implementation and make it computationally feasible. Key assumptions include:

- Infinite Queue Capacity: The model assumes that there is no limit to the number of customers that can wait in the queue, even though real-world systems might have physical capacity constraints.
- Fixed Service Times: In some modules, service times are assumed to be constant or follow a simplified distribution, although in practice, they may vary.
- No Customer Priority: The model assumes that all customers are treated equally, without any priority based on factors such as VIP status.

4.6 Description of the model

The model simulates the interactions between customers and various service modules at the car service point. It is composed of several key components that represent the different aspects of the service point. Each component serves a specific role in the simulation, and their interactions define the system's behavior.

4.6.1 List of components

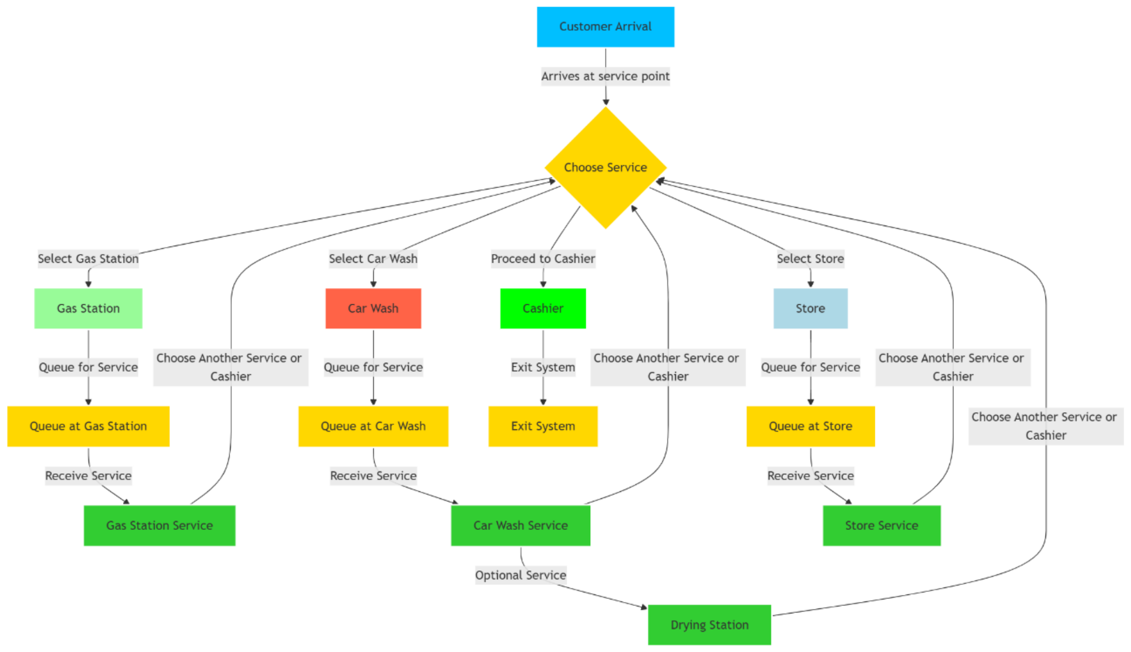
The following table provides a list of the main components that are included in the simulation model. Each component represents a distinct entity or process in the system.

Note that this is a simplified model; real-world components such as clocks, events, and event lists are excluded for clarity.

Component	Features
Customer	Distribution of arrival delays (random intervals).
Car Station	Queue at gas pump, service time distribution (fueling time).
Car Wash	Queue at wash bay, service time distribution (washing time).
Drying Station	Queue after car wash, service time distribution (drying time).
Drive-Through Store	Queue at store, service time distribution (shopping time).
Queue (general)	Infinite capacity by default, but can be limited for specific modules.
Cashier	Final queue and exit point.

Each component has its own set of features that define how it interacts with customers and other components. For example, each service module has its own queue with a specific service time distribution. The queues generally have infinite capacity, except in certain cases where a queue's capacity is limited, such as in the drive-through store, which could be modeled to simulate physical space limitations.

