

Airspace Management Decision Tool

Validating the Behavior and Structure of Software Design



Kerin Thornton

ENPM 643
System Validation and Verification
Fall 2005

Table of Contents

Introduction	3
Problem Statement	3
Assumptions	3
Airport Schematic	4
Flow of Events	5
Goals, Scenarios, and Use Cases	6
Initial Use Case Diagram	7
Use Case Text and Activity Diagrams	8
Use Case 1. Input Data	8
Use Case 2. Issue Request	9
Use Case 3. Data Processing	10
Use Case 4. Issue Order	11
Generation of Requirements	12
High Level Requirements	12
Requirements Traceability	13
System Structure and System Behavior	17
High Level Structure	17
System Models	18
Structural Design of Program Code	23
System Validation and Verification	24
Labeled Transition System Analyzer	25
Conclusion	31
Appendix	32
MATLAB Code	32
MATLAB Output	34
LTSA Code	36
MATLAB Validation with LTSA—50 iterations	37
References	41

Introduction

Problem Statement

Even at small airports, air traffic controllers must maintain a constant awareness of dynamic situations. With aircraft continuously changing locations, altitudes, and airspeeds, the controller must have a precise idea of exactly where each plane is and an idea of where each is going in order to safely and efficiently control the flow of air traffic. Any misinterpretation of a given situation may lead to a wrong decision, which has the potential for disaster. This intense workload and the demands for 100% of attention 100% of the time can affect the human decision-making processes. The scope of this project is to create a model-based program which air traffic controllers at small airports may use to track aircraft positions and to issue orders to the pilot accordingly, essentially automating the decision-making process.

Consider the case of a shift-change in an air traffic control tower. With a tool such as this one, the outgoing controller has entered into the program the locations of all aircraft currently under his or her control. When the incoming controller inherits these aircraft, all he or she has to do is enter the next incoming pilot request into the program, which outputs appropriate action in accordance with the request. Assuming the controller issues orders according to this output, the program is updated to reflect the new current situation.

This paper follows the development of this decision-making tool, from the generation and mapping of requirements, to the UML diagramming the sequences of events, to the generation of the program code, and finally to the verification and validation of the decision support tool.

Assumptions

The scope of this project will assume the following conditions:

1. The airspace immediately surrounding the airport's holding pattern is of an unlimited capacity. This is where pilots make the request to enter the holding pattern upon approach to land.
2. Only one aircraft may be in the holding pattern at any given time.
3. Only one aircraft may be on the runway at a given time (to land or to take off).
4. The airport owns two taxiways. An aircraft may taxi to the gate at the same time another aircraft is taxiing to the runway.
5. Only one aircraft may be at the gate at a given time.
6. Once an aircraft has taken off, it has immediately cleared the airspace and is no longer under tower control.
7. The air traffic control tower may only receive one request at a time. This is enforced by the communications equipment in the tower and on board the aircraft.

8. Incoming pilot requests reach the controller on a first-come, first-serve basis.
9. The controller may issue an order to a pilot who has not made a request.
10. Once an order has been issued to a pilot, the action is carried out instantaneously.

Airport Schematic

This airport may be described by the drawing below. At any given time there is:

- An unlimited number of aircraft in the surrounding airspace
- No more than one aircraft in the holding pattern
- No more than one aircraft at the gate

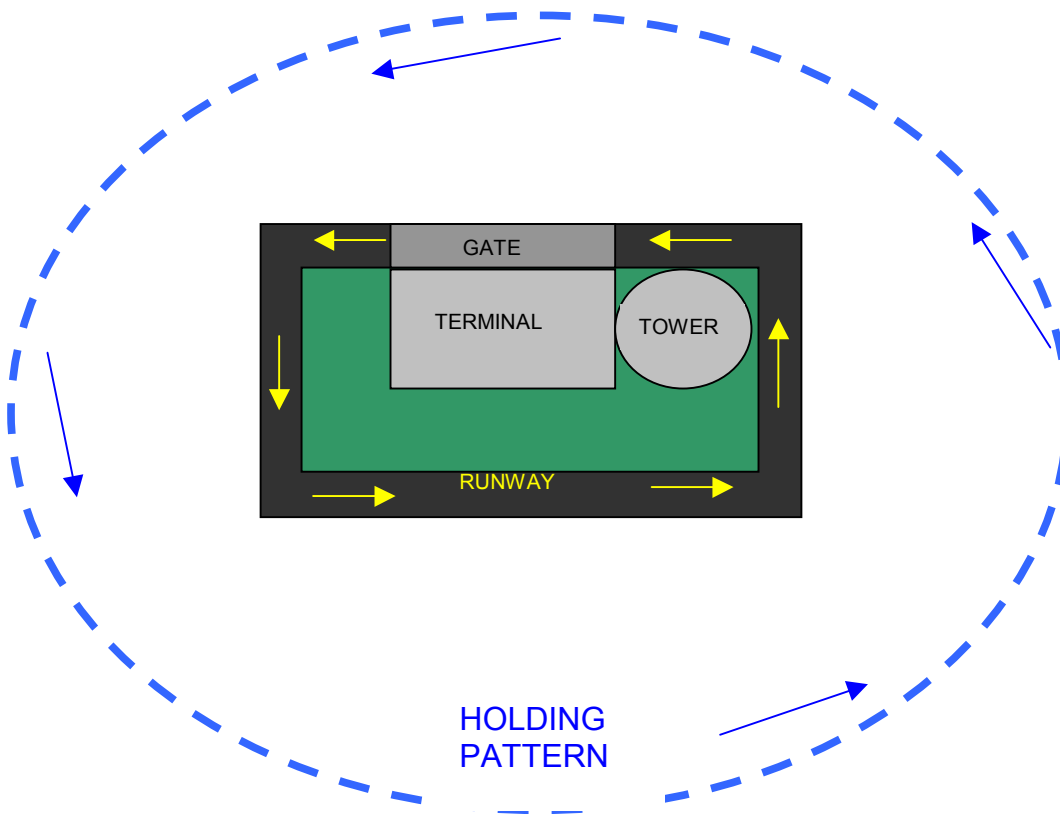


Figure 1: Airport Schematic

Flow of Events

Assume there is an aircraft about to enter the airspace surrounding this airport. The pilot intends to enter the holding pattern, the designated route of flight around an airport that aircraft must occupy while awaiting clearance to land. Upon landing on the runway, the pilot will taxi to the gate to pick up a passenger. Upon passenger boarding, the plane will taxi to the runway and takeoff. The flow of events for this situation is as follows:

1. Plane must request to enter holding pattern (airspace on approach.)
2. The Controller verifies that the requested airspace is clear of other aircraft. If it is not clear, the Controller must issue an order to the aircraft currently in the holding pattern to land. Then the Controller issues an order to the requesting pilot that the aircraft is OK to enter the holding pattern. The plane enters the holding pattern.
3. Plane must request to land on the runway and taxi to the gate.
4. The Controller verifies that the gate is clear of other aircraft. If it is not clear, the Controller must issue an order to the aircraft currently at the gate to taxi to the runway and takeoff. Then the Controller issues an order to the requesting pilot that the aircraft is OK to land and to taxi to the gate. The plane lands on the runway and taxis to the gate.
5. Plane must request to taxi from the gate to the runway and take off.
6. The Controller issues an order to the requesting pilot that the aircraft is OK to taxi to the runway and take off.

Goals, Scenarios, and Use Cases

The **goal** of this Airspace Management Decision Tool is to automate the Air Traffic Controller's decision-making process by modeling the current locations of all aircraft under tower control then outputting the subsequent controller action. The following scenarios define the possible aircraft positions and the corresponding pilot requests that the controller could receive.

- **Scenario 1:** There are no planes under tower control.
 - **Scenario 1.1.** Pilot in the surrounding airspace requests tower approval to enter the holding pattern.
- **Scenario 2:** There is one plane in the holding pattern, no planes at the gate.
 - **Scenario 2.1.** Pilot in the surrounding airspace requests tower approval to enter the holding pattern.
 - **Scenario 2.2.** Pilot in the holding pattern requests tower approval to land and taxi to the gate.
- **Scenario 3:** There are no planes in the holding pattern, one plane at the gate.
 - **Scenario 3.1.** Pilot in the surrounding airspace requests tower approval to enter the holding pattern
 - **Scenario 3.2.** Pilot at the gate requests tower approval to taxi to the runway and take off.
- **Scenario 4:** There is one plane in the holding pattern, one plane at the gate.
 - **Scenario 4.1.** Pilot in the surrounding airspace requests tower approval to enter the holding pattern
 - **Scenario 4.2.** Pilot in the holding pattern requests tower approval to land and taxi to the gate.
 - **Scenario 4.3.** Pilot at the gate requests tower approval to taxi to the runway and take off.

