Theses and Dissertations   Student Graduate Works

3-1999

Modeling and Analysis of Aerial Port Operations

Timothy W. Albrecht

MODELING AND ANALYSIS
OF AERIAL PORT OPERATIONS

THESIS

Timothy W. Albrecht, Captain, USAF

AFIT/GOR/ENS/99M-1

19990409 030

MODELING AND ANALYSIS
OF AERIAL PORT OPERATIONS

Timothy W. Albrecht, B.S.
Captain, USAF

Approved:

_____   _____
Chairman   date
3/11/99

_____   _____
                                                date
3/9/99

AFIT/GOR/ENS/99M-1

MODELING AND ANALYSIS
OF AERIAL PORT OPERATIONS

THESIS

Presented to the faculty of the Graduate School of Engineering

of the Air Force Institute of Technology

Air University

Air Education and Training Command

In Partial Fulfillment of the Requirements for the

Degree of Master of Science (Operations Research)

Timothy W. Albrecht, B. S.

Captain, USAF

March 1999

Acknowledgments

The construction of my thesis was buttressed with the support, guidance, and instruction of several people. To Dr. Ken Bauer, my thesis advisor, and Lt Col J.O. Miller, my thesis reader I owe a debt of gratitude for sharing their time, experience, and nurturing academic environment. To Maj Robert Brigantic, the sponsor of my thesis, I owe my appreciation for allowing latitude in my approach to this thesis, for providing timely response to any questions I had, and for financially supporting my pursuit of the best product. Finally, I wish to say, "Thank you" to my wife, Amy, for her patience and understanding in the midst of wedding planning, long hours of homework, and thesis drudgery.

<div align="right">

Timothy W. Albrecht

</div>

Table of Contents

List of Figures

List of Tables

AFIT/GOR/ENS/99M-1

Abstract

The focus of this thesis effort is gaining useful insight into aerial port operations by employing an animated simulation. Understanding airfield capacity, resources, and functioning allows greater accuracy and efficiency in both planning for future force structures and matching mobility assets with commanders' objectives. Two current simulations, ACE (Airfield Capacity Estimator) and BRACE (Base Resource Allocation and Capabilities Estimator), model mobility activities at the base level with some deficiencies. The model proposed by this thesis, APOM (Aerial Port Operations Model), will provide the mobility analyst an animated simulation with two, new measures of aerial port operations; a real-time estimate of airfield capacity subject to changing levels of airfield resources, and an instantaneous count of serviced aircraft (service MOG). Additionally, APOM will offer an expanded utility to the mobility analyst by modeling a ground transportation network associated with the aerial port.

MODELING AND ANALYSIS OF AERIAL PORT OPERATIONS

I. Introduction

The advantages gained by improving strategic mobility analyses are straightforward. One can maximize the objective of movement of force by improving the accuracy and efficiency of mobility planning. But, what steps can be taken to improve mobility planning?

At the root of strategic mobility planning is an understanding of the operation of an air mobility base. Once the functioning of an airbase can be understood with sufficient fidelity and flexibility, a set of them can be brought together to form a transportation network overwhich a planner can route cargo flow. Capabilities of the transportation network constrained by mobility base resource levels can then be studied.

The topic of this thesis evolved from a need to develop a useful base-level mobility model while recognizing the existence of two such models, BRACE and ACE. This thesis proposes to introduce an additional modeling capability to the BRACE simulation; that of cargo hand-off from airbase material handling equipment (MHE) to ground transport. Further, the airbase will be modeled as a source of a transportation network leading to cargo end nodes. By incorporating a transition between airbase cargo handling and a ground-based transportation network the applicability of the model will be expanded.

1

In addition to the incorporation of a transportation network, the model will include meaningful output functions related to mobility study such as a dynamic plot of airfield capacity and an airfield service measurement. Animation of the aerial port will allow the user to follow the activities at the airfield, to change resource levels, and to observe the impact of new resource levels.

Creating a model built on the capabilities and deficiencies of previous models requires a disciplined approach and an understanding of the user's needs. The methodology used in developing APOM was drawn from the literature on classic simulation study and constitutes the bulk of this thesis effort. And the author understands that there is no defensible justification for the development of a simulation model if it is not ultimately used and usable by the decision maker (Shannon, 1975).

II. Literature Review

Research for this study revolved around several different issues. First, the background supporting mobility studies was explored. Second, the framework and techniques of simulation studies was learned. And third, the details of model translation were researched.