4.6.2 Process diagram



5 Programming implementation of the model

5.1 Programming languages and libraries used

The project is implemented in Java, targeting version 17, and leverages JavaFX for its graphical user interface. The project structure consists of four main folders: Controller, Model, Framework, and View, each responsible for distinct aspects of the simulation. The key libraries and dependencies used are:

- JavaFX:
  - javafx-controls: Provides UI components such as buttons and text fields.
  - javafx-fxml: Supports layout definition using FXML files.
  - javafx-canvas: Enables custom graphics rendering using Canvas and GraphicsContext.
  - javafx-application: Manages application lifecycle and threading.
- Java Standard Library:
  - java.util.ArrayList, HashMap, LinkedList: Core data structures used to manage collections and mappings in the simulation model.
  - java.util.Random: Generates random events, such as customer arrival or service times.
  - java.util.PriorityQueue: Implements the EventList for time-ordered event processing.
  - java.io.BufferedWriter and FileWriter: Writes simulation results to external files for analysis.
  - java.time.LocalDateTime and DateTimeFormatter: Adds timestamps to simulation logs.
  - java.util.concurrent.BlockingQueue: Ensures thread-safe operations for event handling.
- External Libraries:
  - eduni.distributions.Normal: Used for generating random numbers following a normal distribution, simulating variability in service times or arrivals.
- Custom Framework Components:
  - ArrivalProcess, Clock, Engine, Event: Provides essential building blocks for the event-driven simulation, including time management and event handling.

The project uses Maven as the build tool to manage dependencies and configurations. It integrates libraries like Mockito and TestFX for unit and GUI testing (refer to Section 5.6). This combination of JavaFX, custom framework components, and well-structured models ensures a modular and extensible simulation system.

5.2 Architecture

The project adopts an MVC (Model-View-Controller) architecture, ensuring a clear separation of concerns and enhancing maintainability. The architecture consists of the following components:

Model (Logic Layer)

Handles the core simulation logic, including event management, time progression, and service point operations.

Key components:

- MyEngine: Manages the simulation engine, including events and time.
- ServicePoint and Router: Define service points and customer flow.
- Car: Represents customer vehicles and their movement within the simulation.

The Model interacts with the Controller to receive user-defined parameters and update the simulation state.

View (Presentation Layer)

The user interface is defined in an FXML file, providing a structured layout and interactive controls.

Key components:

- Canvas: Displays dynamic visuals of customer vehicles.
- Label: Displays real-time status (e.g., queue sizes, current time).
- Spinner and Slider: Allow users to configure simulation parameters.
- Button: Enables user actions such as starting, pausing, and resetting the simulation.

The View interacts with the Controller through fx:id bindings and event handlers.

Controller (Control Layer)

Bridges the View and Model, handling user interactions and coordinating the simulation logic.

Key responsibilities:

- Initializes simulation parameters from the View and passes them to the Model.
- Updates the UI in real-time, ensuring thread safety using Platform.runLater.
- Manages threads for running simulations and dynamic vehicle rendering.

Framework (Support Layer)

Provides essential utilities and classes for supporting event-driven simulation, decoupling the core simulation logic from lower-level operations.

Key components:

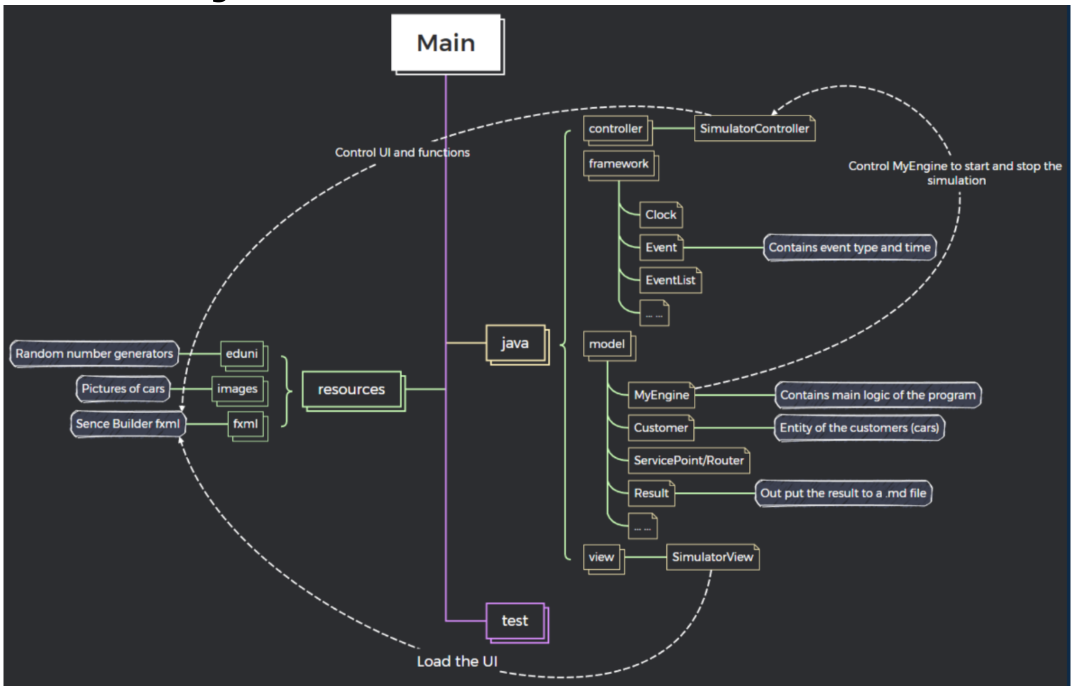
- Clock: Manages simulation time, ensuring synchronization across components.
- EventList: Implements a priority queue to manage and process simulation events in chronological order.
- Trace: Logs simulation events and actions, useful for debugging and monitoring.

