Object Oriented Modeling and Simulation of Airport Surface Traffic Control (ASTC) System

1. Introduction

While there has been a lot of progress in computer science, the adoption of these concepts in engineering has been a little slow. Today, few engineers use C++, Smalltalk, Java or other OOP languages to develop in-house applications or computational routines. Several authors point out that applications using at least some of the concepts of OOP techniques could save development time and effort by reducing coding times and making the programs more efficient to run. Perhaps the best selling point in the use of OOP software development framework will be a substantial reduction in the maintenance cost of a real model. It is estimated that between 45% and 65% of the cost to develop software is needed to maintain a software model based in standard procedural programming techniques. The use of OOP promises a reduction to 25-40% instead.

In the specific area of computer simulation, several OOP simulation languages have been developed over the last fifteen years including Modsim III, Simula 67, ROSS, Sim++, SIMEX 3.0 and QNAP2 among others. While some of these languages have gained widespread use in manufacturing and facilities planning sectors, very few applications of these exist in the area of air traffic control and airport simulation. Today the mainstream airport and airspace simulation models use procedural programming paradigms making models difficult to enhance, debug and maintain.

The scope of this project is the analysis of airport surface traffic control (ASTC) system with objects such as aircraft, runway, control tower, and taxiway etc. to help in analysis and management of the system.

2. Problem Description

We view the ASTC as a task driven system. In a task driven system, various tasks are realized by interaction of participating entities through message exchange. Various types of tasks are performed in ASTC system. For example, during a landing task an airplane asks an air traffic controller for permission to land on an assigned runway. Upon receiving the request, the air traffic controller object will query current position information of all the airplanes that could possibly be on the path of the incumbent aircraft and check the availability of the runway. If the runway is available, then a landing permission message is sent to the target airplane and the landing process starts.

The creation and execution of tasks follows a well-defined sequence in synchronization with a timing mechanism. Each entity involved in task performance implements its own local procedures for completing its share of the task.

Tasks for airfield operations vary in complexity depending on the purpose of the actual airfield traffic model. Nonetheless, a task management object or task engine is necessary for all the simulated situations.

3. Solution Design

3.1 Task Driven Simulation Engine

Tasks are organized sequentially within a task queue. The task engine dispatches tasks from the queue one at a time until the simulation clock stops. The task engine has the following main functions:

Collecting tasks: Whenever a task is created during the simulation, the task engine will add the task into the task queue.

Dispatching tasks: The task engine will dispatch the task into the task process channel whenever the global time matches the time mark of a task in the task queue.

Branching tasks: The task process channel will branch the task to the correspondent sub-channel.

Pending tasks: If the task cannot be executed at the moment and it is expected to be executed as soon as possible, the task can be pushed back into the task queue and its time mark set the same as that of the next available task.

Deleting a task: If a task is finished or needs to be deleted, the task engine will remove and destroy the task.

3.2 Queuing with Priority

Some tasks could be created early and executed late, such as scheduling a flight. On the other hand, some tasks could be created late but required to be executed immediately after their creation, such as holding an airplane in a departure queue. The priority queue object in this project is a classical first in-first out queue. Before adding a task to the task queue, the task engine will first compare the priorities of all of the tasks in the queue with the priority of the newly created task, and then add the task into the queue at the right position according to the outcome of the priority comparison. Priority queues are also modeled in runway operations because landing aircrafts have priority over departures.

3.3 Task Description and Representation

Tasks are represented by a task class. A task includes two pieces of information: an identifier and supplemental data. Task Identifier data is a set of data that specifies which object is going to receive a task. Supplemental data is a set of attributes that tell the targeted object what is happening and usually what needs to be done.

A task identifier should have the following elements:

a. **The primary task label**: Whenever a task is dispatched, the first task label which tells what task it is (e.g. a scheduling task, an airplane related task, or a shortest path searching task). According to the primary task identifier, the task dispatcher will send the task into corresponding task processing channel. The primary label is used for primary task branching.

b. **The secondary task label**: Whenever a task is sent off the simulation task queue, it moves to the target object, which could be an air traffic control processor or an airline task processor etc. Once the respective task processor receives it, the task will then be dispatched again to different processing channels depending on its secondary task identifier. For example, the airplane task processor will dispatch an airplane task into the following built-in sub-task process channels such as, moving, holding, parking, or landing. The secondary label is used in the secondary branching process.

c. **Priority data set**: Tasks queue up in the task engine according to their assigned priorities. Tasks are dispatched in the order of their priorities rather than the order that they are created. The default priority is the time mark at which the task is supposed to be dispatched. Whenever a task is created, the time mark at which the task is to be dispatched is set. Whenever a task is added to the task queue, the simulation engine will automatically search for a right position in the task queue and add the task in that position.