The reference material gathered for this study concerning mobility studies aided in understanding the function of mobility bases and the importance mobility planning has in US force projection. Primary sources were RAND studies providing airfield operation overview (Stucker, 1998), mobility modeling and analysis (Schank, 1991), and application of mobility modeling analysis (Killingsworth, 1997).

The second area of research yielded an understanding of simulation studies. One quotation from a simulation expert shows that time has not influenced the basic process of modeling:

> The process of model development may be usefully viewed as a process of enrichment or elaboration. One begins with very simple models, quite distinct from reality, and attempts to move in evolutionary fashion toward more elaborate models which more nearly reflect the complexity of the actual management situation. (Morris, 1967)

This area of research revealed a method in conducting a simulation study. Guidance in following the steps of this method from problem formulation, to model translation, to verification and validation were essential in focusing the thesis. Secrets of Successful Simulation Studies (Law, 1990)

3

provided the best information on the subject, but several other sources also supported the topic. Among these were System Simulation (Gordon, 1969) and Discrete-Event System Simulation (Banks, 1996).

The final area of interest was accurate and efficient translation of concepts into a coded model. The model concept was developed from research on the ACE and BRACE models and occurred in many stages starting with a very basic air mobility process and ending with a detailed aerial port model.

Two parallel tracks were run in regard to model translation. First, understanding of the ACE and BRACE models was needed. The BRACE model code and user's guide revealed its structure and capability (Cusick, 1997). Since the treatment of cargo was of primary concern, focus was centered on the possibility of appending the BRACE code to include a cargo hand-off to a ground transportation network. Study of the code showed BRACE treated cargo as a counter that was incremented or decremented upon onload or offload. If a ground network were added, cargo would need to be treated as an object to facilitate data collection and simplify the code structure.

Information on the ACE model was found in the RAND study, Understanding Airfield Capacity for Airlift Operations (Stucker, 1998). Gathered in this study was a wealth of information on the ACE model covering its structure, operation, and purpose.

Deficiencies in the ACE and BRACE models, outlined later in this thesis, were illuminated in the thesis work of Maj David Williams (Williams, 1999). The problems found with the ACE and BRACE models aided both the model conceptualization and the model translation phases of APOM.

The second parallel track taken was learning object-oriented simulation programming in MODSIM III. Invaluable resources in this effort were the MODSIM reference manual, SimGraphics manual, and the MODSIM training seminar. Areas of primary concern were resource management, trigger mechanisms, graphical user interfaces, and animation issues.

III. Methodology

The literature is rich with information on conducting a proper simulation study and the benefits of following the prescribed steps. The following subsections outline the course of the thesis simulation study with an introductory paragraph explaining the purpose of each step. Figure 1 shows the flowchart of a simulation study (Law, 1991).

Problem Formulation

The beginning of every simulation study is a clear statement of the study's objectives. It is necessary to specify a clear goal and any issues that need to be addressed.

The focus of the thesis effort is gaining useful insight into aerial port operations beyond those provided by current simulations (ACE and BRACE) as they relate to air mobility operations by appending the list of modeled processes with a ground transportation network capability. Additionally, improving the representation of airfield capacity measurement is desired.

