Airspace Management Decision Tool for Use at a Small Airport

Kerin Thornton
ENPM642
Spring 2005

Table of Contents

Introduction:

Problem Statement

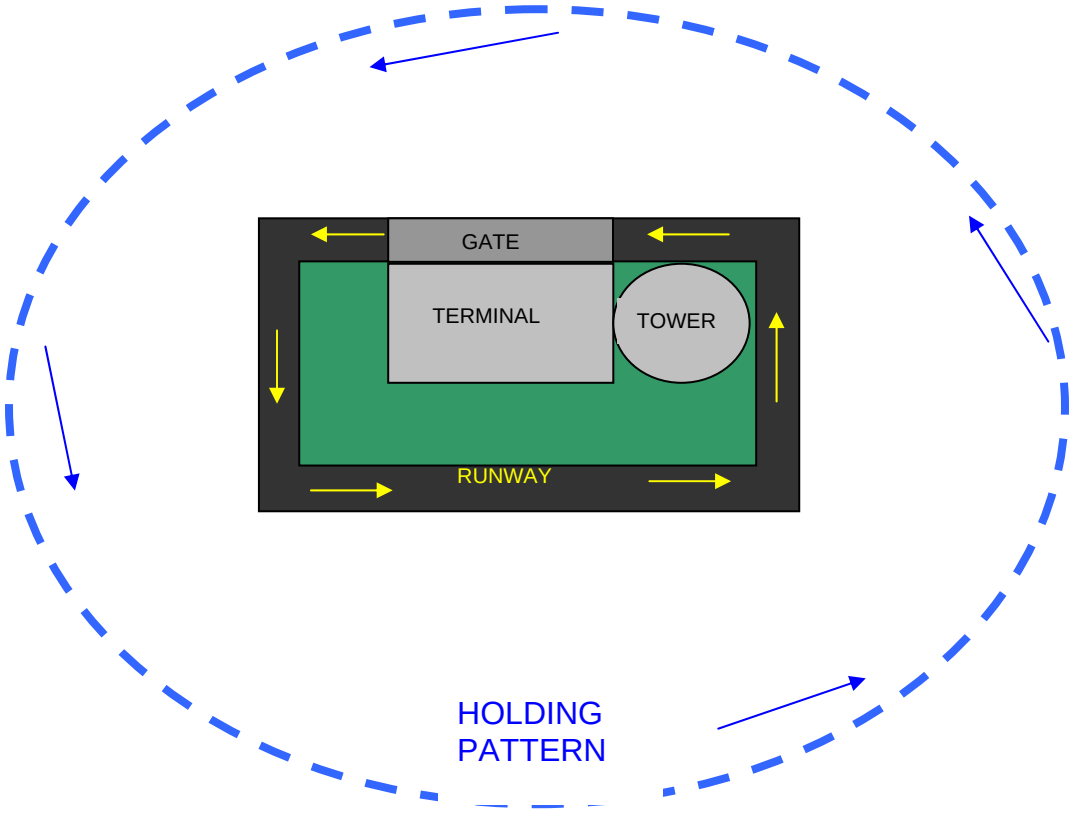
Even at small airports, air traffic controllers must maintain a constant awareness of a dynamic situation, which can be enough of a workload to affect decision-making. Any misinterpretation of a given situation may lead to a wrong decision, which has the potential for disaster. The scope of this project is to create a model-based program which air traffic controllers at small airports may use to track aircraft positions, and issue orders accordingly.

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.

Assumptions

The scope of this project will assume the following:

- The airspace immediately surrounding the airport's holding pattern is of an unlimited capacity. This is where planes make the request to enter the holding pattern upon approach.
- Once a plane has taken off, it has cleared the airspace and is no longer under the tower control.
- Only one plane may be in the holding pattern at a given time.
- Only one plane may be on the runway at a given time (to take off, or land).
- The airport owns two taxiways. Planes may taxi to the runway and taxi to the gate simultaneously.
- Only one plane may be at the gate at a given time.
- ATC may only receive one request at a time.
- The controller may issue an order to a pilot who has not made a request
- Once an issue has been ordered to a pilot, the action is carried out instantaneously.



Flow of Events:
Plane must request to enter holding pattern (airspace on approach.)
Upon permission, plane enters holding pattern.
Plane must request to land.
Upon permission, plane lands and taxis to the gate.
Plane must request to leave gate.
Upon permission, plane taxis and takes off.