Task supplemental data includes the following: the information of time, task primary label, secondary label, and contents of actions. For example, a task that orders an airplane to move from point A to point B contains the following information: a primary label to indicate this is an airplane task, a time mark to tell when the airplane will move, a secondary task label to indicate that this is a moving task, and a set of instructions attached to the task object about information of point A and B as well as other necessary messages.

3.3.1 Airplane Related Tasks

Landing Task: Air traffic control tower sends the landing permission to an approaching airplane, upon receiving the landing permission, the airplane proceeds to land on the runway.
Destination: The targeted airplane.
Label: Land_an_Airplane_Task
Task Data: The identifier is airplane's ID and the supplemental data includes designated runway and gate.

Exiting Task: When an airplane leaves a network segment such as a runway or a taxiway, it sends a task to a network link to change the occupancy status of the link.
Destination: The targeted runway, taxiway or gate.
Label: Exit_a_Link_Task
Task Data: The identifier is the airplane ID and the supplemental data includes the runway IDs, taxiway IDs or Gate ID of current link and next link.

Entering Task: When an airplane enters a network segment such as a runway or a taxiway, it sends a task to a network link to change the occupancy status of the link.
Destination: The targeted runway, taxiway or gate.
Label: Enter_a_Link_Task
Task Data: The identifier is airplane's ID and the supplemental data includes the runway IDs, taxiway IDs, or gate IDs of current link and next link.

3.3.2 Traffic Network Related Tasks

Receiving an Object Task: When a moving airplane enters a traffic network such as a runway, taxiway or gate, these network objects receive a signal that tells that an airplane is moving into its internal queue.
Destination: The targeted traffic network object such as a runway, a taxiway and a gate.
Label: Receive_an_Object_Task
Task Data: The identifier is the network object ID and the supplemental data includes the entering airplane ID and time at which the airplane enters the object's internal queue.

Releasing an Object Task: When an airplane leaves a traffic network object such as a runway, taxiway or gate, these network objects receive a signal that tells an airplane is moving out of the their internal queue.
Destination: The targeted traffic network object such as a runway, taxiway and gate.
Label: Release_an_Object_Task
Task Data: The identifier is the network object ID and the supplemental data include the airplane's ID and time at which the airplane leaves the object's internal queue.

Close_This_Link Task: When a traffic network object such as a runway, taxiway, and gate is not available to any traffic, the network object is closed.
Destination: The targeted traffic network object.
Label: Close_This_Link_Task.
Task Data: The identifier is the targeted traffic network object's ID and the supplemental data includes the time at which the targeted traffic network object is closed.

Open_This_Link_Task: When a traffic network object such as a runway, taxiway, and gate becomes available to serve any traffic flow and temporary storage, the network object is open.
Destination: The targeted traffic network element.
Label: Open_This_Link_Task.

Task Data: The identifier is the targeted traffic network element's ID and the supplemental data includes the time at which the targeted traffic network element is open.

3.3.3 Traffic Control Related Tasks

Moving an Airplane Task: The air traffic control tower sends a command to an airplane in the traffic network to move on, the task is triggered.
Destination: The targeted airplane.
Label: Move_an_Airplane
Task Data: The identifier is the airplane's ID and supplemental data include the original link and the next link, starting time, and ending time.
Holding an Airplane Task: The air traffic controller sends a command to an airplane in the traffic network to hold its position, the task is triggered.
Destination: The targeted airplane.
Label: Hold_an_Airplane_in_Position
Task Data: The identifier is the airplane ID and the supplemental data include the designated link and starting time.
Parking an Airplane Task: The air traffic control tower sends a command to an airplane in the traffic network to park at a gate, the task is triggered.
Destination: The targeted airplane.
Label: Park_an_Airplane.
Task Data: The identifier is the targeted airplane ID and the supplemental data includes the designated gate ID, and the starting time.