Project Plan

The overall study should be planned in terms of the number of people, the cost, and the time required for each aspect of the study (Law, 1991). In the case of this thesis, the number of people directly involved was four (author, advisor,
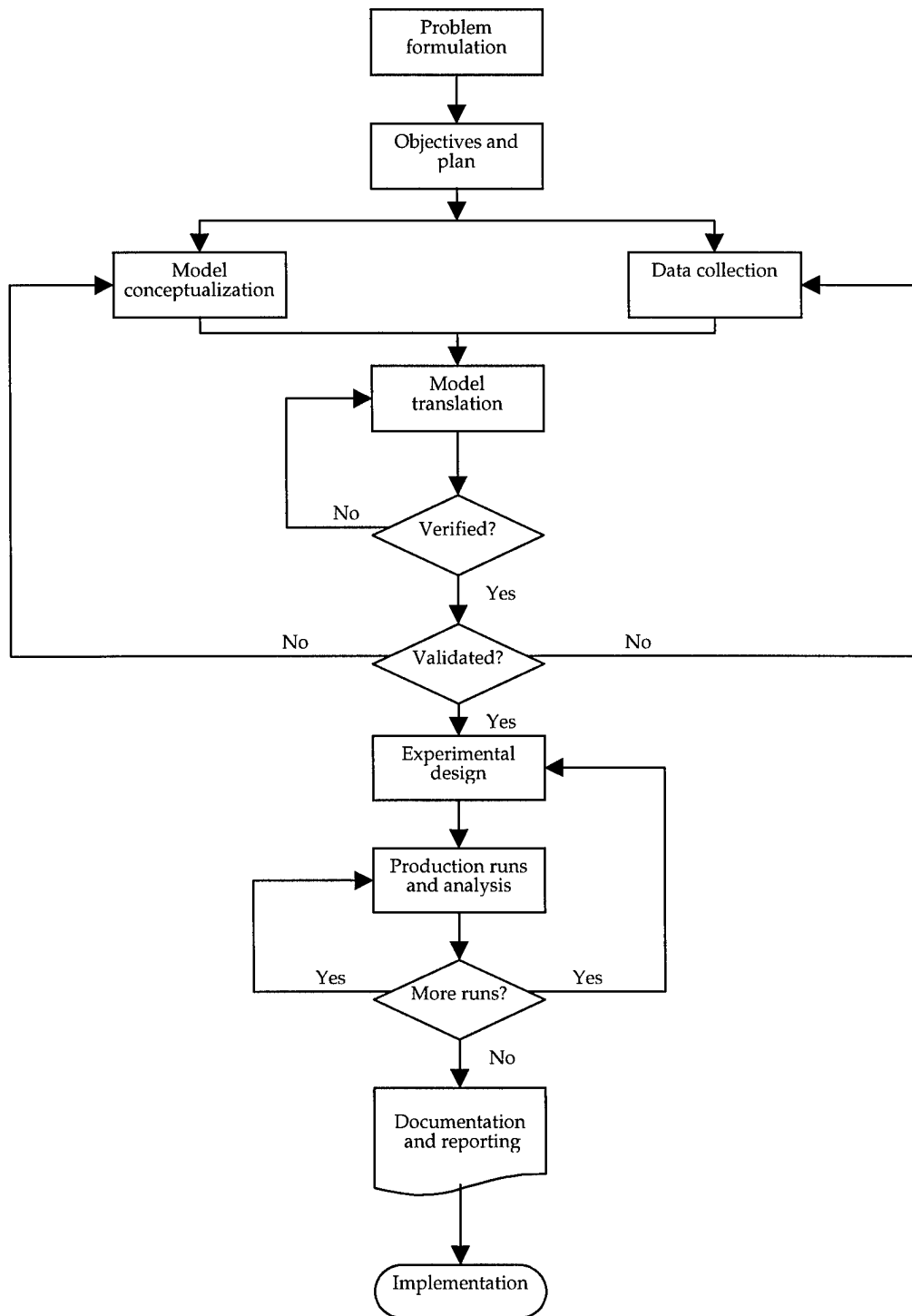
Figure 1 Simulation Study Flowchart

reader, and sponsor), the cost was minimal (less than $1000.00), and the time required was under six months.

It is useful to divide the planning of this simulation study into three stages: gaining a direction of study, understanding the capabilities and limitations of the ACE and BRACE models, and determining the composition of the thesis simulation, APOM.

Thesis Direction

This thesis began with the idea of adding the transition of cargo from an aerial port onto a ground transportation system to the BRACE simulation. As it stood, BRACE did not deal with cargo after it had been offloaded from an aircraft. It was decided that adding this capability to BRACE would result in a more robust model useful in planning force deployments beyond the mobility base.

BRACE is written in an object-oriented simulation language owned by CACI Inc. called MODSIM III. Implementing a new process would require a familiarity with MODSIM III, and any changes to the code would require a MODSIM III license. The thesis sponsor, HQ AMC/XPY, provided the author with both a seat at a week-long MODSIM III training seminar and a MODSIM III license.

With the necessary programming skills learned and a goal in mind, work began on a proof of concept model to show BRACE could be altered to incorporate a transition to a ground transportation network. As the proof of concept code grew and difficulties with BRACE surfaced, it became evident that the thesis simulation could be managed with greater dexterity than the original BRACE code.

Model Backgrounds

The Base Resource Allocation and Capabilities Estimator (BRACE) is a capacitated queuing network simulation written in MODSIM to model an aircraft arrival stream, a mobility airfield's ground activities and resources, and the relationship between resources and aircraft/cargo throughput.

The BRACE model is stochastic with variation found in the aircraft arrival stream. The duration of all ground-based processes, except maintenance, are deterministic and depend on aircraft parameters and resource availability.