Initial Use Case Diagram

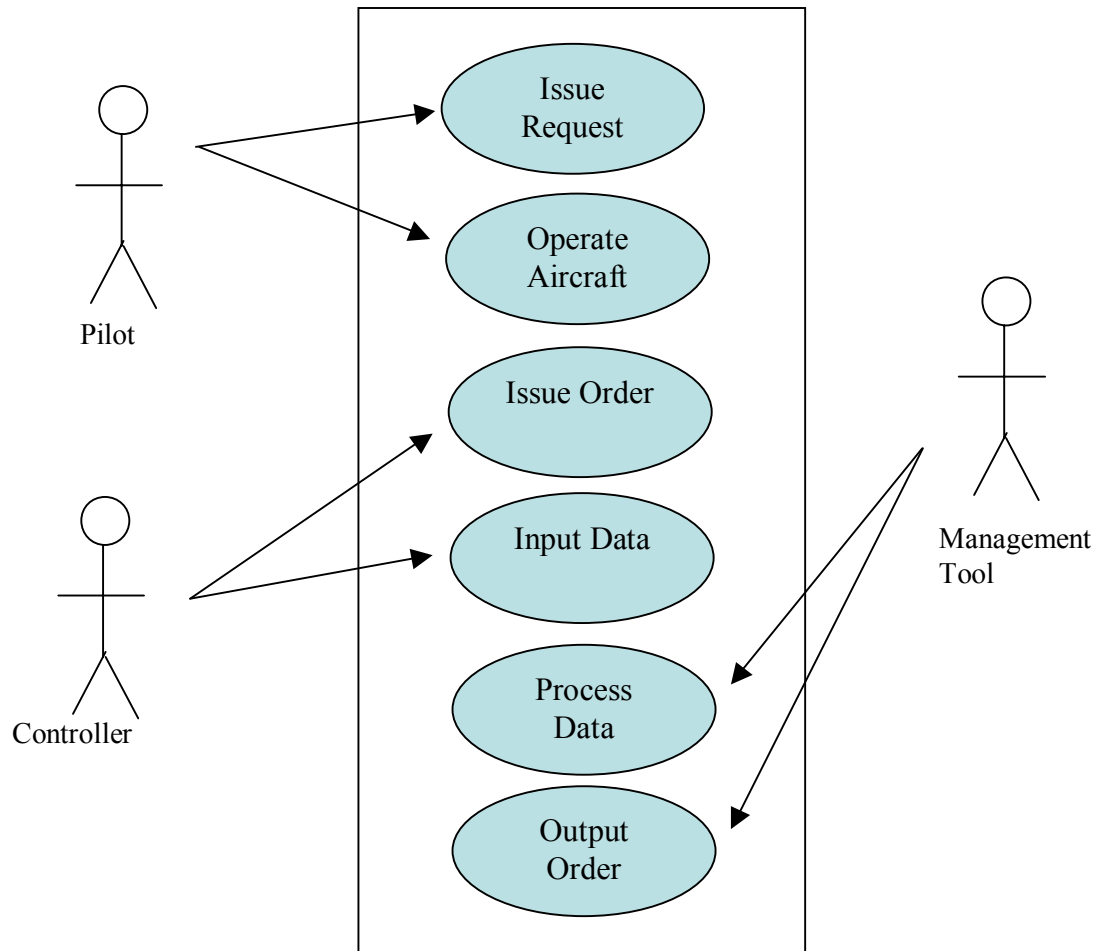


Figure 2: Use Case Diagram

Use Case Text and Activity Diagrams

Use Case 1. Input Data

Primary Actor: Controller

Description: The Controller enters the location of each airport under jurisdiction.

Pre-conditions: The Controller is logged into the program.

Flow of Events:

1. For every aircraft currently within the airport's jurisdiction, the controller makes an entry and specifies location.

Alternative Flow of Events:

2. The controller's shift ends, and another controller assumes control of the situation.

An activity diagram for this use case is given below:

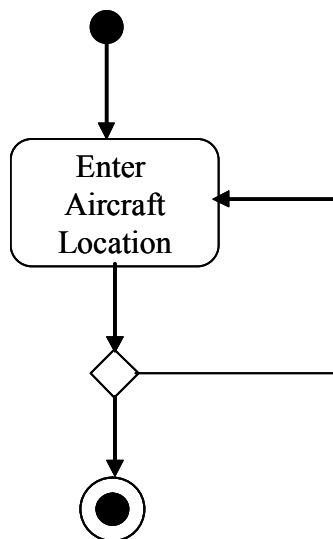


Figure3: Use Case 1: Input Data

Use Case 2. Issue Request

Primary Actor: Pilot

Description: Pilot requests entry into the next airport location.

Pre-conditions: There exists an air-to-ground communications link between the tower and the aircraft that complies with all applicable regulations.

Flow of Events:

1. Pilot activates communications link to tower
2. Pilot achieves contact with controller.
3. Pilot requests approval to enter next phase.

Alternate Flow of Events:

4. Pilot does not achieve contact with the controller because the controller is already in contact with another pilot.

An activity diagram for this use case is given below:

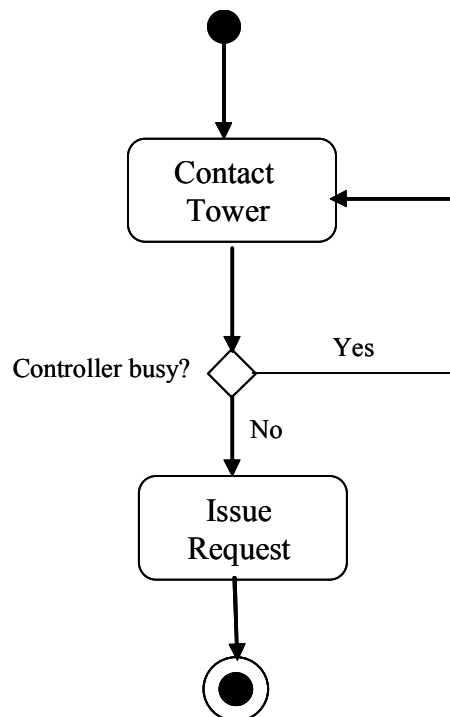


Figure 4: Use Case 2: Issue Request

Use Case 3. Data Processing

Primary Actors: Controller, Decision Management Tool

Description: Decision Management Tool outputs action according to Controller input

Pre-conditions: Controller is logged into the program.

Flow of Events:

1. Controller receives pilot request
2. Controller enters pilot request into system
3. Decision Management Tool processes request
4. Decision Management Tool outputs controller action

An activity diagram for this use case is given below:

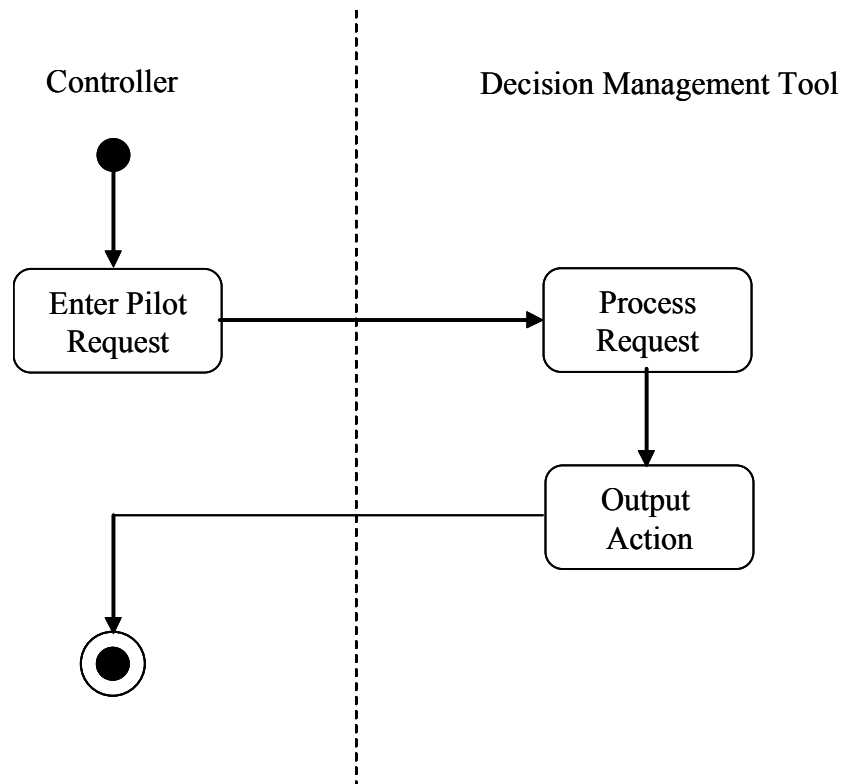


Figure 5: Use Case 3: Data Processing

Use Case 4. Issue Order

Primary Actor: Controller, Pilot

Description: The controller issues an order to the pilot. Pilot maneuvers accordingly.

