# CARLETON UNIVERSITY

## Department of Systems and Computer Engineering

### SYSC 5104: Methodologies for Discrete Event Modelling and Simulation

### Assignment 1

**William Wang** **Student ID:** 101191168

## 1. Model Organization

### 1.0 Assignment Github Location

https://github.com/RTGTX7/Airport\_Ground\_Control/tree/main

### 1.1 Atomic Models

1. ​**Landing\_Queue**​: Manages the airport landing queue, ensuring that planes land in order while prioritizing emergency landings. The internal logic follows a ​**First-In-First-Out (FIFO) principle**​, but also considers fuel emergencies for priority landings.
2. ​**Parking\_Queue**​: Manages the scheduling of landed planes, ensuring they are parked in the order they arrive and proceed to subsequent processes (e.g., maintenance checks) after a short stay. It follows a ​**FIFO principle**​, meaning the first plane to arrive is the first to complete parking. Each plane requires a ​**fixed parking time of 5 seconds**​.
3. ​**MaintenanceCheck**​: Receives landed planes and checks for fuel levels or maintenance needs. If maintenance is required, the processing time is extended. Once completed, the plane is output to the system’s external environment or the next stage.

### 1.2 Coupled Model

The core task of this coupled model is to ​**simulate the complete process of an aircraft from landing to maintenance inspection**​. Its responsibilities include:

1. **Receiving incoming planes** → Planes enter the **LandingQueue** to wait for landing.
2. **Managing landing priorities** → Considering **fuel status** and ​**FIFO**​, ensuring emergency landings are handled properly.
3. **Assigning parking spaces** → Once landed, planes move to the **ParkingQueue** for a short stay.
4. **Maintenance check** → Planes proceed to **MaintenanceCheck** for **fuel replenishment** or ​**maintenance inspection**​.
5. **Completing the full process** → Planes that have finished landing, parking, and maintenance are output.

* ​**External Input Port**​:
  + airplane\_in: Receives new arriving planes (Plane).
* ​**External Output Port**​:
  + finished\_plane\_out: Outputs planes that have completed landing and maintenance.

#### 1.2.1 Hierarchical Structure and Coupling Scheme

main.cpp  
├── Landing\_Control.hpp (Coupled Model: Handles Landing Process)   
│ ├── Landing\_Queue.hpp (Atomic Model)  
│ ├── ParkingQueue.hpp (Atomic Model)  
│ ├── MaintenanceCheck.hpp (Atomic Model)

* **plane\_input\_reader** (Input Stream) → LandingQueue
  + Reads aircraft data from an external file and sends plane data to LandingQueue.
* **LandingQueue → ParkingQueue**
  + Once a plane completes landing, it moves to the parking queue.
* **ParkingQueue → MaintenanceCheck**
  + After completing the parking process, the plane moves to the maintenance check phase.
* **MaintenanceCheck → finished\_plane\_out**
  + After maintenance is completed, the plane is output to the external system.

## 2. Formal Specification

### 2.1 Coupled Model：Landing\_Control

<X, Y, {landing\_queue, parking\_queue, maintenance\_check}, EIC, EOC, IC, SELECT>

X = {∅} Y = {final\_out}

EIC = {∅}

EOC = {(maintenance\_check.plane\_out, final\_out)}

IC = {(plane\_input\_reader.out, landing\_queue.plane\_in), (landing\_queue.landed\_out, parking\_queue.plane\_in), (parking\_queue.parking\_done\_out, maintenance\_check.plane\_in)}

SELECT({plane\_input\_reader, landing\_queue, parking\_queue, maintenance\_check}) = plane\_input\_reader

SELECT({landing\_queue, parking\_queue, maintenance\_check}) = landing\_queue

SELECT({parking\_queue, maintenance\_check}) = parking\_queue

### 2.2 Atomic DEVS

##### 2.2.1 Landing\_Queue (Atomic)；

A diagram of a process

AI-generated content may be incorrect.

* Messages received on the input port plane\_in, representing new incoming Plane objects waiting to land.

* **planeQueue**: A std::deque<Plane> storing planes waiting to land.
* **busy**: A boolean variable indicating whether a plane is currently landing.​
* ​: Remaining landing time; if , the system is idle, waiting for input.
* Messages sent to the output port landed\_out, indicating a plane has just completed landing.
* **Trigger Condition**​: When the landing timer expires (internal event occurs).-
* **State Update Rules​**:
  + Remove the **front plane** from planeQueue (this plane has landed).
  + If planeQueue is not empty, set: busy←true, σ←10.0(Next plane starts landing; fixed landing time is 10 seconds).
  + Otherwise, set: busy←false, σ←∞ (No planes are waiting, returning to idle state).
* Trigger Condition​: When a new plane arrives (external event) after time ee has elapsed.
* State Update Rules​:
  + **Subtract elapsed time**​: If , update:
  + **Insert plane into the queue**​: For each incoming plane newPlane received on plane\_in:
    - If newPlane.fuel < 5, use: planeQueue.push\_front(newPlane) (Emergency landing; plane goes to the front of the queue).
    - Otherwise, use: planeQueue.push\_back(newPlane)(Normal queueing order).
  + ​**Start landing if idle**​: If busy= then set: (Start a new landing process).