BRACE was written by Travis Cusick at Washington University's (St. Louis) Center for Optimization and Semantic Control under the sponsorship of the Air Force's HQ AMC/XPY. The current version, 1.31, is dated 18 Dec 1997 (Cusick, 1997).

The activities modeled include aircraft type and arrival, parking, fueling, maintenance, and departure, as well as cargo and passenger on/off loading.

Ground-based equipment modeled by BRACE includes fuel trucks, fuel pumps, 60k, 40k, and 25k loaders, forklifts, wide body elevators (WBELs), and ramp spaces.

Aircraft arrival may be exponential, Erlang, or triangular distributed. BRACE models C-130, C-17, C-5, C-141, KC-10, B-747, or DC-8 airframes each with a default or user-defined payload, maintenance record, and fuel parameters.

BRACE also allows the user to restrict the operating hours of the airfield, to determine aircraft divert protocol, and to define resource availability.

BRACE outputs information on throughput, resource utilization, delays, activity times, and ground times.

The Airfield Capacity Estimator (ACE) is a spreadsheet model developed by James Stucker and Ruth Berg at RAND's National Defense Research Institute to estimate the capacity of an airfield. The current Microsoft Excel version, ACE 97, was completed on 1 November, 1998 (Stucker, 1998).

ACE inputs include airfield specific parameters, global parameters, and mission specifications. The airfield specific parameters include information on airfield layout, aircraft fueling parameters, and aircraft loading parameters. Global parameters contain information on aircraft characteristics, ground equipment, and global aircraft servicing times, loading parameters, and fuel parameters. Mission specifications include number and type of aircraft, aircraft

configuration (cargo, passenger, or mixed), servicing profile (quick turn or full service), fuel required, and passengers and cargo to be on-loaded and/or off-loaded.

ACE can be run in one of two modes; expected-value mode or the Monte Carlo mode. The expected-value mode is deterministic. The Monte Carlo mode includes limited stochastic effects. Variability is introduced in determining whether aircraft require repair, nitrogen servicing, oxygen servicing, and de-icing, and the time required to accomplish these activities.

ACE determines the daily capacity of a particular airfield through a series of mathematical calculations within various spreadsheets. ACE calculates the average time required per aircraft per service activity (fueling, cargo loading/offloading, and maintenance). Next, an average ground time is determined which is then incorporated in a calculation of an average daily capacity for each resource based on resource availability. The service capacity is determined for each airfield service. The limiting service capacity determines the airfield's overall capacity.

The use and limitations of the ACE and BRACE models are the focus of fellow ENS student Major David Williams' thesis. The following tables of limitations of the models were uncovered through his research effort (Williams, 1999).

Table 1 BRACE Model Limitations

Model Area	Implication
AC delay time output	Calculated to be the difference between actual time on ground and expected time on ground. Not defined as the time spent waiting for service.
AC divert mechanism	BRACE includes a diverted aircraft as one which has been serviced, inflating the number of aircraft serviced, amount of cargo loaded/offloaded, and fuel transferred.
AC serviced output	BRACE reports any AC arrival as a serviced AC. For accuracy, a departed AC should be counted as serviced.
AC servicing order	BRACE allows nearly all servicing activities to occur simultaneously. Erroneous representation of reality where fueling is performed separately from other services.
Airfield capacity measure	BRACE lacks a direct measure of airfield capacity.
Forklift modeling	Simulation runs with no forklifts still show transfer of cargo. Indications of error in modeling of forklifts.
Fuel pit	Does not model hydrant servicing vehicles. Possible for aircraft to land and wait the duration of the simulation for fueling.
Ramp space utilization	Wide-body ramp space utilization calculations are erroneous.
Run length	BRACE requires user attention during run-time to ensure correct termination (see below).
Termination	Once BRACE reaches the user-defined termination point, it continues to run until the airfield is flushed of AC. This corrupts the data collected.
Input Parameters	BRACE input parameters are not readily available. Contributes to lack of model use.
Model Use	BRACE is not used for routine analysis for a variety of reasons. Improvements to the model are needed.

Of the limitations listed in Table 1, several warrant restating. First, the divert mechanism in BRACE treats diverted aircraft as if they had been serviced. The simulation includes the cargo and fuel that would have been transferred in its output, thus inflating throughput statistics. Second, the fueling of aircraft

should occur separately from any other service. Third, a proper termination of the simulation is needed to aid analysis. And last, a direct measure of airfield capacity is needed.