The Framework interacts closely with the Model to handle time progression and event scheduling, enabling modularity and reuse.

Interaction Workflow

1. Users configure parameters via the UI and initiate the simulation through the “Start/Continue” button.
2. The Controller initializes the Model and Framework components, such as Clock and EventList, and starts the simulation engine.
3. The Framework manages time and events, which drive the Model's operations.
4. The Model processes events and updates the simulation state, which is reflected in the View through periodic UI updates.
5. Users can pause or reset the simulation, and the Controller adjusts the Model, Framework, and View accordingly.

Architecture Diagram



5.3 Structural description of the user interface

The user interface (UI) of the Gas Station Simulator is implemented using JavaFX. The layout is defined in an FXML file, which is connected to the SimulatorController class. The UI is designed to provide a clear and intuitive interface for controlling and observing the simulation. It includes multiple interactive components for user input and real-time displays of simulation states.

Key UI Sections

The UI is divided into four primary sections:

- Header

  Displays the title of the simulator: "Gas Station Simulator".

  Located at the top of the window using a Label.
- Simulation Area (Center)

  A dynamic visualization area implemented with a Canvas.

  Displays real-time updates of vehicle movements and station states using graphical elements.

  Includes background imagery of the station (ImageView) and labels showing queue sizes and customers served for different service points (e.g., refueling, washing).
- Control Panel (Left)

  Contains a VBox with interactive controls for configuring simulation parameters, including:

    Spinners for setting mean and variance of arrival rates and service times for each station.

    A Slider for controlling the simulation delay.

  Displays current simulation metrics, such as total arrived customers and elapsed time, using Label components.
- Control Buttons (Bottom)

  A horizontal button panel (HBox) for managing the simulation:

    Start/Continue button: Starts or resumes the simulation.

    Pause button: Pauses the simulation.

    Reload button: Resets the simulation to its initial state.

Interaction with the UI

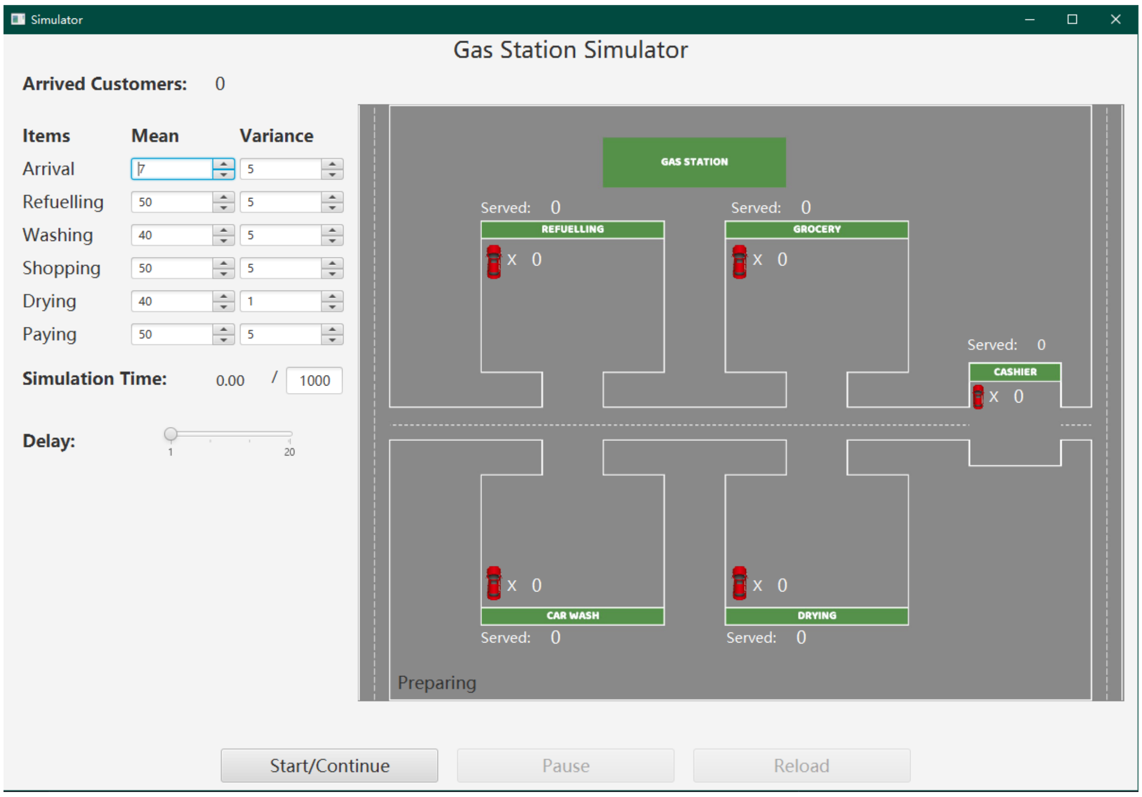
- Simulation Time:

The current time displayed in the UI represents simulation time, managed by the Clock singleton class in the Framework module.
Simulation time progresses based on the event-driven simulation logic rather than real-world time, allowing the simulation to skip idle periods or accelerate operations.
The currentTime label in the UI updates periodically to reflect the value of Clock.get-Clock().

- Real-Time Updates:

Although the simulation itself is not based on real-world time, the UI is updated frequently to reflect changes in simulation state, such as queue sizes and customer counts.

UI Screenshot



Gas Station Simulator

Arrived Customers: 0

Items	Mean	Variance
Arrival	7	5
Refuelling	50	5
Washing	40	5
Shopping	50	5
Drying	40	1
Paying	50	5

Simulation Time: 0.00 / 1000

Delay: 1 – 20

5.4 Description of internal logic

The internal logic of the Gas Station Simulator is based on an event-driven simulation framework. It leverages a virtual simulation clock and a priority queue to manage and execute events in a precise chronological order. This approach ensures flexibility and efficiency by advancing time only when necessary, skipping idle periods.

Event Management

Events in the simulation are represented by the Event class, which encapsulates:

- Event Type: Encoded using the IEventType interface, defining the specific action or operation represented by the event.
- Event Time: Indicates the scheduled time for the event to occur in simulation time.
- Sorting and Priority: Implements the Comparable interface, allowing events to be sorted by their time. This ensures that the event list processes events in the correct chronological order.

The EventList class manages events using a priority queue. Key operations include:

- Add Event: New events are added to the queue using the add() method, maintaining their chronological order.
- Remove Event: The earliest event is removed and processed using the remove() method.
- Peek Next Event: The time of the next event can be queried using the getNextEventTime() method, without removing it.

Simulation Time Management

The simulation operates on a virtual clock (Clock), which is implemented as a singleton to ensure global synchronization.

- Set Time: The current time is updated via Clock.setClock(double time), typically during the execution of events.
- Get Time: The current simulation time can be retrieved using Clock.getClock().
- The Clock ensures all components in the simulation operate on a consistent timeline, facilitating coordinated state transitions.

Simulation Engine

The Engine class serves as the central coordinator for the simulation, managing event execution and time progression. Its key responsibilities include:

- Initialization: Prepares the initial state of the simulation, including scheduling the first event(s).
- Event Execution:
    - phase: Advances the simulation clock to the time of the next event.
    - phase: Processes all events scheduled at the current simulation time using the runBEvents() method. Events are retrieved from the EventList and executed via the runEvent(Event) method, which is implemented in subclasses.

  - phase: Attempts to schedule or process additional events using tryCEvents(), a customizable method in subclasses.
- Termination: The simulation ends when the clock exceeds the user-defined simulationTime.
- Results Collection: Final statistics or results are computed and displayed at the end of the simulation.

Logging and Debugging

The Trace class provides a lightweight logging mechanism, supporting three levels of logging:

- INFO: General informational messages about simulation progress.
- WAR (Warning): Non-critical issues or potential problems.
- ERR (Error): Critical errors affecting the simulation.