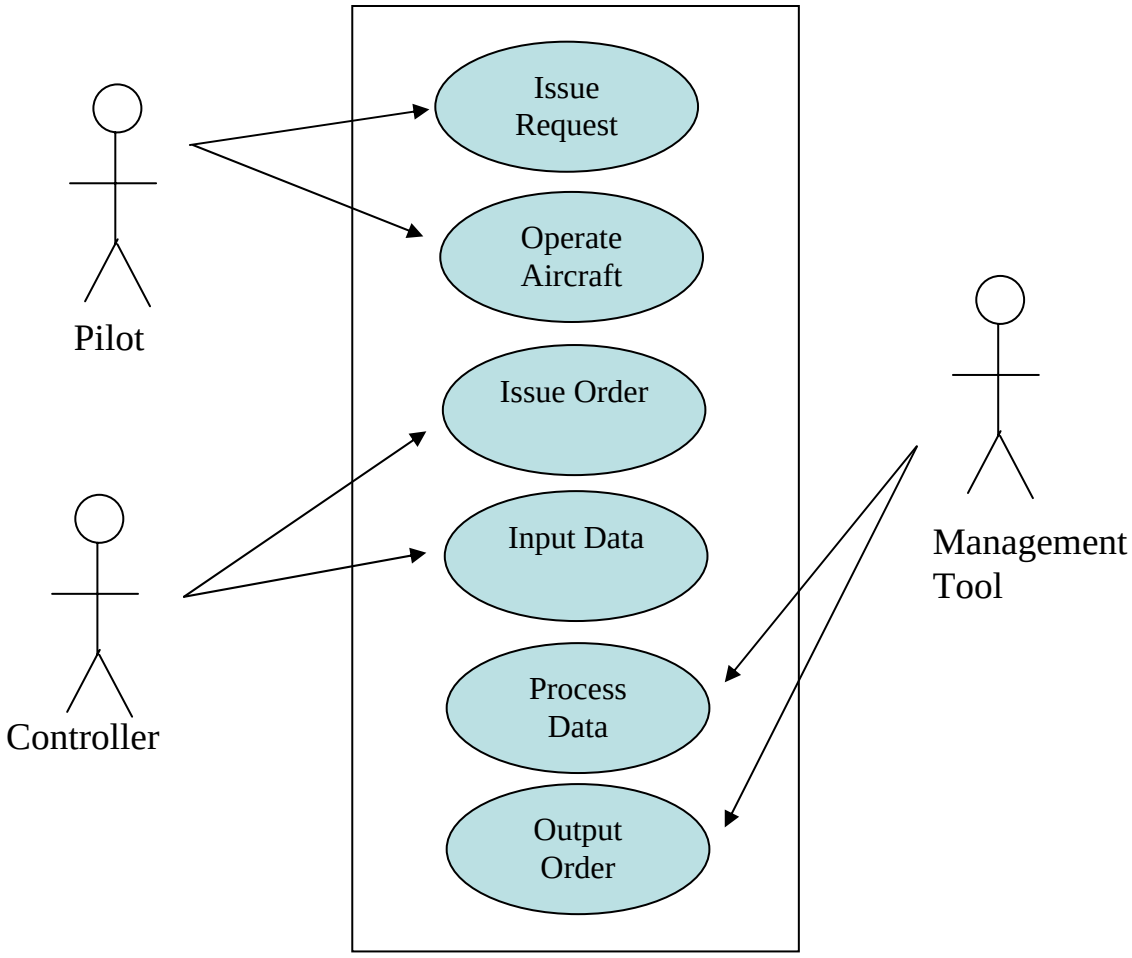
Goals, Scenarios, and Use Cases:

Goals & Scenarios

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. 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



Use Case Text and Activity Diagrams

Use Case 1. Input Data

Primary Actor: Controller

Flow of Events:

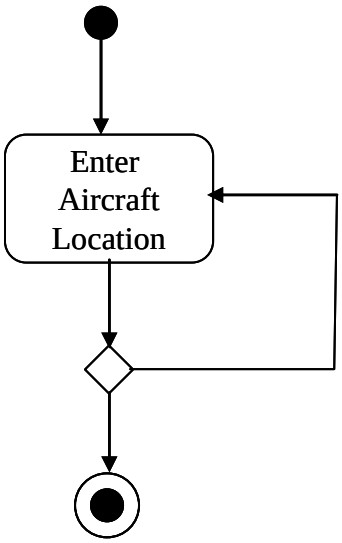
1. For every aircraft currently within the airport's jurisdiction, the controller makes an entry and specifies location.

Alternative Flow of Events:

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

Post-condition:
Assumption:.