Table 2 ACE Model Limitations

Model Area	Implication
Deterministic nature	Uses expected values in calculations. Results are "optimistic" or upper limit capacities.
Fuel Trucks	ACE inaccurately models the use of fuel trucks. Use of fuel trucks does not impact simulation results.
Variety of AC within mission	AC within a mission must have the same parameters (i.e. same aircraft). Limits scope of ACE studies.
Model Use	ACE is not used for routine mobility analysis.

The most significant problem with ACE is its modeling of fueling operations. ACE allows the user to include fuel trucks and hydrant servicing vehicles (HSV's) in its modeling of aircraft fueling. Through repeated trials the number of fuel trucks made no difference in the average amount of time required to fuel an aircraft, or to the airfield's capacity for fueling operations (Williams, 1999).

Thesis Simulation

The next step in the thesis process was deciding a direction for the thesis simulation. The capabilities and shortcomings of the ACE and BRACE models along with the idea of appending a ground transportation network provided a

recipe for a useful, first generation air mobility base simulation. It was decided to evolve the code for the original proof of concept model by adding to it those capabilities deemed useful in ACE and BRACE while avoiding those pitfalls illuminated by Maj Williams' research. The new model would be structured and documented to lend itself to future enhancements.

Following the logic above, the thesis simulation must model to a reasonable level of detail both the flow of cargo into the aerial port and the operation of the transportation network leading from the aerial port to final destinations. Such details include accurate modeling of aircraft traffic flow, passenger offload, aircraft fueling, aircraft maintenance, aerial port cargo handling (bulk and rolling), and cargo flow over the transportation network. In addition, the simulation should be animated, employ a concrete divert mechanism, terminate succinctly, and include a direct measure of airfield capacity.

Model Conceptualization

This step consists of distilling the complex system to be modeled to its essential components and involves feedback from the perspective user during the process. An additional consideration in this step is inclusion of those aspects of the system that are relevant to the study objectives (Gordon, 1969).

Modeled Behavior

APOM models the flow of cargo through an airfield and into a specified ground transportation network. This process can be broken down into three parts: the arrival of aircraft, the handling of cargo, and the distribution of cargo.

The arrival times of aircraft are randomly generated according to an exponential distribution with user-determined mean value. Two types of aircraft are modeled, the C-17 and the C-5. Further enhancements to the model could expand the choice of aircraft easily. A random draw determines the type of aircraft arriving to the airfield according to a user-defined proportion of traffic flow. At instantiation each aircraft is given a predetermined set of cargo, passengers, and required fuel load depending on aircraft type.

An approach to the airfield is made when two conditions are met. First, that the maximum allowed number of aircraft on the ground will not be exceeded by landing the aircraft, and second, that the runway is not in use by another aircraft. The maximum allowed number of aircraft on the ground is defined by the user and includes all aircraft who landed, who are awaiting service by the cargo handlers, who are currently being serviced by cargo handlers, or are taxiing to takeoff. Not included are aircraft on their takeoff roll. A divert occurs when the maximum allowed number of aircraft on the ground denies the aircraft approach. In this way, the simulation avoids boundless aggregation of aircraft should the airfield resources be insufficient to handle the

aircraft flow. If the runway is in use at the time of approach (departing aircraft is taxiing across the runway) the approaching aircraft does not divert, but simply orbits until the runway is available.

Once an aircraft has landed and taxied to a preset location, it awaits assignment to a parking space. A parking space must be assigned before servicing can be initiated. The number of parking spaces on the ramp is user-defined and can be altered at any time during the simulation run.

After the aircraft is assigned a parking space it undergoes four processes: passenger offload, cargo offload, refueling, and concurrent maintenance. Passenger offload occurs first and is concurrent with cargo offloading and maintenance. Refueling begins once all other processes are complete.

Both passenger and rolling cargo offload are simple processes incurring delays proportional to the number of units offloaded. These processes involve no airfield resources.

The palletized cargo offloading process begins with the aircraft awaiting the service of the airfield's cargo handlers. Once assigned cargo handlers, the service icon is colored yellow. Cargo handlers are assigned to aircraft according to the following rule. If there are more cargo handlers available than the aircraft requires for a single-run-offload (defined as the number of cargo handlers required to remove all palletized cargo without returning to aircraft), then the aircraft is assigned its full compliment of cargo handlers. If the aircraft requires

more cargo handlers than the airfield has available it will grab those cargo handlers which are available.