​

* Trigger Condition: Just before an internal event (when a landing is completed).
* ​Output Rule​:
  + If planeQueue is not empty, send the front plane to the landed\_out port, indicating that it ​**has just completed landing**​.
  + If the queue is empty, there is no output.

,

* If busy is ​true​, σ is usually ​10.0​.
* ​If busy is ​false​, σ is ∞, meaning the system is waiting for input.

##### 2.2.2 Parking\_Queue (Atomic)

A diagram of a system

AI-generated content may be incorrect.

* Messages received on the input port plane\_in, representing planes that have landed and are now arriving at the parking area.
* **planeQueue**: A std::queue<Plane> storing planes waiting for parking processing.
* **busy**: A boolean variable indicating whether a plane is currently being processed in the parking area.
* : Remaining parking service time; if , the system is idle and waiting for a new plane.
* Messages sent on the output port parking\_done\_out, indicating that a plane has just completed its parking process.
* **Trigger Condition**: When the parking service time expires (internal event occurs).
* **State Update Rules**:
  + Remove the **front plane** from planeQueue (this plane has completed parking).
  + If planeQueue is not empty, then set: , (Start processing the next plane, with a fixed parking time of 5 seconds).
  + Otherwise, set: , (No planes waiting; system becomes idle).
* **Trigger Condition**: When new plane(s) arrive (external event) after elapsed time .
* **State Update Rules**:
  + **Subtract elapsed time**: If , update .
  + **Insert new plane(s)**: For each plane received on plane\_in, append it to planeQueue using standard queue insertion.
  + **Start service if idle**: If the system is idle (i.e., ) and planeQueue is not empty, then set: , (Initiate the parking service for the next plane).
* **Trigger Condition**: Just before an internal event (i.e., when a plane’s parking service is completed).
* **Output Rule**:
  + If planeQueue is not empty, output the **front plane** to the parking\_done\_out port, indicating that it has just finished parking.
  + If the queue is empty, there is no output.

,

* **Time Advance Function**:
  + When is true, is the fixed parking time (5.0 seconds).
  + When is false, , meaning the system is waiting for input.

##### 2.2.3 Maintenance\_Check (Atomic)

A diagram of light pot

AI-generated content may be incorrect.

* Messages received on the input port plane\_in, representing incoming Plane objects that require a maintenance check.
* **hasPlane**: A boolean variable indicating whether a plane is currently being checked/maintained.
* **plane**: A Plane object that is currently undergoing maintenance (valid only if hasPlane is true).
* : Remaining maintenance time; if , the system is idle and waiting for a new input.
* Messages sent to the output port plane\_out, indicating that a plane has just completed its maintenance check.
* **Trigger Condition**: When the maintenance timer expires (i.e., an internal event occurs).
* **State Update Rules**:
  + Set **hasPlane** to false (indicating maintenance is completed).
  + Reset to , meaning the system returns to an idle state.
* **Trigger Condition**: When a new plane arrives (external event) after elapsed time .
* **State Update Rules**:
  + **Subtract elapsed time**: If , update .
  + **For each incoming plane** p received on plane\_in:
    - If the system is idle (i.e., **hasPlane** is false), then:
      * Set **plane** to p and **hasPlane** to true.
      * Determine the maintenance duration based on fuel:
        + If p.fuel < 30, set (maintenance needed).
        + Otherwise, set (quick check).
    - If the system is already busy (i.e., **hasPlane** is true), ignore the additional plane (or alternatively, implement a queueing/discard strategy).
* **Trigger Condition**: Just before an internal event (when the maintenance check is completed).
* **Output Rule**:
  + If **hasPlane** is true, output the stored plane object to the port plane\_out, indicating the maintenance check has been completed.

,

* **Time Advance Function**:
  + If **hasPlane** is true, represents the remaining maintenance time (15.0 or 1.0, depending on fuel level).
  + If **hasPlane** is false, , meaning the system is waiting for a new input.