An activity diagram for this use case is given below:



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 **c**ommunications link between the tower and the aircraft that complies with all applicable regulations.
Flow of Events:

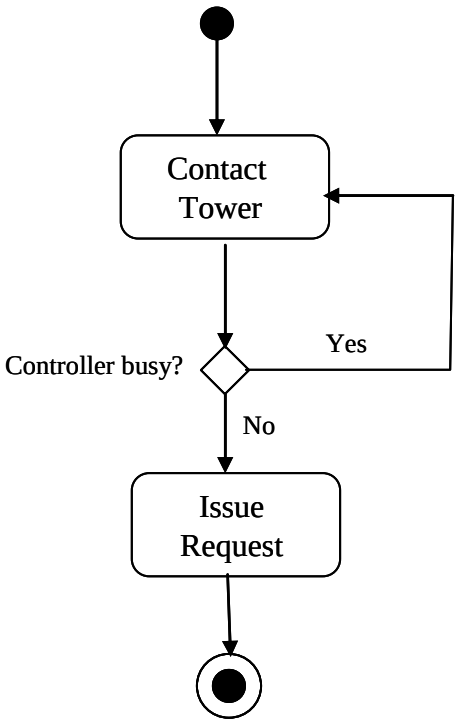
1. Pilot activates communications link to tower
2. Pilot achieves contact with controller.
3. Pilot requests approval to enter next phase.

Alternate Flow of Events:

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

Post-condition:
Assumption:

Activity diagram for this use case is given as below:



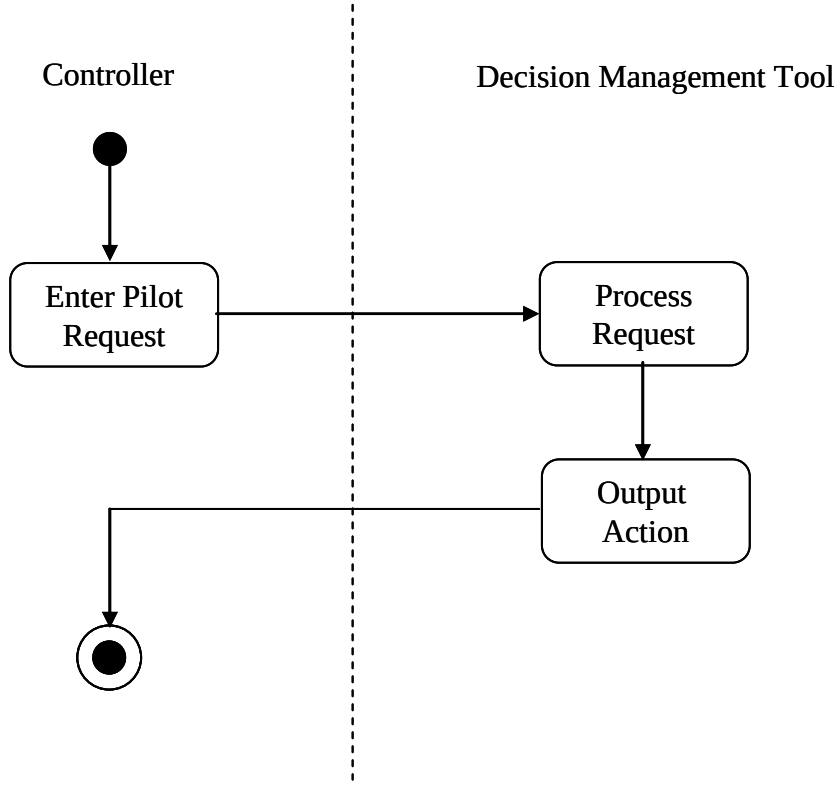
Use Case 3. Data Processing

Primary Actors: Controller, Decision Management Tool
Description: Decision Management Tool outputs action according to Controller input
Pre-conditions:.
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

Post-condition:
Assumption:

Activity diagram for this use case is given as below:



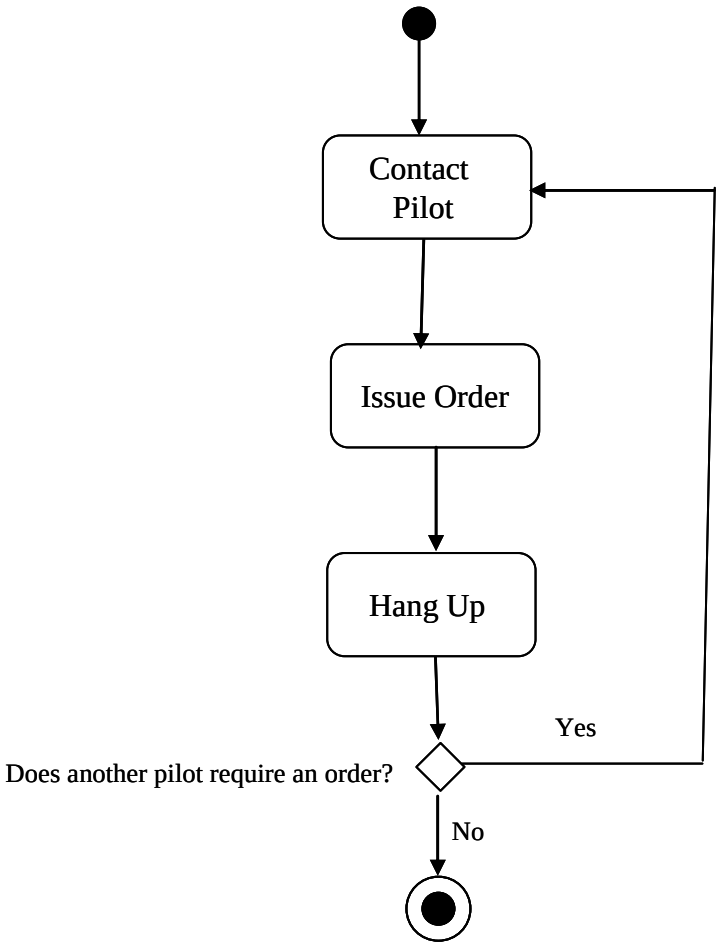
Use Case 4. Issue Order

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

Post-condition:
Assumption:

Activity diagram for this use case is given as below:



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

Flow down of Requirements from Use Cases

SOURCE	DESTINATION	
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.

Traceability of Requirements to Use Cases / Scenarios

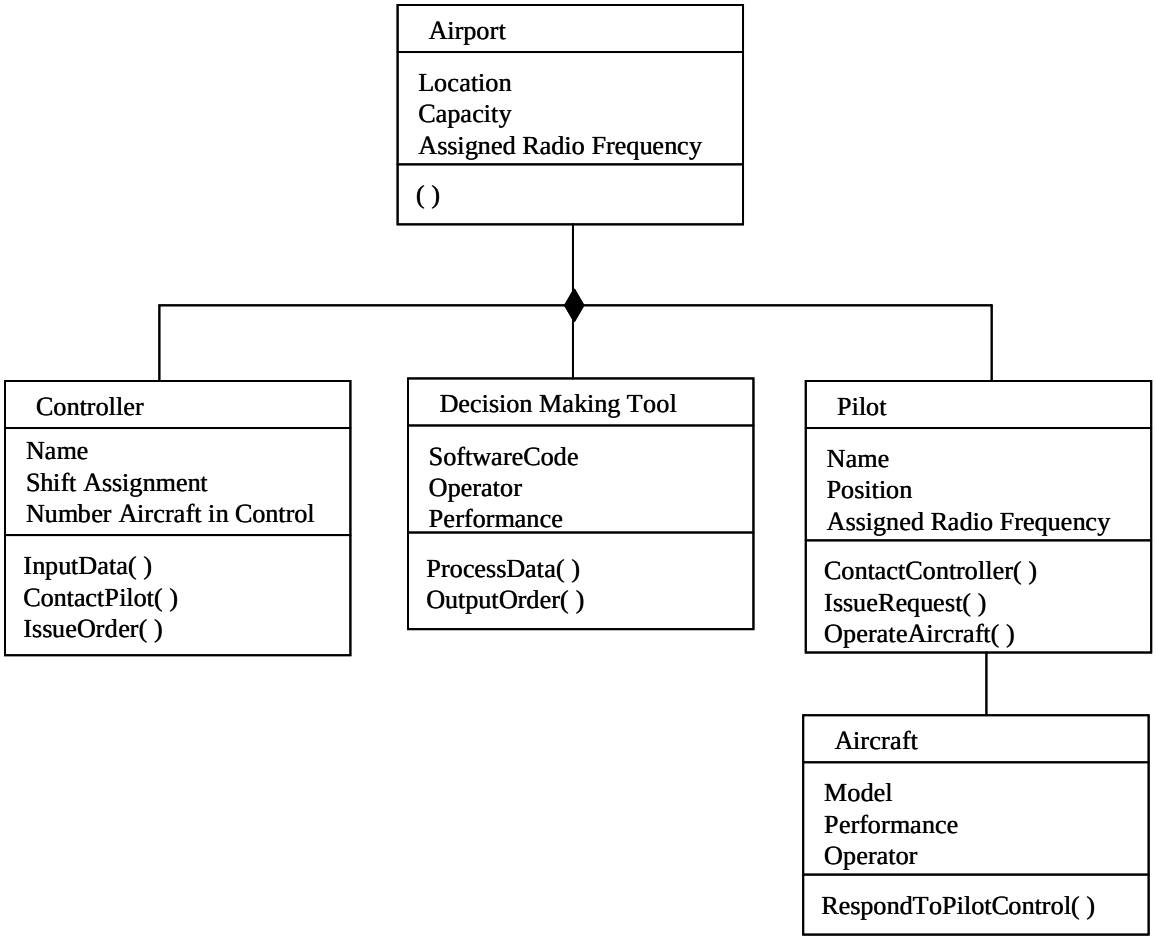
SOURCE		DESTINATION
Req. #	**Description**	**Use Case**
1.I	The pilot shall be familiar with airport regulations and procedures.	2
1.II	The pilot shall comply with all orders issued by the controller.	4
1.III	The pilot shall issue a request to the Controller to indicate desire to maneuver the aircraft 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
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.