Once assigned a number of cargo handlers the aircraft will retain possession of that number until its palletized cargo has been offloaded. The assigned team of cargo handlers travels to the parked aircraft (a delay is incurred as a function of aircraft location), queue at the aircraft, and offload the pallets at a user defined rate. The team of cargo handlers travels back to the loading dock, awaits assistance from available forklifts, and places the pallets onto the dock.

At the time of offload, the simulation instantiates a cargo object with a birth time, a cargo type (pallet, rolling, or passenger), and a final destination. The birth time is used to track the time in system of each piece of cargo. The final destination of each piece of cargo is determined randomly according to a user-defined distribution. The default setting is for equi-probable destinations.

Aircraft maintenance is determined by user-defined data defining probability of breakdown and duration of repair. Once maintenance and cargo offloading have been completed, the aircraft prepares to be refueled.

Aircraft refueling can take place by fuel pit or fuel truck depending on the availability of fuel pits. If a fuel pit is available, fuel is transferred at a user-defined rate until the aircraft is filled. If a fuel pit is not available, fuel trucks must be assigned to refuel the aircraft. Once assigned to an aircraft a fuel truck

will deliver its fuel at a user-defined rate until it is empty (returns to loading dock to refill) or the aircraft is filled.

The simulation also includes a non-concurrent maintenance feature that determines whether an aircraft will require repair and the duration of the repair (both user-defined). Maintenance is carried out after the aircraft has offloaded its cargo and been refueled.

After being offloaded from the aircraft, the cargo are sorted according to destination and type and placed in a FIFO queuing system. Once the number of cargo heading to a particular destination reaches a user defined level a call is placed for a transport to haul the cargo.

There are a user-defined number of transports available at the airfield. If one is available, it is dispatched. If one is not available, the cargo waits in a FIFO manner until one becomes available. With a transport available the cargo are loaded by forklift and the transport departs for its respective destination. After an offload delay, the transport returns to the loading dock for further use.

Graphical Layout

The features of the airfield layout include: two runways (one takeoff and one landing), one ramp for aircraft parking, an aerial port facility, and a simple ground transportation network with four end nodes.

There are several simulation meters displaying pertinent simulation data at run time. These meters are grouped into three areas of the simulation board: the bottom display, the aerial port display, and the chart area. The bottom display contains the simulation clock and a divert counter.

The aerial port contains four counters that display the number of cargo handlers available, the number of fuel trucks available, the number of forklifts available, and the number of trucks available in the motorpool. Additionally, a parking space queue meter is located on the ramp and displays the number of aircraft awaiting ramp space assignment. The aerial port area also contains a dynamic, loading dock operations chart showing the current levels of cargo on the dock organized by cargo type.

The chart area contains three dynamic displays. The first is an aircraft time-in-system histogram. The second is a real-time estimate of airfield capacity. And, the third is a capacity measure called service MOG.

The airfield capacity display allows the user to see the progression of airfield capacity estimates as the simulation runs. With each aircraft departure, the display is updated with a new estimate of airfield capacity. The new capacity is determined according to the following equation:

$$C(t) = \frac{D(t)}{t} \cdot 1440$$

where C(*t*) is the estimated airfield capacity at time *t* measured in aircraft per day, D (*t*) is the aggregate number of aircraft departures at time *t*, *t* is the simulation time measured in minutes, and 1440 is the number of minutes in one day.
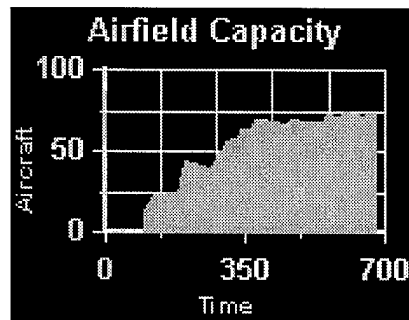
Figure 2 Airfield Capacity Display

The service MOG display shows the user the number of aircraft on the ground in service at one time. The graph updates whenever a service state change occurs and can never be greater than the number of ramp spaces available at the airfield. The model reports the service MOG according to the following equation:

$$S_{MOG}(t) = AC_{park}(t) \cap AC_{ser}(t)$$

where $AC_{park}(t)$ and $AC_{ser}(t)$ are the number of aircraft parked and the number of aircraft in service at time *t*, respectively.