## 3. Experimentation Strategy

In our experimentation, the formal DEVS model is defined as:

where:

* **(X)**: The set of external input events.
* **(Y)**: The set of external output events.
* **(M)**: The collection of atomic submodels, including:
  + **LandingQueue**: Processes incoming aircraft and manages landing order.
  + **ParkingQueue**: Handles post-landing parking operations.
  + **MaintenanceCheck**: Performs post-parking maintenance and fuel checks.
* **(EIC)**: Maps the external input (from an IEStream reading events) to the input ports of the internal models.
* **(IC)**: Defined by the following couplings:
* **(EOC)**: Maps the output of the MaintenanceCheck to the external output of the coupled model.
* **(SELECT)**: Resolves simultaneous events (managed by the simulation framework).

### Test Input Events

The following input events are used in the experiment:

15 Arrival,MU980,China Eastern,Airbus A320,00:03:10,60  
16 Arrival,SQ318,Singapore Airlines,Boeing 747,00:3:40,25  
20 Arrival,UA347,United Airlines,Boeing 757,00:04:05,4  
27 Arrival,JL210,JAL,Boeing 787,00:05:35,10

These events are read by the **IEStream** component and injected into the model at the specified simulation times.

### 3.1.1Test Scenario A: Normal Operations

#### **Events Involved**

* ​**Time 15**​: Arrival of MU980 (Fuel = 60)
* ​**Time 16**​: Arrival of SQ318 (Fuel = 25)

#### **Expected Behavior**

1. **LandingQueue**
   * MU980 enters the queue normally (appended at the back) since its fuel level is sufficient.
   * SQ318 also joins the queue at the back since its fuel is above the emergency threshold.
   * The model processes these aircraft in order, with each landing operation taking ​**10.0 time units**​.
2. **ParkingQueue**
   * Once landing is complete, the aircraft are forwarded to ​**ParkingQueue**​, where each aircraft is parked for a fixed ​**5.0 time units**​.
3. **MaintenanceCheck**
   * After parking, aircraft enter ​**MaintenanceCheck**​.
   * Since both MU980 and SQ318 have fuel levels above 30, they only require a ​**quick check (1.0 time unit)**​.

#### **Observed Results**

* **LandingQueue** updates state correctly, producing expected output (landed\_out).
* **ParkingQueue** and **MaintenanceCheck** update their state sequentially.
* Aircraft are processed without delay, confirming the system operates correctly under normal conditions.

### 3.1.2 Test Scenario B: Emergency Insertion

#### **Event Involved**

* ​**Time 20**​: Arrival of UA347 (Fuel = 4)

#### **Expected Behavior**

1. **LandingQueue**
   * UA347 triggers **emergency insertion** due to critically low fuel.
   * Instead of being appended at the end, UA347 is inserted ​**at the front of the queue**​.
   * UA347 lands **before** previously scheduled aircraft.
2. **ParkingQueue**
   * After landing, UA347 is forwarded to ​**ParkingQueue**​.
3. **MaintenanceCheck**
   * Since UA347’s fuel is critically low (Fuel < 30), it requires ​**extended maintenance (15.0 time units)**​.

#### **Observed Results**

* The simulation log confirms that at ​**Time 20**​, UA347 **jumps ahead** in the queue.
* **LandingQueue** correctly prioritizes emergency cases.
* The **MaintenanceCheck** applies extended processing time for UA347.

### 3.2.1 Integration Testing

Run different simulation experiments, modify the original experimental framework, and demonstrate the model’s response to various inputs (including non-standard inputs).

# Case 1: Invalid time format (Missing seconds)  
10 Arrival,MU980,China Eastern,Airbus A320,00:03,60   
  
# Case 2: Missing flight number  
15 Arrival,,United Airlines,Boeing 757,00:04:05,4   
  
# Case 3: Negative fuel (Impossible value)  
20 Arrival,JL210,JAL,Boeing 787,00:05:35,-10   
  
# Case 4: Out-of-order timestamp (event time is earlier than previous one)  
27 Arrival,BA156,British Airways,Airbus A350,00:02:15,50   
  
# Case 5: Zero fuel (should be emergency but may break logic)  
30 Arrival,AF190,Air France,Boeing 777,00:07:25,0   
  
# Case 6: Corrupt aircraft type (random characters)  
35 Arrival,LH404,Lufthansa,???,00:08:40,30   
  
# Case 7: Missing airline name  
40 Arrival,DL567,,Boeing 737,00:09:55,12   
  
# Case 8: Extremely large fuel value (unrealistic)  
45 Arrival,EK231,Emirates,Boeing 777,00:11:10,999999   
  
# Case 9: Special characters in flight number  
50 Arrival,Q@888,Qatar Airways,Airbus A380,00:12:25,90   
  
# Case 10: Malformed entry (missing fields)  
55 Arrival,AA250,American Airlines,Boeing 737   
  
# Case 11: Overlapping timestamps (multiple arrivals at the same second)  
60 Arrival,CX101,Cathay Pacific,Airbus A321,00:16:10,40   
60 Arrival,KL789,KLM,Boeing 787,00:16:10,15   
  