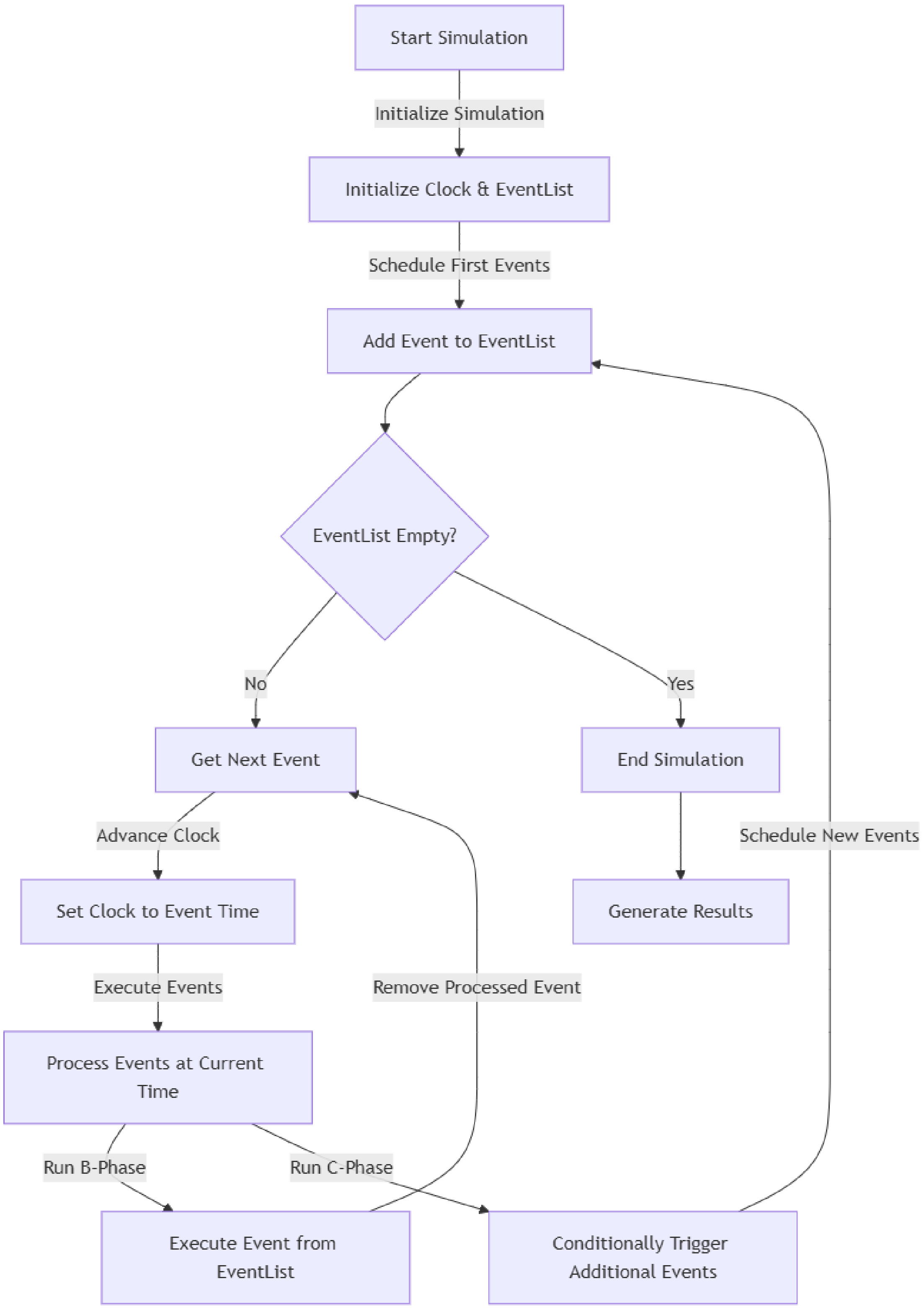
  Logging can be controlled dynamically using Trace.setTraceLevel(Level lvl). All simulation steps and events are logged to the console, aiding in debugging and analysis.

Simulation Phases

The simulation progresses through the following phases:

- Initialization: Sets up initial conditions, schedules the first event(s), and prepares the simulation environment.
- Execution Loop:
  - Time advances to the next event (A-phase).
  - All events scheduled for the current time are executed (B-phase).
  - Additional events are conditionally triggered (C-phase).
- Termination: Ends when the simulation time exceeds the user-defined limit or all events are processed.
- Results Generation: Computes and displays metrics such as total customers served and average waiting times.