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 (location of aircraft)		Input (Request from pilot)	
S1	[0,0]	X1	Hold
S2	[0,1]	X2	Land
S3	[1,0]	X3	Take Off
S4	[1,1]		

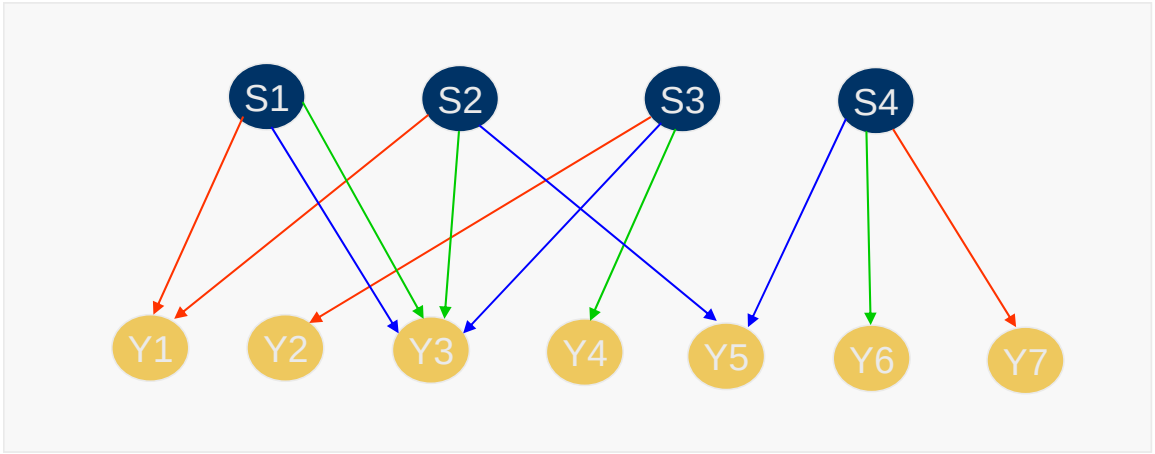
Output (Action for Controller)	
Y1	Order Hold
Y2	Order Land; Order Hold
Y3	Invalid Request
Y4	Order Land
Y5	Order Take Off
Y6	Order Take Off; Order Land
Y7	Order Take Off; Order Land; Order Hold

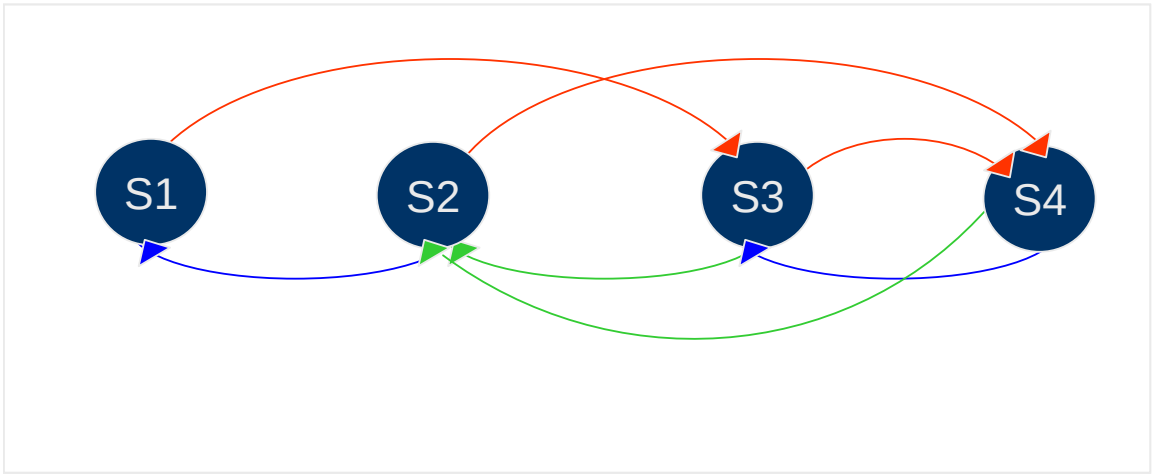
Output/Controller Action					
		Initial State			
		S1	S2	S3	S4
Input	X1	Y1	Y1	Y2	Y7
	X2	Y3	Y3	Y4	Y6
	X3	Y3	Y5	Y3	Y5

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

According to the table, there are 12 possible outputs given the combinations of 4 possible states with 3 possible input. 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.