The menu bar provides the user with flexible control of the simulation. There are four choices at the menu bar; the *control* button, the *settings* button, the *window* button, and the *info* button.
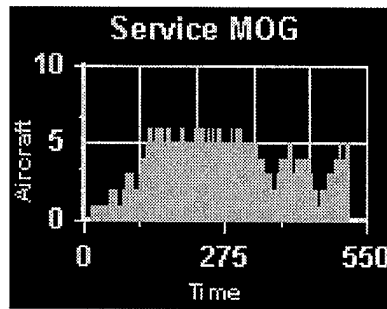
Figure 3 Service MOG Display

The *control* button gives the user start/stop options. Start simulation will begin the flow of aircraft into the airfield according to the default parameters or those specified by the user. Stop simulation halts the simulation and provides the user with the summary statistics before exiting the simulation.

The *settings* button allows the user to change parameters in four areas: airfield, aircraft, cargo, and transportation network. Choosing the airfield button opens a dialog box through which the user can change the simulation parameters such as aircraft interarrival time, airfield resource levels, simulation end time, and simulation speed. The aircraft parameter dialog box allows the user to define the amount of cargo carried on each aircraft as well as maintenance descriptors. The cargo parameter dialog box allows the user to define the distribution of cargo, the capacity of the cargo handlers, and the time to unload cargo. The transportation network parameter dialog box allows the user to define the distance to each of the four cargo destinations as well as the amount of cargo carried by a single truck.

The *window* button allows the user to select either airfield statistics or destination statistics. The statistics kept by the simulation are detailed below. Destination statistics outline the type/amount of cargo at each of the four cargo destinations.

The *info* button calls a brief message regarding the origin of the simulation to the screen.

Data Collection

The simulation reports on various statistics throughout a run. The statistics are grouped into three areas: airfield statistics, cargo statistics, and ground transport statistics. Table 3 lists the simulation statistics.

The research on the ACE and BRACE models revealed the importance of several types of output data. Resource utilization statistics, aircraft arrival, departure, and divert, and a direct measure of airfield capacity were specifically cited as necessary output statistics.

Model Translation

Model translation refers to the technical process of converting a conceptual model into a functioning simulation.

APOM was written using MODSIM III for PC's by CACI Inc. The code is broken up into six modules. The three primary modules describe the airfield, the aircraft, and the cargo. Two functional modules handle global variables and the

graphics employed by the simulation. The final module, the main module, initializes the simulation.

Table 3 Simulation Statistics

Airfield	
Aircraft arrivals	the number of aircraft having landed at the airfield
Aircraft departures	the number of aircraft having left the airfield
Aircraft diverts	the number of aircraft diverting from the airfield (violation of the max on the ground parameter)
Aircraft breakdowns	the number of aircraft experiencing a maintenance problem
Aircraft time in system	dynamic histogram showing the number of aircraft experiencing a given time in system (from landing to takeoff)
Airfield capacity	dynamic chart showing the current estimate of airfield capacity (aircraft serviced in 24 hr period)
Number available	real-time display of fuel trucks available
Output file	APOM writes airfield capacity data to an output file, "output.txt"
Service MOG	dynamic chart showing the number of aircraft in service
Queue/Resource statistics	mean and max length of parking space queue, mean wait time for parking space, utilization of parking space, cargo handler, forklift, and fuel truck resources
Cargo	
Number available	real-time display of forklifts available
Number available	real-time display of cargo handlers available
Loading Dock Display	dynamic chart showing current amount of cargo on the loading dock by cargo type
Cargo offloaded	counter for total cargo offloaded from aircraft by cargo type
Ground Transport	
Number available	run-time display of available ground transport units
Destination information	table of cargo at destinations 1-4 by type
Utilization Rate	utilization rate for motorpool resource

There are three principal objects in the simulation code. The first is the airfield object. In the airfield object resources related to the operation of the airfield are created and managed (e.g. ramp spaces, runways, ground transports, and cargo handlers). The airfield also controls the arrival of aircraft, the servicing of the aircraft, and the operation of the ground transports.

Table 4 Airfield Object Fields

Field	Function
Cargo queues	12 queues used to hold cargo at the loading dock. Organized by type and destination.
CargoHandler	Airfield cargo handler resource object
Forklifts	Airfield forklift resource object
FuelTrucks	Airfield fuel truck resource object
LoadingDock	Airfield ramp space resource object
MotorPool	Airfield ground transport resource object
Parking space queue	Queue of parking space objects. Used to keep track of ramp space status
Runway	Resource object used to deconflict taxiing aircraft and landing aircraft

Table 4 lists the fields of the airfield object. Only one airfield object is instantiated at the beginning of the simulation. Along with the airfield object come its various resources such as cargo handlers, forklifts, transport trucks, and fuel trucks. Each of these is a resource object and has a structure defined by MODSIM III allowing simple employment and built in statistic gathering.