3.4 Classes in the Model

3.4.1 The Simulation Clock Class

A simulation clock is a global time object which governs the simulation process for a preset time frame. A global clock must be able to communicate with other objects such as task, airplanes, and air traffic control. At the same time the global clock must be an independent object. A global clock consists of a global time structure which has hour, minutes, and seconds; and methods to advance the clock, set the time, and get the current time. Methods of getting and setting the global clock allow other objects to communicate with the clock, especially the task engine which dispatches a task according to the current clock time and the time mark set in the task objects.

3.4.2 Airplane List Class

An airplane list object contains a list of airplane object pointers each of which points to the individual airplane object and several methods to execute various tasks. These methods are:
a) a method to create an airplane for scheduling a arrival of departure
b) a method to assign an airplane ID number to a newly created airplane
c) a method to add an airplane into the list
d) a method to remove an airplane from the list
e) a method to send a task to the target airplane with given airplane ID and,
f) a method to get the target airplane object with a given airplane ID.
The airplane list object provides all the management functions to control every aircraft in the simulation.

3.4.3 Airplane Class

An airplane can have in one of a number of states during the simulation. For example it can be parked at a gate, taxi to the runway and depart, or on the opposite direction, an airplane can land, taxi out of the airfield network, and park at the gate.

Airplanes in most airport simulations are categorized into two flows, arrival and departure. An airplane in the arrival flow has valid states such as final approach, touch down, decelerating on the runway, network taxiing, holding, parking at the gate. An airplane in the departure flow has feasible states such as parking at the gate, taxiing on the airport ground network, holding, taking off, moving in the air terminal and leaving the air terminal. Figure 1 shows states of an airplane.

An object of the airplane class contains a set of variables: 1) airplane ID, 2) airplane orientation in the network, 3) the point at which the airplane enters the simulated airport traffic network, 4) airplane destination at which the airplane leaves the simulated traffic network, 5) the airplane current position, 6) next position, 7) total travel time or abstract type of total travel cost, 8) assigned best travel path, and 9) destination indicator. The basic attributes are illustrated in the following piece of source code.

```
class Airplane_class {
// airplane ID
int ID;
//airplane orientation
int orientation;
//airplane destination
int destination;
//current starting point
int from;
//next point
int to;
//total travel cost
int total_cost;
//destination indicator
int reach_destination;
//best travel path
char path[60];
//time mark at which the air
plane enter the simulation
time_type start;
//time mark at which the air
plane reaches the
//destination
time_type end;};
```

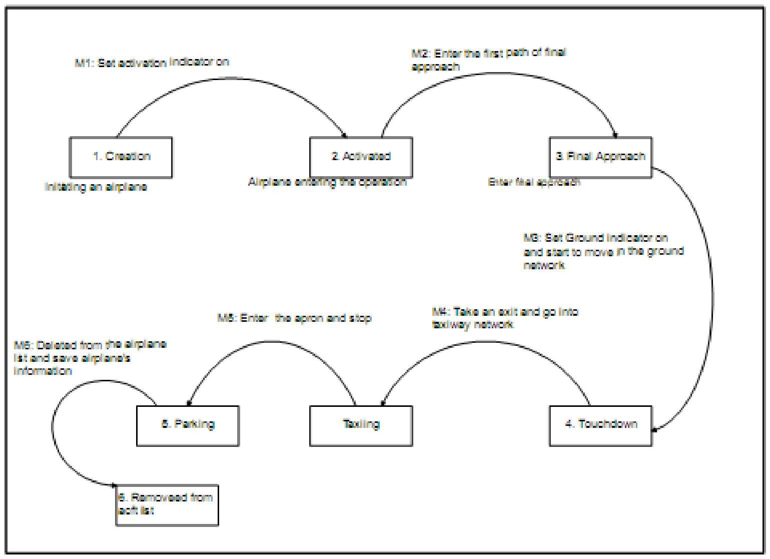

M1: Set activation indicator on
M2: Enter the first path of final approach
1. Creation
2. Activated
3. Final Approach
Initating an airplane
Airplane entering the operation
Enter final approach
M3: Set Ground indicator on and start to move in the ground network
M5: Enter the apron and stop
M4: Take an exit and go into taxiway network
M6: Deleted from the airplane list and save airplane's information
5. Parking
Taxiing
4. Touchdown
6. Removeed from acft list

Figure 1: Airplane State Chart

Methods of airplane class are shown in Figure 2.