Pre-conditions: A pilot operating an aircraft under the jurisdiction of the tower has made a request to advance to the controller.

Flow of Events:

1. Controller issues order to the pilot.
2. Pilot maneuvers accordingly.
3. Controller issues next order to the next pilot

An activity diagram for this use case is given below:

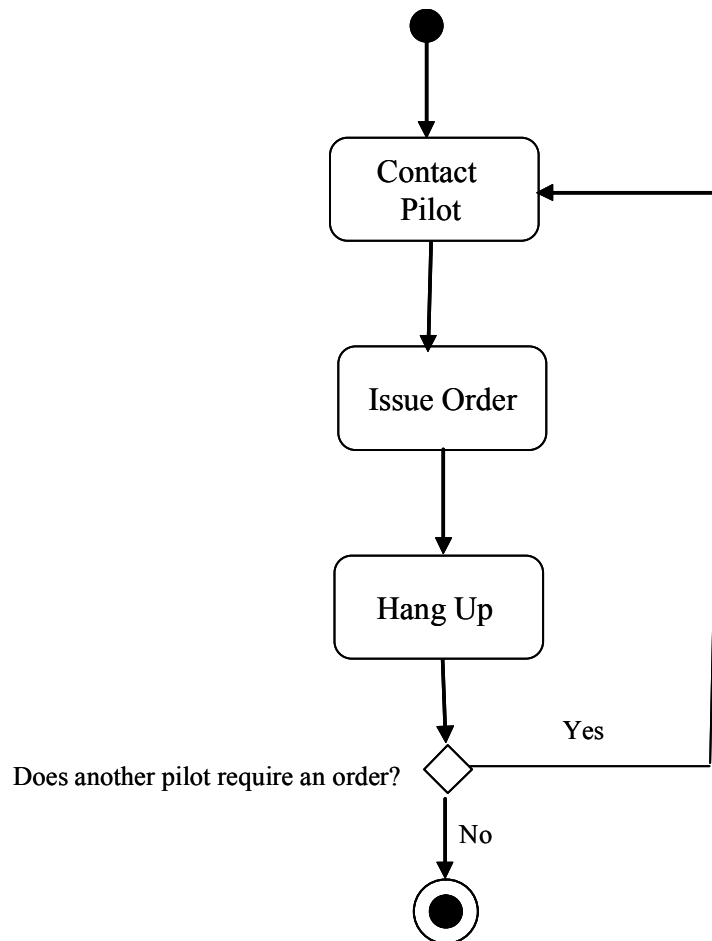


Figure 6: Use Case 4: Issue Order

Generation of Requirements

High Level Requirements

1. Pilot requirements
 - I. The pilot shall be familiar with airport regulations and procedures.
 - II. The pilot shall comply with all orders issued by the controller.
 - III. The pilot shall issue a request to the Controller to indicate desire to maneuver the aircraft to the next location.
2. Controller Requirements
 - I. The controller shall be familiar with airport regulations and procedures.
 - II. The controller shall be certified to operate the Decision Management Tool.
 - III. The controller shall issue orders based on the actions output by the Decision Management Tool to each respective pilot.
 - IV. The controller shall input each pilot request into the Decision Management Tool.
3. Decision Management Tool (Program) Requirements
 - I. The program shall accept input from Controller.
 - II. The program shall issue output.
 - III. The program shall process data in accordance with all airport regulations.
 - IV. All input shall be processed in near real-time.
 - V. All actions shall be output in near real-time.
4. Communications
 - I. A reliable air-to-ground radio communications link shall exist.
 - II. All aircraft within the tower jurisdiction and the immediately surrounding airspace shall have access to the tower frequency.
 - III. The tower may only be in contact with one aircraft at a given time.
 - IV. Incoming calls from pilots shall be received by the controller on a first-come-first-serve basis.

Requirements Traceability

Table 1: Flow down of Requirements from Use Cases

Use Case	Req. #	Description
1 Input Data	2.II	The controller shall be certified to operate the Decision Management Tool.
	2.IV	The controller shall input each pilot request into the Decision Management Tool.
	3.I	The program shall accept input from Controller.
	3.IV	All input shall be processed in near real-time.
2 Issue request	1.I	The pilot shall be familiar with airport regulations and procedures.
	1.III	The pilot shall issue a request to the Controller to indicate desire to maneuver the aircraft to the next location.
	4.I	A reliable air-to-ground radio communications link shall exist.
	4.II	All aircraft within the tower jurisdiction and the immediately surrounding airspace shall have access to the tower frequency.
	4.III	The tower may only be in contact with one aircraft at a given time.
	4.IV	Incoming calls from pilots shall be received by the controller on a first-come-first-serve basis.
3	3.I	The program shall accept input from Controller.

	3.II	The program shall issue output.
	3.III	The program shall process data in accordance with all airport regulations.
	3.IV	All input shall be processed in near real-time.
	3.V	All actions shall be output in near real-time.
4 Issue Order	2.I	The controller shall be familiar with airport regulations and procedures.
	2.II	The controller shall be certified to operate the Decision Management Tool.
	2.III	The controller shall issue orders based on the actions output by the Decision Management Tool to each respective pilot.
	4.I	A reliable air-to-ground radio communications link shall exist.
	4.II	All aircraft within the tower jurisdiction and the immediately surrounding airspace shall have access to the tower frequency.
	4.III	The tower may only be in contact with one aircraft at a given time.
	1.II	The pilot shall comply with all orders issued by the controller.

Table 2: Traceability of Requirements to Use Cases / Scenarios

Req. #	Description	Use Case
1.I	The pilot shall be familiar with airport regulations and procedures.	2
1.II	The pilot shall comply with all orders issued by the controller.	4
1.III	The pilot shall issue a request to the Controller to indicate desire to maneuver the aircraft to the next location.	2
2.I	The controller shall be familiar with airport regulations and procedures.	4
2. II	The controller shall be certified to operate the Decision Management Tool.	1, 4
2.III	The controller shall issue orders based on the actions output by the Decision Management Tool to each respective pilot.	4
2.IV	The controller shall input each pilot request into the Decision Management Tool.	1
3.I	The program shall accept input from Controller.	1, 3
3.II	The program shall issue output.	3
3.III	The program shall process data in accordance with all airport regulations.	3
3.IV	All input shall be processed in near real-time.	1, 3
3.V	All actions shall be output in near real-time.	3

Req. #	Description	Use Case
4.I	A reliable air-to-ground radio communications link shall exist.	2, 4
4.II	All aircraft within the tower jurisdiction and the immediately surrounding airspace shall have access to the tower frequency.	2, 4
4.III	The tower may only be in contact with one aircraft at a given time.	2, 4
4.IV	Incoming calls from pilots shall be received by the controller on a first-come-first-serve basis.	4

System Structure and System Behavior

High Level Structure

The diagram below illustrates the classes and functions within the system. All system functions may be traced to the system behavior. The scope of this project is generating the structure of the software code based on behavioral modeling and analysis.