# Case 12: Random text noise (unstructured input)  
this\_is\_not\_valid\_data  
  
# Case 13: Duplicated entries (testing duplicate detection)  
70 Arrival,AC202,Air Canada,Boeing 737,00:20:05,30   
70 Arrival,AC202,Air Canada,Boeing 737,00:20:05,30   
  
# Case 14: Extreme time value (Beyond 24-hour format)  
80 Arrival,TK700,Turkish Airlines,Boeing 777,25:30:50,20

### 3.2.2✔ Aircraft Landing System - Integration Test Cases & Results

| **Case** | **Input Description** | **Expected Result** | **Actual Model Behavior** | **Pass/Fail** |
| --- | --- | --- | --- | --- |
| **1** | Invalid time format (Missing seconds)10 Arrival,MU980,China Eastern,Airbus A320,00:03,60 | Should be rejected or flagged as an error due to incomplete time format. | Model detected format error and rejected input. | ✔ |
| **2** | Missing flight number15 Arrival,,United Airlines,Boeing 757,00:04:05,4 | Should be rejected or flagged as a warning, requiring complete data. | Warning was generated, and the entry was ignored or marked as invalid. | ✔ |
| **3** | Negative fuel value (Impossible case)20 Arrival,JL210,JAL,Boeing 787,00:05:35,-10 | Should be rejected, as fuel cannot be negative. | Model detected negative fuel and rejected input. | ✔ |
| **4** | Out-of-order timestamp27 Arrival,BA156,British Airways,Airbus A350,00:02:15,50 | Should be flagged for incorrect event sequence, rejected or reordered. | Log shows timestamp order inconsistency, but model did not correct it. | ✘ |
| **5** | Zero fuel (Should trigger emergency logic)30 Arrival,AF190,Air France,Boeing 777,00:07:25,0 | Should trigger emergency landing priority. | Emergency insertion correctly triggered, treating 0 fuel as an emergency. | ✔ |
| **6** | Corrupt aircraft type (Random characters)35 Arrival,LH404,Lufthansa,???,00:08:40,30 | Should either process as an unknown type or flag a warning. | Model processed with a warning and stored as an unknown type. | ✔ |
| **7** | Missing airline name40 Arrival,DL567,,Boeing 737,00:09:55,12 | Should flag incomplete data and require valid input. | Partial missing fields detected, warning issued, but data was partially processed. | ✘ (Data integrity issue) |
| **8** | Extremely large fuel value (Unrealistic case)45 Arrival,EK231,Emirates,Boeing 777,00:11:10,999999 | Should flag as invalid or apply range checks. | Model did not check extreme values and processed an unrealistic number. | ✘ (Lacks numerical range validation) |
| **9** | Special characters in flight number50 Arrival,Q@888,Qatar Airways,Airbus A380,00:12:25,90 | Special characters should be either allowed or flagged based on rules. | Model processed the entry correctly based on system rules. | ✔ |
| **10** | Malformed entry (Missing fields)55 Arrival,AA250,American Airlines,Boeing 737 | Should be rejected due to missing required fields. | Model detected incomplete fields, ignored entry, and issued a warning. | ✔ |
| **11** | Overlapping timestamps (Multiple arrivals at the same second)60 Arrival,CX101,Cathay Pacific,Airbus A321,00:16:10,40​60 Arrival,KL789,KLM,Boeing 787,00:16:10,15 | Should correctly process multiple events with potential order resolution. | Model handled simultaneous arrivals, but order consistency remains uncertain. | ✔ |
| **12** | Random text noise (Unstructured input)this\_is\_not\_valid\_data | Should be rejected outright due to unrecognizable format. | Model detected an invalid entry and ignored it. | ✔ |
| **13** | Duplicated entries (Testing duplicate detection)70 Arrival,AC202,Air Canada,Boeing 737,00:20:05,30(Duplicate entries) | Should detect duplicates and process only once or flag as duplicate. | Model treated both as separate events, failing to detect duplication. | ✘ |
| **14** | Extreme time value (Beyond 24-hour format)80 Arrival,TK700,Turkish Airlines,Boeing 777,25:30:50,20 | Should be rejected due to an invalid timestamp exceeding 24 hours. | Model detected invalid time format and rejected the entry. | ✔ |