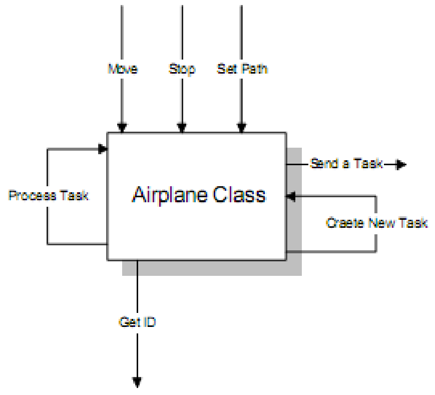


Figure 2: Airplane Methods

3.4.4 Airport Ground Network Class

Airport ground networks include runways, taxiways, and gates. During the simulation, airplanes move through the network following the rules of air traffic control and physical laws. In the airfield simulation framework, airport ground network objects are an abstract data type which provides ground network topology information to other objects such as airplane list, shortest path processor, and traffic information collector objects.

Whenever an airplane moves into a link in the network, the state of the link is changed from empty to occupied and the airplane ID is added into the airplane ID list associated with that link. A ground traffic flow information collector can query current traffic flow information to verify which link is empty and, if not, how many airplanes and which airplanes are there. By collecting all the needed traffic flow information, end users can easily apply their knowledge based traffic control rules or even optimization algorithms to control the airport ground traffic flow. Airport ground network serves as an information provider of both ground network topology and traffic flow. Figure 3 illustrates the message flow capabilities of the airport network class.

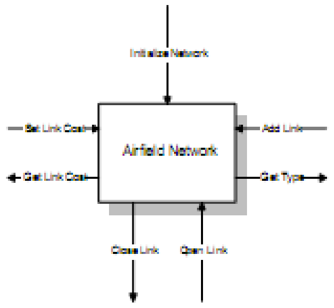


Figure 3: Network Methods

Network objects have associated states. Runways normally have two states. A runway can be opened or closed by the air traffic control tower. Physically runways have two directions of operation. Each direction could be set either open or close. If a runway is open, it will be further defined as occupied or available.

Taxiways also have two states: opened or closed. Each taxiway can also be used from two directions. Each

direction could be set either open or close. If a taxiway is open, it will be further defined as occupied or available. Gates normally have two states. A gate can be opened or closed by the traffic control tower. If a gate is open, it will be further defined as occupied or available. An open gate is simply a gate that is available to upload or download cargo or passengers and a closed gate is either closed or already occupied by an airplane.
The following definitions apply in the modeling framework.
Runway Open: When a runway is open, normally it will be available for the incumbent arriving or departing airplane.
Runway Close: When a runway is close, it could be occupied by an airplane or closed by air traffic control.
Taxiway Open: When a taxiway is open, normally it will be available for an airplane to pass.
Taxiway Close: When a taxiway is close, it could be occupied by other airplanes or closed by air traffic control.
Gate Open: When a gate is open, normally it will be available for an airplane to be parked.
Gate Close: When a gate is close, it could be occupied by an airplane or closed by the air traffic control tower.
Final State: When the simulation stops, the runway, taxiway, or gate objects become quiescent. They continue to exist, but have no subsequent dynamic behavior.