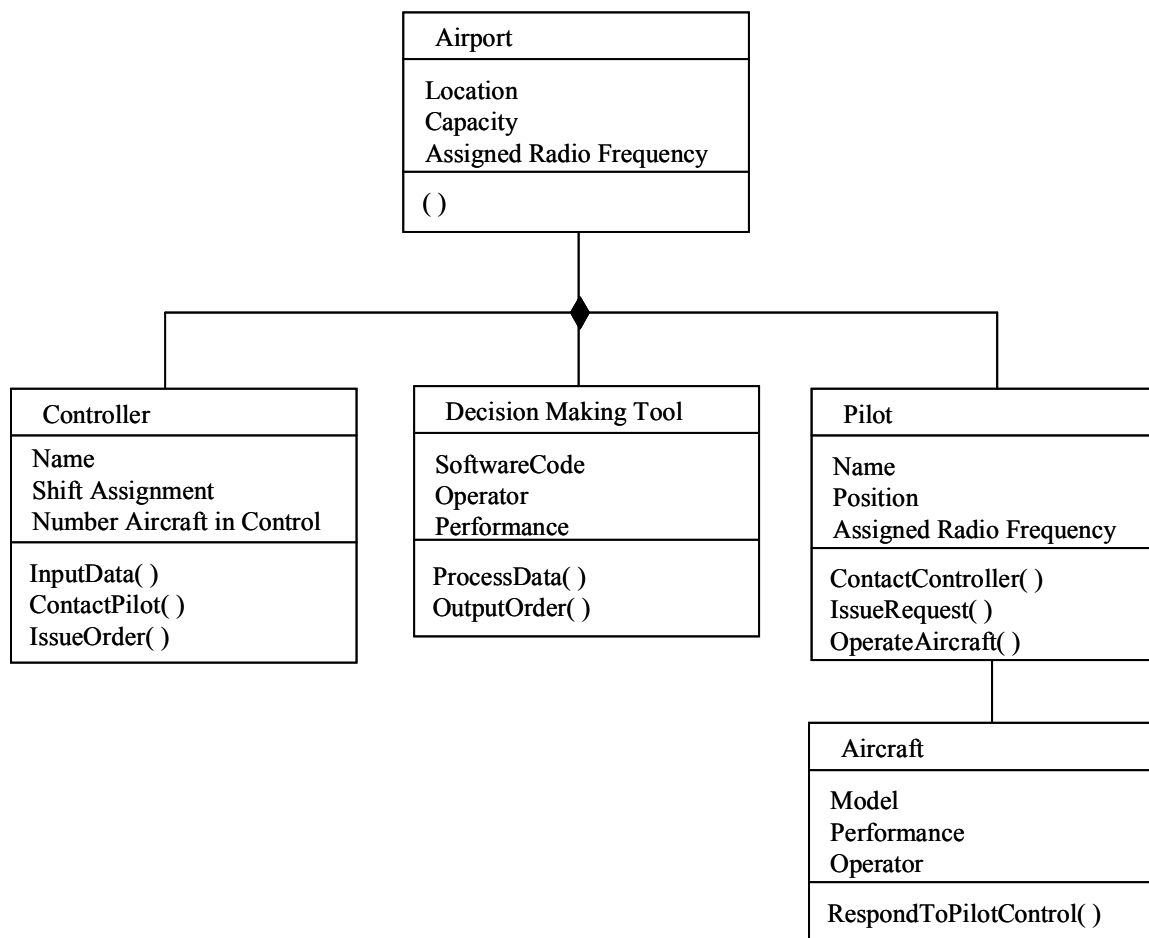


Figure 7: Class Diagram

System Models

This Airspace Management Decision Tool may be modeled as a system with four possible initial states. These states, as described in the scenarios, are:

- No planes under aircraft control
- No planes in the holding pattern; one plane at the gate
- One plane in the holding pattern; no planes at the gate
- One plane in the holding pattern; one plane at the gate

Given any one of these states, the controller may receive any one of three requests, which are entered as input:

- Request to enter holding pattern
- Request to land and taxi to the gate
- Request to taxi to the runway and take off

Once the input is entered into the program, the output is displayed. The output is dependent on the combination of initial state and input. Then, the state is updated to reflect the new current state.

To determine which output will correspond to which initial state and input, the system can be modeled as a finite state machine. This modeling process is made simpler if we can first identify and then eliminate any redundancies in output. The following validation tables display the output according to some given initial state and input, as well as the updated state

States [holding pattern, gate]	
S1	[0,0]
S2	[0,1]
S3	[1,0]
S4	[1,1]

Table 3: System States

Input (Request from Controller)	
X1	Hold
X2	Land
X3	Take Off

Table 4: System Input

Output (Action for Controller)	
Y1	Order Hold
Y2	Order Land; Order Hold
Y3	Invalid Request
Y4	Order Land
Y5	Order Take Off
Y6	Order Take Off
Y7	Order Take Off, Order Land, Order Hold

Table 5: System Output

Output/Controller Action					
Input		Initial State			
		S1	S2	S3	S4
	X1	Y1	Y1	Y2	Y7
	X2	Y3	Y3	Y4	Y6
	X3	Y3	Y5	Y3	Y5

Table 6: Output/Controller Action

New State					
Input		Initial State			
		S1	S2	S3	S4
	X1	S3	S4	S4	S4
	X2	#	#	S2	S2
	X3	#	S1	#	S3

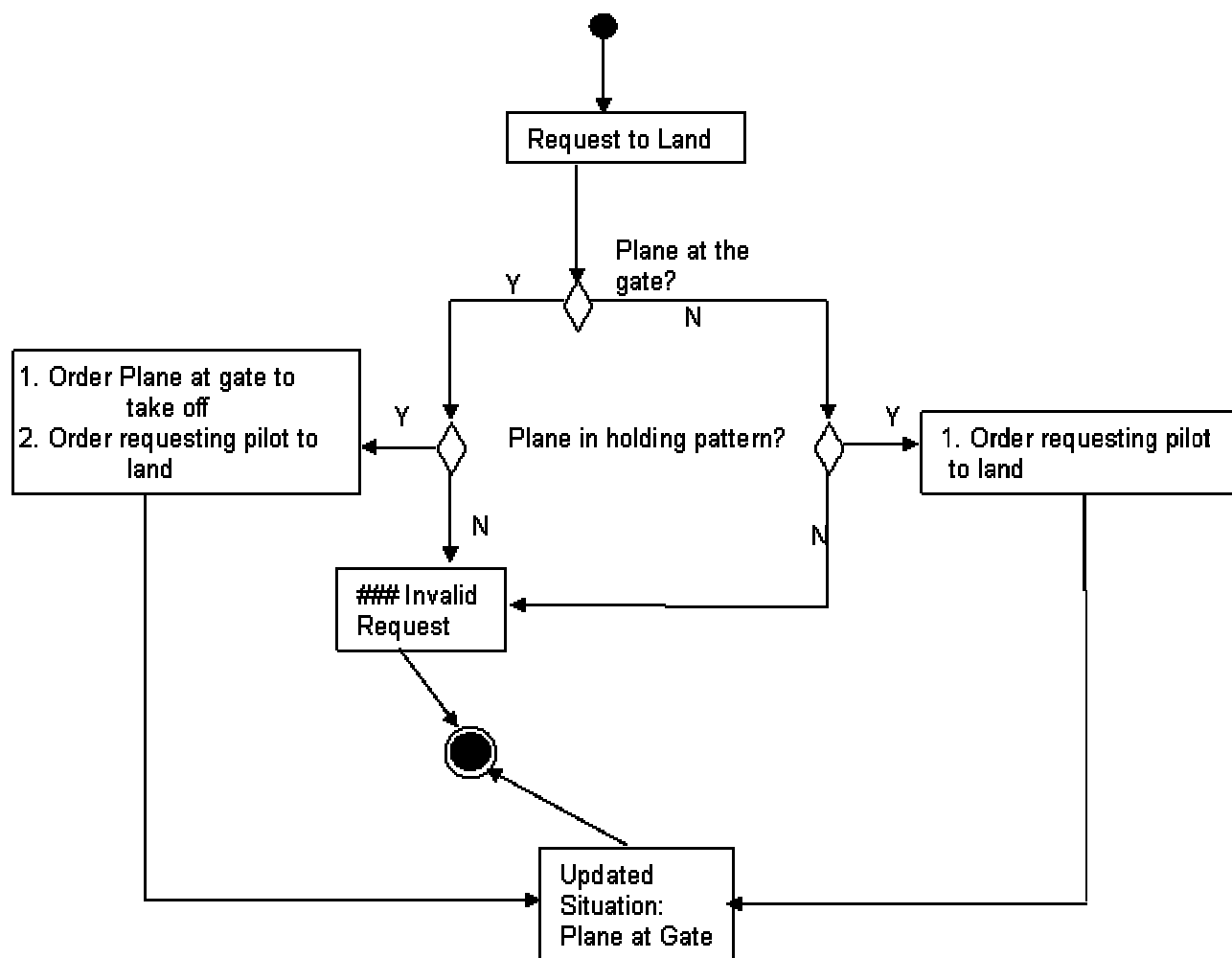
Table 7: New States

According to Table 6, there are 12 possible outputs given the combinations of 4 possible states with 3 possible inputs. However, there are only seven *different* outputs. The table also illustrates which new state is achieved according to the initial state and the input. In the case where a '#' is displayed, the program should recognize the request as 'Invalid.' For example, if there are no aircraft currently under the tower control, the controller will never receive a request to take off. Therefore these validation tables display an initial mapping of system behavior, which will be reflected in the program's software code when it is generated.

These finite state models eliminate redundancies in output and display *how* each output and new state is determined. Each output can be traced back to an initial state via the input. Each updated state can be obtained via some initial state and some input.

With a better understanding of the system behavior based on the finite state machine modeling, state chart diagrams can be made, and will serve as the behavioral-structural foundation of the software code. The state chart diagrams below illustrate the output and updated state based on the input and the initial state.