Logic diagram



5.5 Descriptions of external data repositories

The simulation results are stored locally in a Markdown file. This file includes detailed metrics and derived statistics, enabling users to review the performance of the gas station simulation after execution. No external databases or advanced data repositories are utilized, ensuring simplicity and ease of use.

- Purpose: The results file provides a summary of key metrics, including the number of customers served, waiting times, service point utilization, and throughput.
- Implementation:
  The Result class generates the file using Java's BufferedWriter and FileWriter classes. The file is named results.md and is overwritten each time the simulation is run.
- File Format:
  Markdown (.md), a lightweight format that is both human-readable and easily renderable in Markdown viewers.
  Example file content includes:
    Directly Observable Variables (e.g., total customers served, busy time).
    Derived Metrics (e.g., service point utilization, average queue length).

Limitations

- Static Format: The current implementation stores results in Markdown format only, limiting compatibility with other tools (e.g., spreadsheets).
- Scalability: For simulations with very large data sets, file-based storage may become inefficient compared to database solutions.

5.6 Testing

The project incorporates a comprehensive testing strategy to ensure the correctness and reliability of its functionality. Each Java file in the project is accompanied by a corresponding test file located in the test directory, ensuring full code coverage.

Testing Framework

The project utilizes the following tools for testing:

- JUnit 5 (JUnit Jupiter): Provides the primary framework for writing and running unit tests.
- Mockito: Used for mocking dependencies where needed to isolate test cases.

Test Coverage

Full Code Coverage: Each Java file in the project has a corresponding test class, ensuring all classes and methods are tested.

Focus Areas:

- Core Logic:

  Tests for the Engine class validate event execution, time progression, and simulation termination conditions.

  Tests for the EventList class verify the correct handling of event insertion, removal, and priority ordering.

- Utilities:

  Tests for the Clock class ensure accurate time management and synchronization.

  Tests for the Trace class verify correct logging behavior based on trace levels.

- Results and Metrics:

  The Result class tests validate correct calculation of metrics (e.g., busy time, waiting time) and proper file generation in Markdown format.

Testing Process

Automated Testing:

All test cases are automated and executed regularly to ensure regression testing.

Tests are run using Maven’s integrated testing lifecycle (mvn test).

6 Simulator user manual

Overview

The Gas Station Simulator allows users to simulate customer flow, queueing, and service times in a gas station environment. Users can configure various parameters, observe real-time updates in the user interface, and analyze detailed results generated after the simulation.