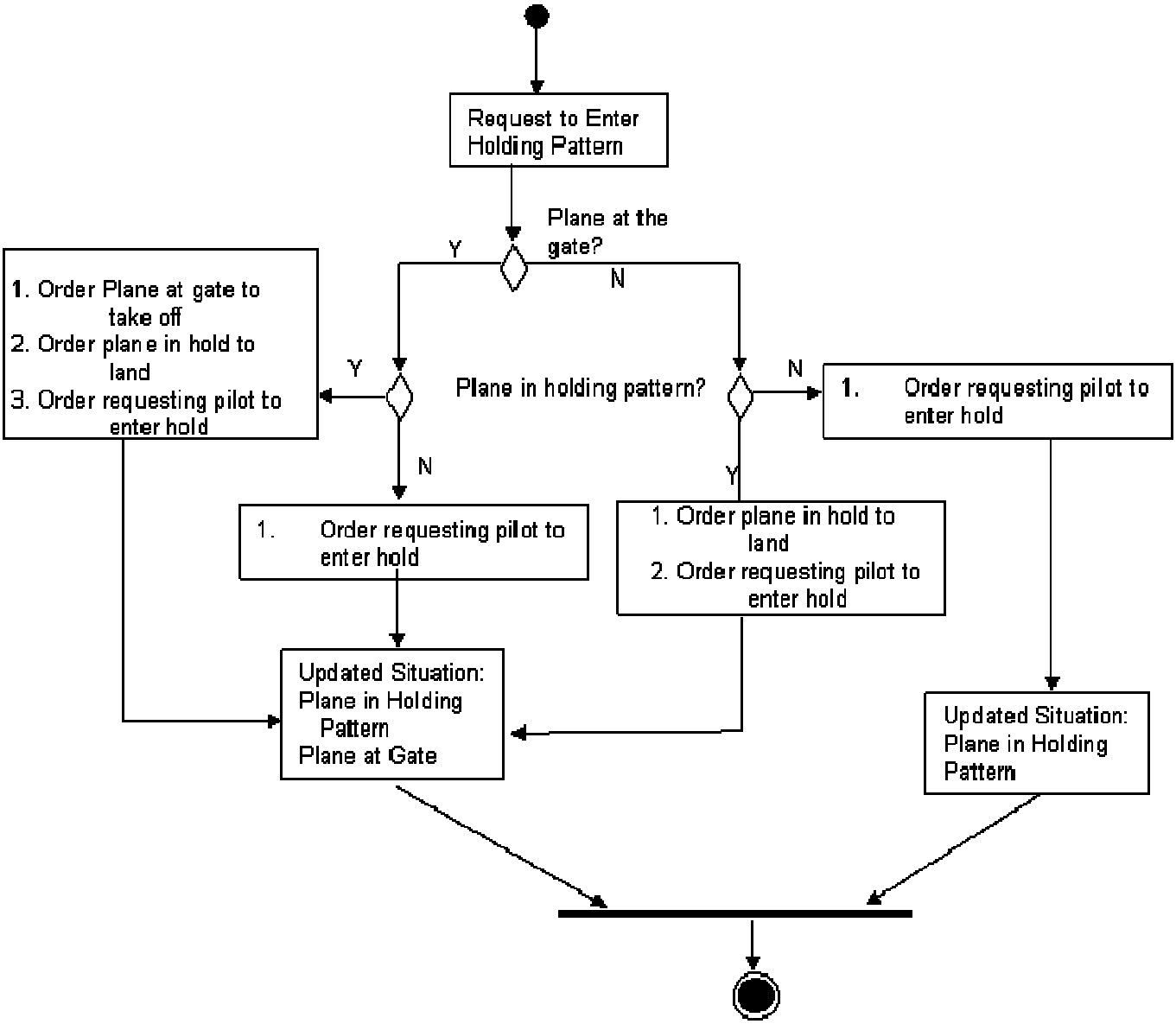
The validation tables may be modeled as finite state machines, as seen below.