Figure 8: Request to Land



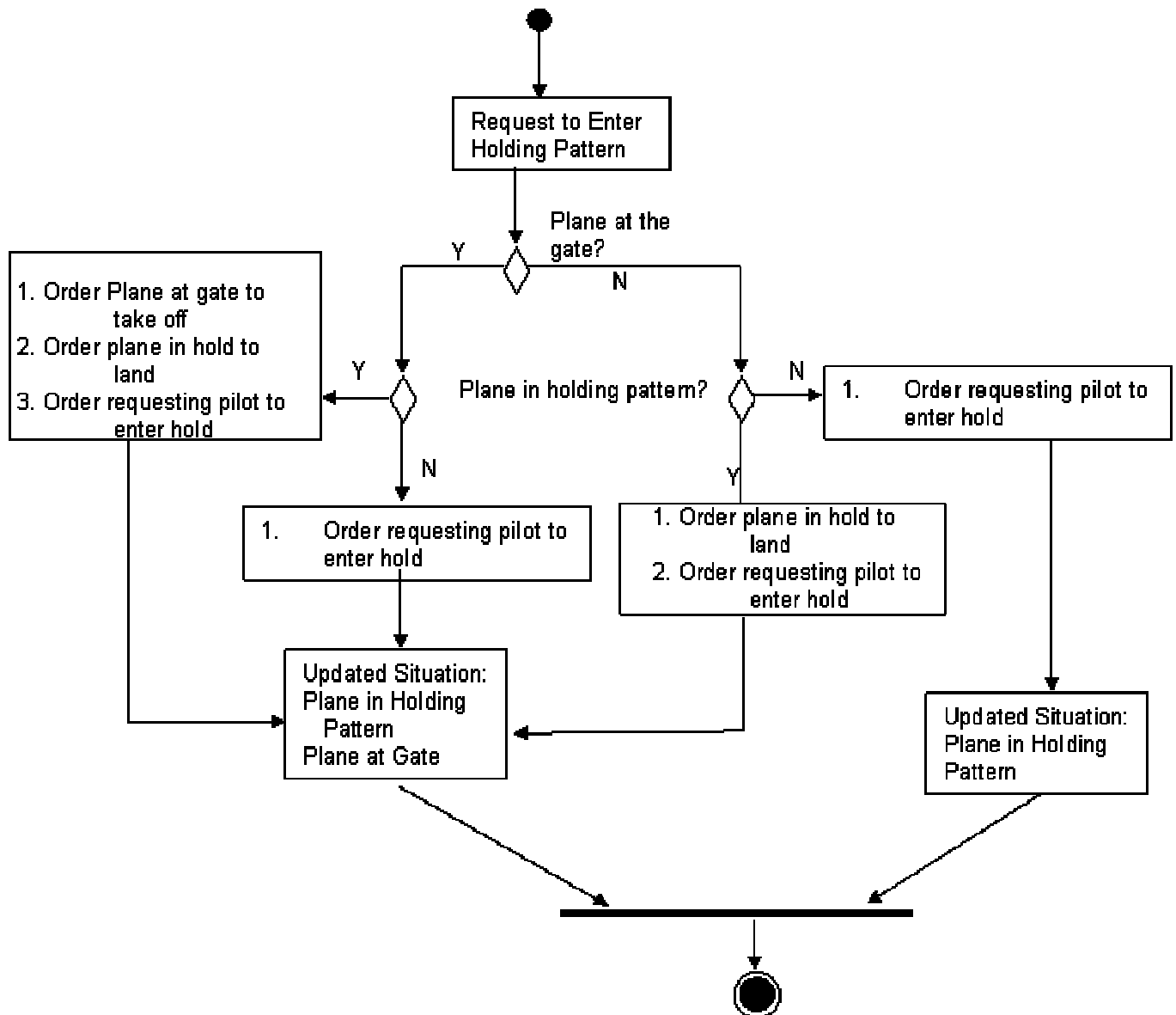
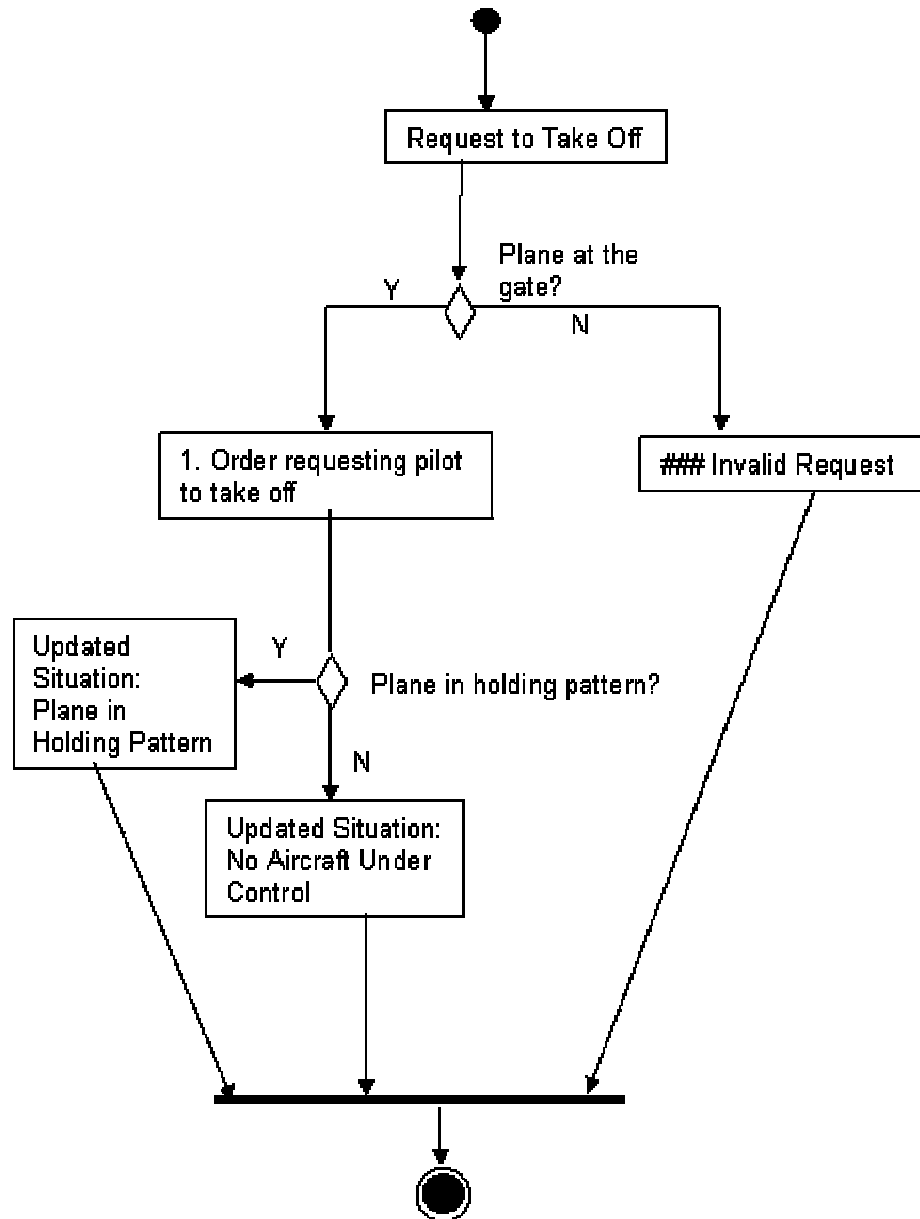


Figure 9: Request to Enter Holding Pattern

Figure 10: Request to Take Off



Structural Design of Program Code

The Airspace Management Decision Tool has been modeled as a finite state machine, and its outputs and revised states have been traced and verified according to a specific input and initial state. These behavioral analyses can now be used to structure the actual software code.

This program is written on MATLAB version 7.0. In order to simulate an air traffic control environment, the program generates initial states randomly. The locations of aircraft are displayed both in the binary format described in the validation tables, and also in words. Then the user enters in some input:

E if the pilot request is to enter the holding pattern

L if the pilot request is to land and taxi to the gate

T if the pilot request is to taxi to the runway and take off.

Then, in accordance with the models above, the program outputs the user action and displays the new state. The user is again prompted to enter input as the request. This process is done for some specified number of iterations. The code is printed in the appendix, followed by a scenario with five iterations.

Once the system software has been developed, it is time to validate and verify, in order to ensure that it performs as required. Moreover,

Figure 11: Basic Procedure for Creating and Evaluating the Physical Design (Austin p. 304)



Following the process indicated in Figure 11, the behavior can be described as the set of input, the changes to the system state, and the resulting output, all of which provide the structure for the Finite State Machines. Finite State Machines in turn serve as the foundation for the structural model for the MATLAB program which runs system scenarios and specifies controller actions as output based on the given input. Now it is time to test and evaluate that these scenarios behave as expected.