1. Getting Started

1.1. System Requirements

Java Version: Java 17 or higher

Operating System: Windows, macOS, or Linux

Required Tools: Maven for project setup and execution

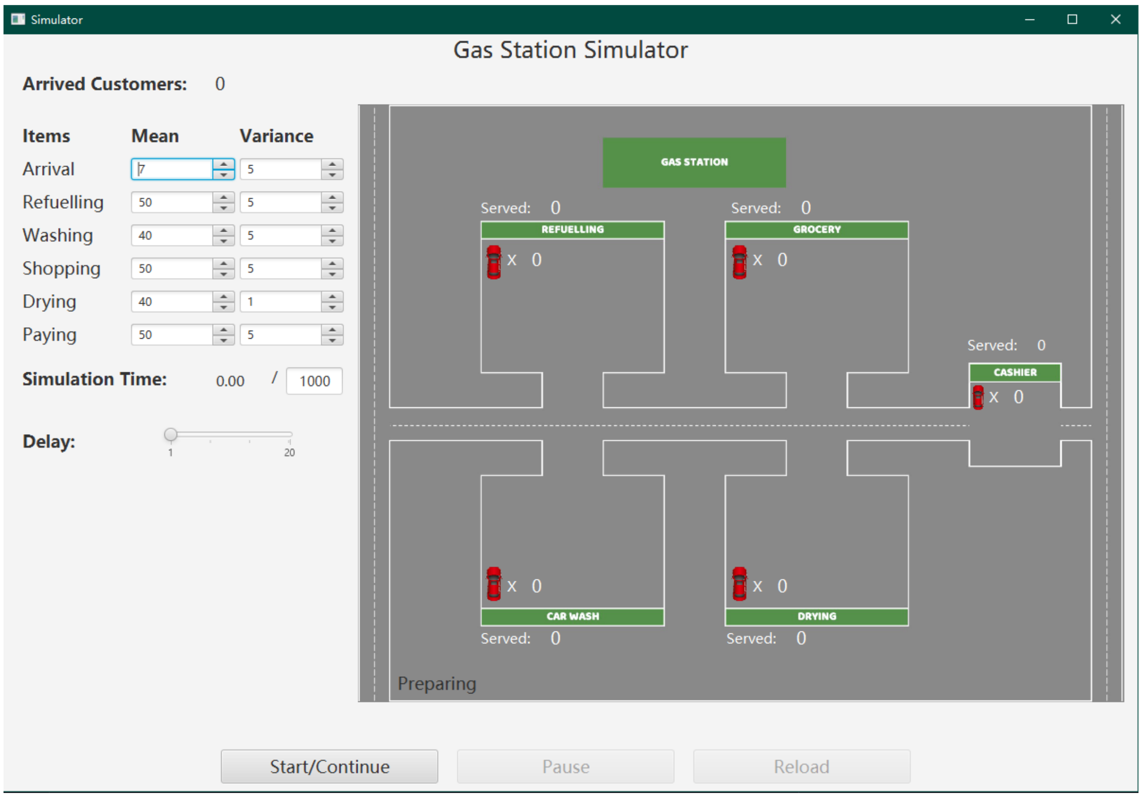
1.2. How to Launch

1) Ensure Java and Maven are installed on your system.
2) Navigate to the project directory in your terminal or IDE.
3) Use the command mvn javafx:run to start the application, or directly run the Main.java file in your IDE.

2. User Interface

2.1. Interface Layout

- Header: Displays the simulator title, "Gas Station Simulator," located at the top of the screen.
- Simulation Area: The central area displays real-time visuals of vehicle movements and service point statuses.
- Control Panel: Positioned on the left, this panel contains spinners and sliders for configuring simulation parameters such as arrival rates and service times.
- Control Buttons: Located at the bottom, this area includes three primary buttons:
  - Start/Continue: Starts or resumes the simulation.
  - Pause: Temporarily halts the simulation.
  - Reload: Resets the simulation to its initial state.



Arrived Customers: 0

Items	Mean	Variance
Arrival	7	5
Refuelling	50	5
Washing	40	5
Shopping	50	5
Drying	40	1
Paying	50	5

Simulation Time: 0.00 / 1000

Delay: 1 – 20

2.2. Controls and Inputs

Spinners:

Adjust the mean and variance for the following parameters:

- Arrival rates
- Refueling times
- Washing times
- Shopping times
- Drying times
- Paying times

Slider:

Modify the simulation delay to control the execution speed (e.g., slow down or accelerate).

Buttons:

- Start/Continue: Begin the simulation or resume after pausing.
- Pause: Temporarily stops the simulation.
- Reload: Clears the current simulation and resets all parameters.

3. Simulation Workflow

3.1. Configuring Parameters

1) Use the left-side control panel to configure:
   - Arrival rate mean and variance using the Arrival spinners.
   - Service time mean and variance for each station (Refueling, Washing, Shopping, Drying, Paying).
2) Adjust the delay slider to set the desired simulation speed.

3.2. Running the Simulation

1) Click the Start/Continue button to begin.
2) Observe vehicle movements and queue statuses in the central simulation area.
3) Use the Pause button to temporarily halt the simulation if needed.

3.3. Observing Real-Time Updates

Monitor:

- Queue sizes at each service point.
- Total customers served.
- Current simulation time.

Updates are reflected dynamically in the simulation area and control panel.

4. Results and Interpretation

4.1. Accessing Results

After the simulation ends, a results file (results.md) is automatically generated in the project directory. This file contains all simulation metrics and derived statistics in Markdown format.

4.2. Understanding the Results

Key sections of the results file include:

- Directly Observable Variables:

  A: Total number of customers who arrived.

  C: Total number of customers served.

  B: Total busy time across all service points.

  T: Total simulation time.
- Derived Metrics:

  U: Service point utilization (B / T).

  X: Service throughput (C / T).

S: Average service time per customer (B / C).

- Additional Metrics:

  W: Total waiting time for all customers.

R: Average response time (W / C).

N: Average queue length (W / T).