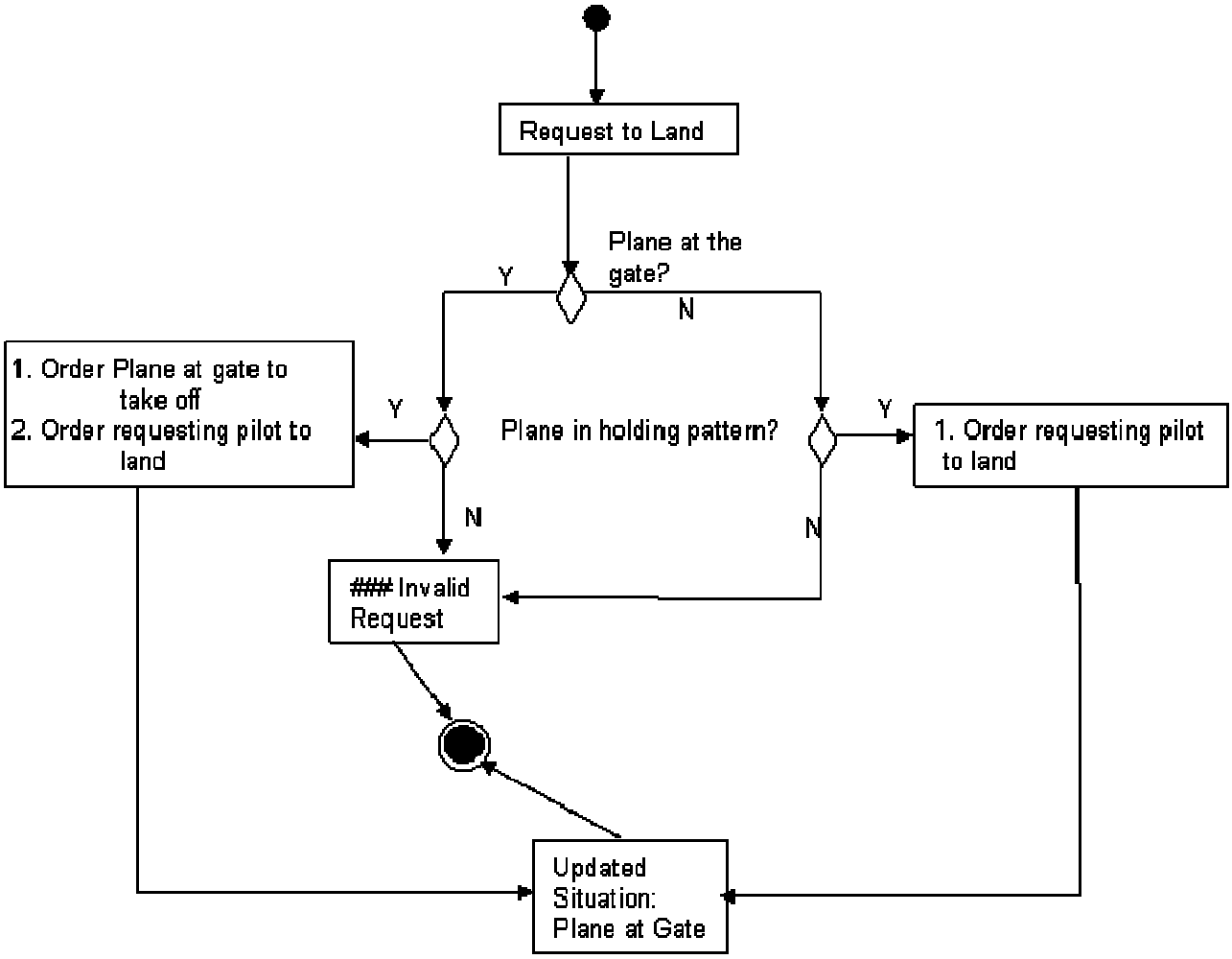


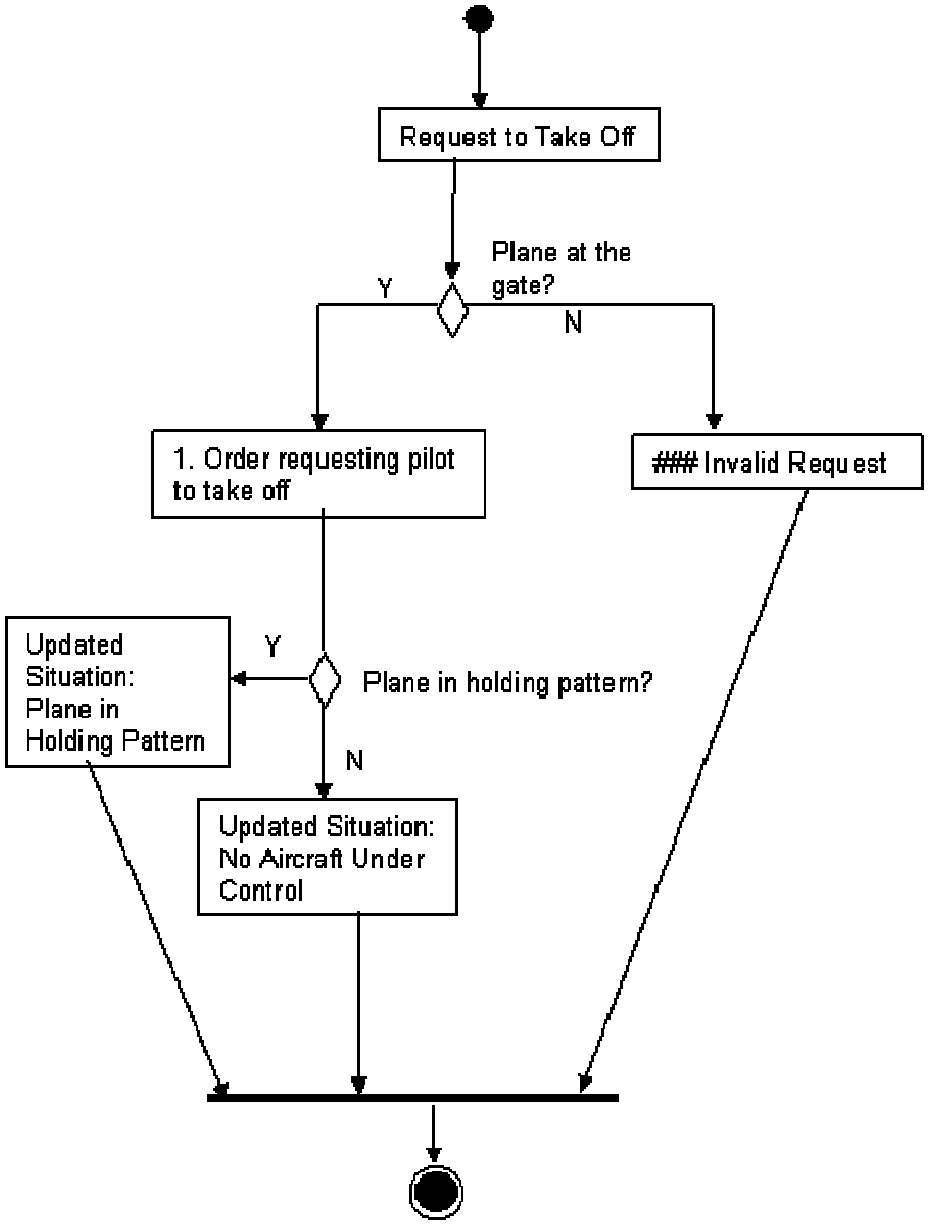


These finite state models eliminate redundancies in output and display *how* each output and new state is determined. Each output can be traced back to an initial state via the input. Each updated state can be obtained via some initial state and some input.

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.


Request to Enter Holding Pattern
Plane at the gate?
Y
N
1. Order Plane at gate to take off
2. Order plane in hold to land
3. Order requesting pilot to enter hold
Y
Plane in holding pattern?
N
1. Order requesting pilot to enter hold
N
Y
1. Order requesting pilot to enter hold
1. Order plane in hold to land
2. Order requesting pilot to enter hold
Updated Situation: Plane in Holding Pattern Plane at Gate
Updated Situation: Plane in Holding Pattern


Request to Land
Plane at the gate?
Y
N
1. Order Plane at gate to take off
2. Order requesting pilot to land
Y
Plane in holding pattern?
Y
1. Order requesting pilot to land
N
N
Invalid Request
Updated Situation: Plane at Gate


Request to Take Off
Plane at the gate?
Y
N
1. Order requesting pilot to take off
Invalid Request
Updated Situation: Plane in Holding Pattern
Y
Plane in holding pattern?
N
Updated Situation: No Aircraft Under Control

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 below, followed by actual output on a situation with five iterations.

%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
        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
```

MATLAB Output

```
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
```

Conclusions and Future Work

This project shows the how behavioral modeling can be transformed into a functional software design. The various models (validation tables, finite state machines, and state chart diagrams) can be used to validate one another and can be used to structure the code. The code itself is also a model of the system, which is evident in its “IF” statements.

Further analysis of this system will continue to focus on its behavior at a more detailed level. Animation and more precise validation methods will be used. Additionally, some of the assumptions initially stated in this project will change; for example, the program may be modified to factor in the element of time, so that the controller and pilot actions are no longer assumed to be instantaneous. This will require the use of an extended set of software validation and design tools, including LTSA and UPPAAL.