Validation is essential, especially in a transportation environment. In the world of air traffic, a controller could potentially have charge over an aircraft with hundreds of people on board. The lives and safety of all of these people depend on the air traffic control system working as expected without compromising safety. This means that even the unexpected must be accounted for when testing and evaluating. The system must demonstrate it behaves as designed, and must verify that it does *not* do what it was *not* designed to do.

Simulation is an effective technique in validation and verification. By running the system through scenarios created to mimic real world situations, the system behavior can be analyzed to verify that it performs as expected in these situations. The system can then be modified as needed and then retested until the system behavior is error-free. However, system behavior can only be verified for the scenarios the system runs through. Unrealized scenarios may potentially uncover system behavior not previously accounted for; therefore the more scenarios the system covers, the broader the range of system behavior that can be validated.

Labeled Transition System Analyzer

The behavior of this Airspace Management Decision Tool will be validated using Labeled Transition State Analyzer (LTSA). LTSA is a tool which models systems as a collection of finite states and generates state diagrams. It counts the states and transitions and can identify any deadlocks or errors within the system. LTSA also comes equipped with an animation feature which displays the current action the system is performing for each process.

Figure 12 illustrates the airport as a finite state process from the perspective of a single aircraft.

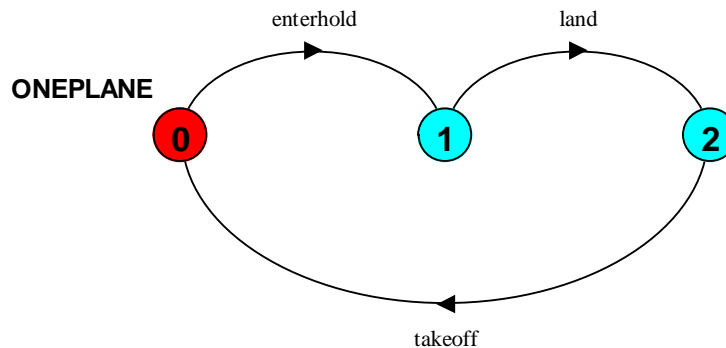


Figure 12: One Plane in the Airport

The nodes correspond to states as follows:

Node 0 = Aircraft in surrounding airspace [0,0]

Node 1 = Aircraft in holding pattern [1,0]

Node 2 = Aircraft at gate [0,1]

Each arrow corresponds to a particular action that allows for the transition from one state to the next. Based on the process illustrated in this figure, an aircraft starts at the system at [0,0] and from there the only possible action is to enter the holding pattern, arriving at [1,0]. From there the aircraft must land and taxi to the gate, bringing the state to [0,1]. Next the aircraft must taxi to the runway and take off, bringing the state back to [0,0].

It is clear from this diagram that an aircraft must progress through the airport in this order: EnterHold \rightarrow Land \rightarrow Takeoff. Any deviation from this order would result in an error, as an aircraft cannot logically put in a request to 'land' while it is already on the ground at the gate, for example. To ensure that an aircraft does not deviate from this order, we can specify a property in LTSA. With this property in place, the diagram illustrating the process of one plane progressing through the phases of the airport can be depicted as in Figure 13. Now we can see that any deviation from the logical process will result in error.

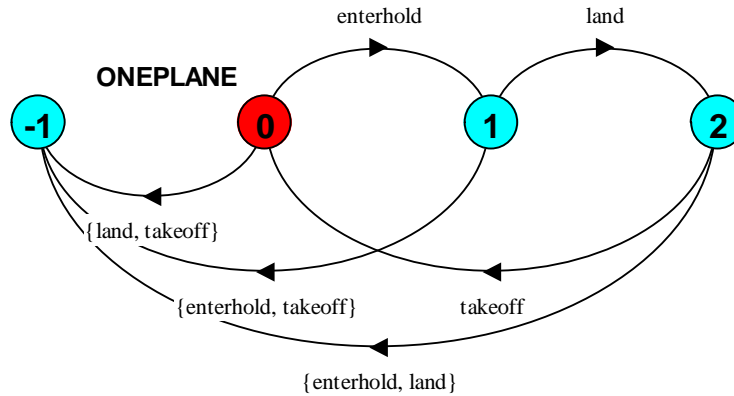


Figure 13: One Plane in the Airport with Progress Property

The nodes correspond to states as follows:

Node 0 = Aircraft in surrounding airspace [0,0]

Node 1 = Aircraft in holding pattern [1,0]

Node 2 = Aircraft at gate [0,1]

This describes the system from the point of view of one aircraft. But what about the point of view from the whole airport, where there is the potential for multiple aircraft

requesting clearances and progressing through the airport phases simultaneously. Figure 14 illustrates this situation.

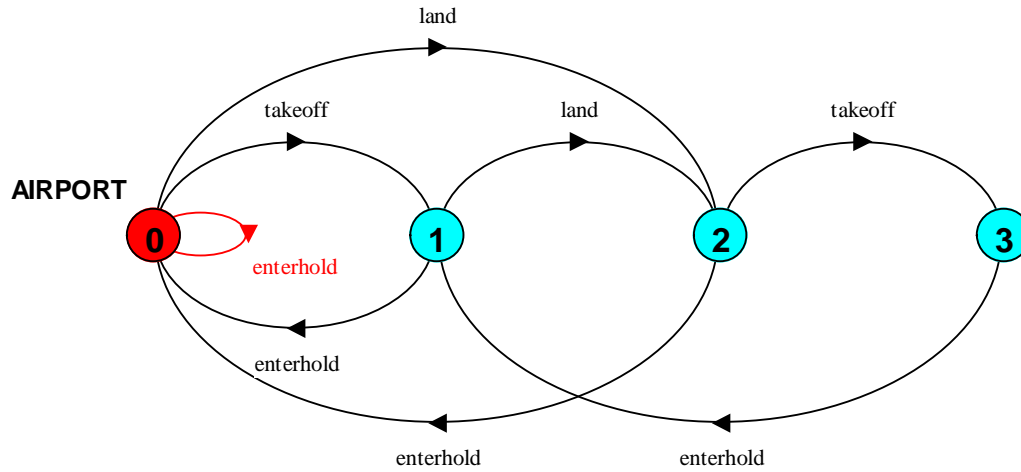


Figure 14: Airport Phases

The nodes correspond to states as follows:

- Node 0 = Aircraft in holding pattern, aircraft at gate [1,1]
- Node 1 = Aircraft in holding pattern [1,0]
- Node 2 = Aircraft at gate [0,1]
- Node 3 = No aircraft in airport phases [0,0]

Each arrow is labeled with its corresponding action. This model can be used in verifying the scenarios generated by the MATLAB program. This is evidenced by the table below. The scenario begins with a state randomly generated by MATLAB. In the real world, this could be the equivalent to some state entered by an outgoing controller, before an incoming controller takes over.

The columns under the MATLAB headings are the states we expect, the states that will be updated as the controller performs the corresponding actions. The columns under the LTSA headings are the actual states we get by using the LTSA-generated model. In order to run this simulation, the LTSA animation feature will be used to track the updates in states based on the selected actions. Note that depending on the node, only certain actions are available for selection, as listed in the 'Possible Input' column.

Iteration	MATLAB - Expected			LTSA - Actual			
	Current State	Input	New State	Current State	Possible Input	Selected Input	New State
1	[1,1]	EnterHold	[1,1]	[1,1]	EnterHold Land Takeoff	EnterHold	[1,1]
2	[1,1]	TakeOff	[1,0]	[1,1]	EnterHold Land Takeoff	TakeOff	[1,0]
3	[1,0]	EnterHold	[1,1]	[1,0]	EnterHold Land	EnterHold	[1,1]
4	[1,1]	Land	[0,1]	[1,1]	EnterHold Land Takeoff	Land	[0,1]
5	[0,1]	TakeOff	[0,0]	[0,1]	EnterHold TakeOff	TakeOff	[0,0]

Table 8: Expected and Actual Output

We can see that the results produced by running the scenarios through LTSA match the results expected in MATLAB. Therefore, the system checks out for *this* scenario. This table is expanded in the appendix: a scenario was generated with 50 iterations to test not only the accuracy and correctness, but also the performance and the stability of the program as it runs over an extended range of iterations.

Once again, we can introduce properties to ensure an aircraft does not deviate from the EnterHold → Land → TakeOff order of airport progression.