4.3. Practical Interpretation

- High Waiting Times (W): Indicates potential bottlenecks; consider increasing service capacity.
- Low Utilization (U): Suggests underutilized resources; adjust arrival rates or service capacity.
- High Throughput (X): Indicates efficient service delivery.

7 Simulation tests carried out

The purpose of the simulation tests is to analyze the behavior of the gas station under varying configurations and scenarios. By adjusting key parameters such as mean service times and their variances, we aim to understand system performance, bottlenecks, and optimization opportunities. The simulator also allows users to indirectly simulate changes in service capacity by modifying the mean values of service times.

7.1 Parameter Design and Assumptions

The simulation uses mean and variance to define the service time distributions at each service point:
Variance:

- For manual services like refueling and shopping, a larger variance (e.g., 25 to 49) is used to simulate wider variability in service durations caused by human factors.
- For automated services like drying, a smaller variance (e.g., 1) is chosen, reflecting consistent service durations with minimal fluctuations.

Mean:

- The mean value represents the average service time per customer for a single resource (e.g., one refueling station).
- Adjusting the mean simulates changes in the number of resources:

For example, a mean of 50 time units assumes one refueling station. Reducing the mean to 25 time units represents the addition of a second station, effectively halving the average service time per customer.

7.2 Test Scenarios

The following test scenarios were designed to evaluate system performance under different configurations. Simulation unit time is 1000.

Scenario1	Arrival	Refuelling	Washing	Shopping	Drying	Paying
Mean	4	50	40	50	40	50
Variance	5	5	4	5	1	5
Scenario2	Arrival	Refuelling	Washing	Shopping	Drying	Paying
Mean	8	50	40	50	40	50
Variance	5	5	4	5	1	5

Scenario3	Arrival	Refuelling	Washing	Shopping	Drying	Paying
Mean	8	12	10	25	20	12
Variance	5	5	4	5	1	5

7.3 Results

The following results were obtained from the simulation runs, highlighting key metrics such as utilization, waiting time, and response time.

	Utilization (U)	Average Queue Length (N)	Average Response Time (R)	Served Clients (C)	Throughput (X)
Scenario1	1.74	40.32	2034.39	268	0.02
Scenario2	1.81	25.11	1270.39	122	0.02
Scenario3	2.35	51.6	750.18	128	0.07

7.4 Observation

Scenario 1:
Under the simulator's default settings (one machine at each service point), setting the customer arrival rate to one every 4 time units significantly exceeds the capacity of the gas station. During the simulation, the queue lengths at several service points reached a staggering 70+, highlighting severe system overload.

Scenario 2:
With the customer arrival rate adjusted to one every 8 time units, the system operated within a reasonable range. However, some service points, such as the car wash, still experienced excessively long queues. This indicates that further adjustments are needed in the next simulation to determine the optimal number of machines required for each service point.

Scenario 3:
After making adjustments based on the observations from the second scenario, the queue lengths at all service points were within acceptable limits in this simulation. The total time required to serve all customers was significantly reduced, demonstrating improved system efficiency.

7.5 Conclusion

The simulation tests highlight the following insights:

1) Impact of Arrival Rates:
   Higher arrival rates significantly increase waiting and response times, emphasizing the importance of resource planning during peak hours.
2) Impact of Service Time Variability:
   Reducing variance (e.g., through automation) improves predictability and reduces delays, even under high utilization.
3) Resource Adjustment:
   Increasing capacity (e.g., reducing mean service time) effectively mitigates congestion, but may lead to underutilized resources during off-peak hours.

8 Summary

The Gas Station Simulator was developed as a flexible tool to model and analyze customer flow, queueing, and service times in a gas station setting. The simulator allows users to configure key parameters, such as arrival rates and service times, and observe system behavior under various conditions. By generating detailed metrics and results, the simulator helps identify system bottlenecks and provides insights for optimization.
Despite its strengths, the simulator has some limitations. It simplifies customer behavior by not considering scenarios where customers abandon queues due to excessive waiting times. Additionally, it does not account for real-world disruptions like machine breakdowns or unpredictable traffic spikes. While it is currently tailored to a gas station environment, expanding its scope to other service systems could significantly broaden its applications.
This project provided us with a valuable opportunity to practice the Java programming skills and JavaFX framework we learned this semester. By designing and implementing the MVC model, we deepened our understanding of event-driven simulation and gained hands-on experience in building user interfaces and managing complex system interactions. The project also enhanced our problem-solving and teamwork skills as we analyzed system performance, refined configurations, and proposed practical improvements.