3.4.5 Shortest Path Class

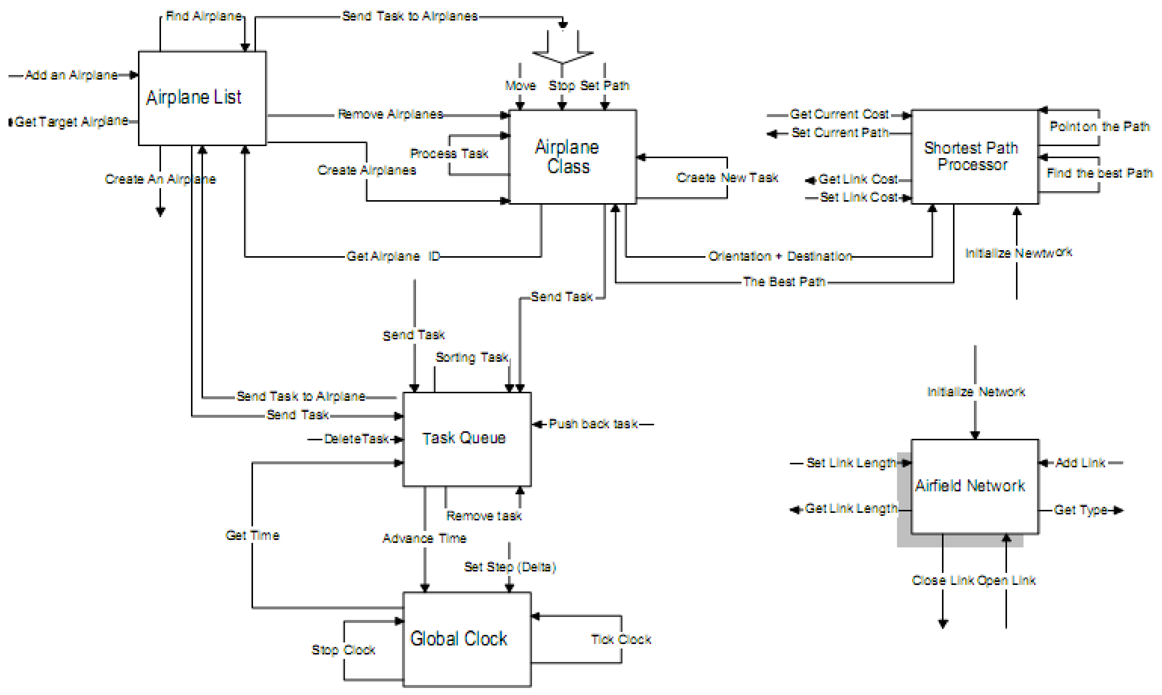
The shortest path method is an algorithm that can search a feasible path in a given abstract network with the minimum total travel cost. The end users can easily test different air traffic control methods to maximize the ground network throughput capacity and reduce delay.
The travel cost of any given link could be dynamically changed to represent the real time situations (such as traffic congestion). The basic rule or objective is to minimize the total travel cost between two given points. However, the minimum travel cost rules could be overwritten with other rules, such as maximizing the traffic flow between two given points.

```
class Path_finder_class {
public:
typedef struct atom_struct_def {
char from; char to;
char type; int cost;
} atom_path;
............
private:
atom_path **traffic_net;
int n_max_segments;
int n_segments;
char *current_path;
int current_cost;
int num_points_on_path;
char *best_path;
long best_path_cost;
int num_points_on_the_best_path;
char orientation; char destination;
int head_tail_indicator;
};
```
.

3.5 Other Possible Classes

a. Airplane Flight Scheduler
b. AirfieldTrafficFlowAnimator (graphically presents the movements of all airplanes in the airfield network)
c. ControlLogic (contains all of the control logic that is implemented by the end users)
d. TrafficParam (sets the parameters for the simulation including aircraft mix, category mix, runway occupancy time, exit choice distribution, and travel times for each taxiway segment in the ground network.

Overall structure of the simulation engine is shown in Figure 4.


Find Airplane
Send Task to Airplanes
Add an Airplane
Airplane List
Get Target Airplane
Create An Airplane
Move
Stop
Set Path
Remove Airplanes
Process Task
Create Airplanes
Airplane Class
Create New Task
Get Current Cost
Set Current Path
Shortest Path Processor
Point on the Path
Find the best Path
Get Link Cost
Set Link Cost
Get Airplane ID
Orientation + Destination
The Best Path
Initialize Newtwork
Send Task
Send Task
Sorting Task
Send Task to Airplane
Send Task
DeleteTask
Task Queue
Push back task
Remove task
Get Time
Advance Time
Set Step (Delta)
Global Clock
Stop Clock
Tick Clock
Initialize Network
Set Link Length
Get Link Length
Airfield Network
Add Link
Get Type
Close Link
Open Link