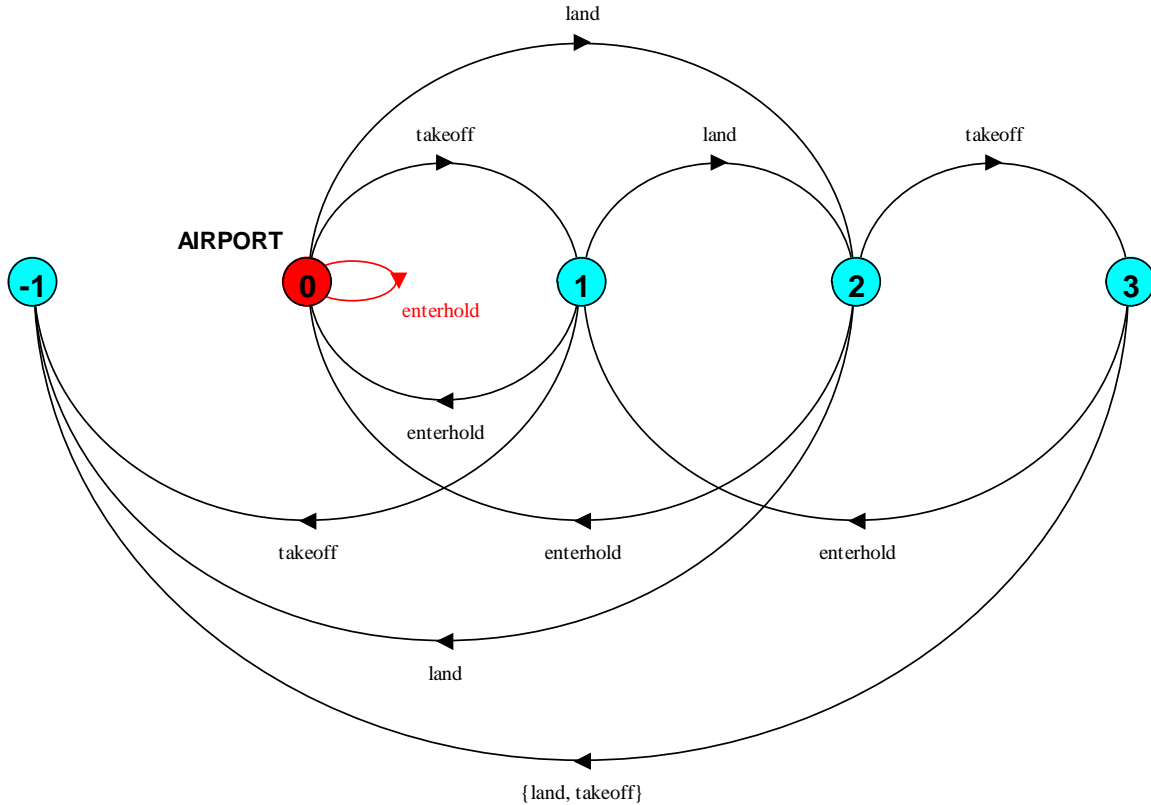


Figure15: Airport Phases with Error

The nodes correspond to states as follows:

- Node -1 = Error
- Node 0 = Aircraft in holding pattern, aircraft at gate [1,1]
- Node 1= Aircraft in holding pattern [1,0]
- Node 2 = Aircraft at gate [0,1]
- Node 3 = No aircraft in airport phases [0,0]

Figure 15 illustrates an error situation should an aircraft deviate from this order, that is, request an action that is not logical given its current state. It is important to state that in the real world, this error would be attributed to human error. It is the pilot who puts in the request to the controller; it is the controller who enters the pilot's request into the program. It is inconceivable that a pilot would physically make a request that deviates from the logical order, for example a pilot would not request to enter hold and then to takeoff. On the other hand, it seems more possible for a controller to mistakenly enter an illogical input by mistyping. This error is accounted for in MATLAB by returning an 'invalid request' warning, corresponding to the error node in LTSA.

Other properties can be set in LTSA to ensure safety. Consider the situation of deadlock. Is the system designed in such a way so that any aircraft entering the airport can make it out of the airport? Furthermore, given the potential for a constant influx of air

traffic at the airport, aircraft constantly entering and exiting, can the airport ever achieve the $[0,0]$ state?

System safety can be checked within LTSA to ensure that the system cannot become deadlocked. In the situation that models one aircraft through the airport's EnterHold→Land→TakeOff phases, we can check for progress violations with respect to the property imposed that an aircraft must progress in this order. A progress check reveals that there is no deadlock.

For the situation that models the airport states, we want to check the safety through a progress violation. In this case, a progress property is not declared so that a default check is performed. In LTSA, the default checks *all* actions for progress violations. A progress check reveals that there is no deadlock, thus system safety is ensured. All states are reachable, all actions are eventually executed, and all states have at least one outgoing transition. Therefore an aircraft that enters the airport will make it out, and the system enables the airport to achieve its original, empty state.

Conclusion

This project shows how behavioral modeling can be transformed into a functional software design and validated. The various behavioral and structural models (i.e. scenarios, UML diagrams, and validation tables) can be used to validate one another and can be used to structure the software code. The MATLAB code itself is also a model of the system, which is evident in its “IF” statements. It generates systems scenarios by updating system states based on a user input, outputting the required action, and updating the system state. Before this code can be applied in any situation, it must be validated to ensure that the system achieves its expected behavior. Additionally, validation identifies any system errors or violations of safety or progress based on defined system behavior and defined system properties. One form of such violations is deadlock, the case where a state is reachable, but the system cannot advance from a particular state.

Applying deadlock to an air traffic system could potentially propagate into air traffic problems on a global scale. If an air traffic system is deadlocked, it will inevitably cause delays at the airport. Therefore all incoming traffic will be delayed, and will congest the surrounding airspace, much like roads experience back-up at the scene of a car accident or road closure. To relieve the strain on the airspace capacity, other airports would initiate ground delay programs by delaying outbound flights not only to the affected airport, but also any flights which utilize the congested airspace. Delays are costly to the airline industry, so it is clear that any type of deadlock could have adverse effects on the *entire* air traffic system. More importantly the safety and well-being of the passengers are at stake. If an air traffic system behaves in an unpredicted manner, for example, the controller is given the wrong action to take, the lives of the passengers are compromised. In the simple example presented in this project, any wrong action could cause gate delays, runway incursions, and the worst case scenario—a midair collision.

LTSA is the validation tool used to verify that the behavior expected is actually achieved. By running the MATLAB-generated scenarios through LTSA’s animation, we validate the system behavior. By imposing properties and running checks on safety and progress violations, it can be seen that the system never reaches deadlock.

Appendix

MATLAB Code

%Program Code: Airspace Management Decision Tool

```
n=input ('Enter number of iterations');
k=1;

%Display current state
hold=randint;
gate=randint;
current_state=[hold, gate]

while k<=n

    if hold==1
        disp('Aircraft in Holding Position')
    end

    if gate==1
        disp('Aircraft at Gate')
    end

    if current_state==[0,0]
        disp('No Aircraft Under Tower Control')
    end

    %Enter Next Request: E for enter holding pattern, L for land and
    taxi to
    %gate, T for taxi to runway and take off

    Request=input('Enter Pilot Request','s')

    if Request==( 'E' )
        if hold==1
            if gate==1
                disp ('Order aircraft at gate to taxi to runway and
takeoff')
                disp ('Order aircraft in hold to land and taxi to
gate')
                disp('Approve request to enter holding pattern')
                hold=1;
                gate=1;
                current_state=[hold, gate]
            else disp ('Order aircraft in hold to land and taxi
to gate')
                disp('Approve request to enter holding pattern')
                hold=1;
                gate=1;
                current_state=[hold, gate]
            end
        end
    end
end
```



```

        else disp('Approve request to enter holding pattern')
        hold=1;
        current_state=[hold, gate]
        end
    k=k+1;
end

if Request==( 'L' )
    if hold==0
        disp('Invalid Request')
    else
        if gate==1
            disp('Order aircraft at gate to taxi and takeoff')
            disp('Approve request to land')
            hold=0;
            gate=1;
            current_state=[hold, gate]
        end
        if gate==0
            disp('Approve request to land')
            hold=0;
            gate=1;
            current_state=[hold, gate]
        end
    end
    k=k+1;
end

if Request==( 'T' )
    if gate==0
        disp('Invalid Request')

    else
        if hold==0
            disp('Approve Request to takeoff')
            hold=0;
            gate=0;
            current_state=[hold, gate]
        end
        if hold==1
            disp('Approve Request to takeoff')
            hold=1;
            gate=0;
            current_state=[hold, gate]
        end
    end
    k=k+1;
end
end
end

```

MATLAB Output

Sample output from MATLAB code with five iterations:

```
Enter number of iterations5

current_state =

     1     0

Aircraft in Holding Position
Enter Pilot RequestT

Request =

T

Invalid Request
Aircraft in Holding Position
Enter Pilot RequestE

Request =

E

Order aircraft in hold to land and taxi to gate
Approve request to enter holding pattern

current_state =

     1     1

Aircraft in Holding Position
Aircraft at Gate
Enter Pilot RequestL

Request =

L

Order aircraft at gate to taxi and takeoff
Approve request to land

current_state =

     0     1

Aircraft at Gate
Enter Pilot RequestT

Request =

T
```

Approve Request to takeoff

current_state =

0 0

No Aircraft Under Tower Control

Enter Pilot RequestE

Request =

E

Approve request to enter holding pattern

current_state =

1 0

LTSA Code

LTSA codes:

- One plane in the airport, with progress property:

ONEPLANE=(enterhold->land->takeoff->ONEPLANE).
property ONEPLANE=(enterhold->land->takeoff->ONEPLANE).

- Entire Airport:

AIRPORT(N=3)=PHASES[N],
PHASES[i:0..N]=
 if(i==0)then (enterhold->PHASES[i+1])
 else if (i==1)then (land->PHASES[i+1]|enterhold->PHASES[i+2])
 else if (i==2)then (takeoff->PHASES[i-2]|enterhold->PHASES[i+1])
 else if (i==3) then (enterhold->PHASES[i+0]|land->PHASES[i-1]|takeoff->PHASES[i-2]).

- Entire Airport with error:

AIRPORT(N=3)=PHASES[N],
PHASES[i:0..N]=
 if(i==0)then (enterhold->PHASES[i+1]|land->ERROR|takeoff->ERROR)
 else if (i==1)then (land->PHASES[i+1]|enterhold->PHASES[i+2]|takeoff->ERROR)
 else if (i==2)then (takeoff->PHASES[i-2]|enterhold->PHASES[i+1]|land->ERROR)
 else if (i==3) then (enterhold->PHASES[i+0]|land->PHASES[i-1]|takeoff->PHASES[i-2]).

MATLAB Validation with LTSA—50 iterations

This table shows the comparison of a MATLAB-generated scenario with 50 iterations with the state diagrams generated by the validation tool LTSA. The scenario begins with a state randomly generated by MATLAB. In the real world, this could be the equivalent to some state entered by an outgoing controller, before an incoming controller takes over.

The columns under the MATLAB headings are the states we expect, the states that will be updated as the controller performs the corresponding actions. The columns under the LTSA headings are the actual states we get by using the LTSA-generated model. In order to run this simulation, the LTSA animation feature will be used to track the updates in states based on the selected actions. Note that depending on the node, only certain actions are available for selection, as listed in the ‘Possible Input’ column.

In some cases, the MATLAB produces an ‘Invalid Request’ output. This would correspond to some situation in which a pilot mistakenly makes an illogical request, or to a situation where a controller mistakenly mistypes the request. In MATLAB, the state reverts to the last updated state. In LTSA, this would correspond to the error node. In the LTSA columns in the table, an error is reached anytime the selected input is not one of the possible input listed. This would transition to the error node. In this table, the new state reverts to the last current state.

Iteration	MATLAB - Expected			LTSA - Actual			
	Current State	Input	New State	Current State	Possible Input	Selected Input	New State
1	[1,0]	Land	[0,1]	[1,0]	Enter Land	Land	[0,1]
2	[0,1]	Enter	[1,1]	[0,1]	Enter Takeoff	Enter	[1,1]
3	[1,1]	Takeoff	[1,0]	[1,1]	Enter Land Takeoff	Takeoff	[1,0]
4	[1,0]	Enter	[1,1]	[1,0]	Enter Land	Enter	[1,1]
5	[1,1]	Land	[0,1]	[1,1]	Enter Land Takeoff	Land	[0,1]
6	[0,1]	Takeoff	[0,0]	[0,1]	Enter Takeoff	Takeoff	[0,0]
7	[0,0]	Enter	[1,0]	[0,0]	Enter	Enter	[1,0]
8	[1,0]	Enter	[1,1]	[1,0]	Enter Land	Enter	[1,1]
9	[1,1]	Enter	[1,1]	[1,1]	Enter Land Takeoff	Enter	[1,1]

10	[1,1]	Land	[0,1]	[1,1]	Enter Land Takeoff	Land	[0,1]
11	[0,1]	Takeoff	[0,0]	[0,1]	Enter Takeoff	Takeoff	[0,0]
12	[0,0]	Enter	[1,0]	[0,0]	Enter	Enter	[1,0]
13	[1,0]	Takeoff	INV	[1,0]	Enter Land	Takeoff	ERROR
14	[1,0]	Land	[0,1]	[1,0]	Enter Land	Land	[0,1]
15	[0,1]	Takeoff	[0,0]	[0,1]	Enter Takeoff	Takeoff	[0,0]
16	[0,0]	Enter	[1,0]	[0,0]	Enter	Enter	[1,0]
17	[1,0]	Enter	[1,1]	[1,0]	Enter Land	Enter	[1,1]
18	[1,1]	Enter	[1,1]	[1,1]	Enter Land Takeoff	Enter	[1,1]
19	[1,1]	Takeoff	[1,0]	[1,1]	Enter Land Takeoff	Takeoff	[1,0]
20	[1,0]	Land	[0,1]	[1,0]	Enter Land	Land	[0,1]
21	[0,1]	Enter	[1,1]	[0,1]	Enter Takeoff	Enter	[1,1]
22	[1,1]	Enter	[1,1]	[1,1]	Enter Land Takeoff	Enter	[1,1]
23	[1,1]	Takeoff	[1,0]	[1,1]	Enter Land Takeoff	Takeoff	[1,0]
24	[1,0]	Land	[0,1]	[1,0]	Enter Land	Land	[0,1]
25	[0,1]	Enter	[1,1]	[0,1]	Enter Takeoff	Enter	[1,1]
26	[1,1]	Takeoff	[1,0]	[1,1]	Enter Land Takeoff	Takeoff	[1,0]
27	[1,0]	Land	[0,1]	[1,0]	Enter Land	Land	[0,1]
28	[0,1]	Enter	[1,1]	[0,1]	Enter Takeoff	Enter	[1,1]
29	[1,1]	Takeoff	[1,0]	[1,1]	Enter Land Takeoff	Takeoff	[1,0]

30	[1,0]	Takeoff	INV	[1,0]	Enter Land	Takeoff	ERROR
31	[1,0]	Land	[0,1]	[1,0]	Enter Land	Land	[0,1]
32	[0,1]	Land	INV	[0,1]	Enter Takeoff	Land	ERROR
33	[0,1]	Takeoff	[0,0]	[0,1]	Enter Takeoff	Takeoff	[0,0]
34	[0,0]	Enter	[1,0]	[0,0]	Enter	Enter	[1,0]
35	[1,0]	Enter	[1,1]	[1,0]	Enter Land	Enter	[1,1]
36	[1,1]	Land	[0,1]	[1,1]	Enter Land Takeoff	Land	[0,1]
37	[0,1]	Takeoff	[0,0]	[0,1]	Enter Takeoff	Takeoff	[0,0]
38	[0,0]	Enter	[1,0]	[0,0]	Enter	Enter	[1,0]
39	[1,0]	Enter	[1,1]	[1,0]	Enter Land	Enter	[1,1]
40	[1,1]	Enter	[1,1]	[1,1]	Enter Land Takeoff	Enter	[1,1]
41	[1,1]	Takeoff	[1,0]	[1,1]	Enter Land Takeoff	Takeoff	[1,0]
42	[1,0]	Land	[0,1]	[1,0]	Enter Land	Land	[0,1]
43	[0,1]	Enter	[1,1]	[0,1]	Enter Takeoff	Enter	[1,1]
44	[1,1]	Enter	[1,1]	[1,1]	Enter Land Takeoff	Enter	[1,1]
45	[1,1]	Takeoff	[1,0]	[1,1]	Enter Land Takeoff	Takeoff	[1,0]
46	[1,0]	Land	[0,1]	[1,0]	Enter Land	Land	[0,1]
47	[0,1]	Enter	[1,1]	[0,1]	Enter Takeoff	Enter	[1,1]
48	[1,1]	Takeoff	[1,0]	[1,1]	Enter Land Takeoff	Takeoff	[1,0]
49	[1,0]	Enter	[1,1]	[1,0]	Enter Land	Enter	[1,1]

50	[1,1]	Land	[0,1]	[1,1]	Enter Land Takeoff	Land	[0,1]
----	-------	------	-------	-------	--------------------------	------	-------

References

Austin, Mark. *Information-Centric Systems Engineering. Lecture Notes for ENSE 621-622-623*. Fall Semester, 2005. University of Maryland, College Park, MD.

Magee, Jeff. *LTSA: Labeled Transition System Analyser*. 1999.
<http://www.doc.ic.ac.uk/~jnm/book/ltsa/LTSA.html>

Magee, Jeff and Kramer, Jeff. *Concurrency: State Models & Java Programs*. (Slides). 1999. <http://www.doc.ic.ac.uk/~jnm/book/slides.html>