Table 5 Airfield Object Methods

Method	Function
Decrement	Series of methods used to decrement airfield resources according to user input.
DoAircraftMx	Performs concurrent maintenance on aircraft.
DriveTruck	Graphical method called to show a truck being driven to/from a destination
FuelAircraft	Fuels aircraft by either pit or truck.
GenerateArrivals	Controls arrival of aircraft. Instantiates aircraft objects
Increment	Series of methods used to increment airfield resources according to user input.
ManageLoadingDock	Places cargo into cargo queues. Calls for transport.
NextParkingSpace	Returns the location of the nearest available ramp space.
ObjInit	Instantiates airfield object. Creates airfield resources and queues.
ProcessCargo	Second step in unloading palletized cargo. Instantiates individual cargo handler objects.
SendTruck	Called inside manageLoadingDock method. Requests transport for a load of cargo.
UnloadCargo	First step in unloading palletized cargo. Assigns cargo handlers.
UnloadPassengers	Unloads passengers from aircraft.
UnloadRollingCargo	Unloads rolling cargo from aircraft.
UpdateParkingSpace	Manages the status of a ramp space.

The second object is the aircraft object. The fields of the aircraft object contain information about the amount of cargo, fuel capacity, and maintenance record for each type of aircraft. The aircraft object also carries out the graphical mechanics of landing, taxiing, parking, and taking off.

Table 6 Aircraft Object Fields

Field	Function
Finish	Boolean variable indicating whether the aircraft object has completed the specified service.
Fuel	Real value of aircraft's fuel capacity.
FuelRate	Aircraft's max fuel take-on rate.
Icon	The five graphical icons indicating the status of aircraft servicing.
Location	The integer value of the ramp space occupied by the aircraft object.
MaxFuelers	Max number of fuel trucks hook-ups
Mx/prob/dur	Probabilities and duration for both concurrent and non-concurrent maintenance.
Pallet/rolling/passenger Cargo	Integer value of aircraft's cargo capacity.
Trigger	Fuel trigger fires when aircraft is ready to be fueled. Finished Trigger fires when aircraft is completely serviced.
WaitTime	Captures the begin and end time for each aircraft object. Used to determine aircraft time in system (histogram).

The main method of the aircraft object is the land method. Once the airfield object instantiates an aircraft object, the aircraft object is asked to carryout its land method. The land method oversees the activities of the aircraft object from touchdown to takeoff. Inside the land method, control is passed back to the airfield object for servicing of the aircraft. Once the airfield is finished servicing the aircraft, a trigger object fires allowing the aircraft object to taxi from its ramp space and prepare to takeoff.

26

Table 7 Aircraft Object Methods

Method	Function
Begin/endIcon	Updates the service icons by coloring them according to status.
CheckForCompletion	Prior to disembarking the ramp space this method is called to make sure all services are complete.
DecCargo	Decrements specified type of cargo from aircraft object.
InitIcons	Draws the five icons representing the five services performed on the aircraft. Initializes them to "red" signifying they've yet to begin.
Land	Draws the aircraft object Directs the aircraft object throughout its stay at the airfield.
RepairAircraft	Performs non-concurrent maintenance on the aircraft object.
Set/GetLocation	Assigns and retrieves integer value location of aircraft object.
SetWaitTime	Sets the begin and end wait time used in calculating the aircraft object's time in system.
TakeOff	Graphically represents aircraft takeoff. Disposes of the aircraft object.

The third object is the cargo object. The cargo object is a relatively simple object whose primary function is easing the capture of vital information on the cargo transition process. Although the cargo object is simple in structure, its presence is a departure from the manner in which BRACE modeled cargo handling. BRACE treated cargo as a simple counter, not as an object. The proof of concept model that was the origin of APOM sought to create cargo objects and pass them from aircraft, to material handling equipment, to loading dock, to a ground transport, and finally to a destination point.

27

In addition to the cargo object, the cargo module contains the cargo handler object. It functions as an individual cargo handler such as a 60k-loader.

Table 8 Cargo Module

Object	Field/Method	Function
Cargo	Destination (f)	Integer value of the cargo object's destination
	IncrementCargoOnDock (m)	Used in parallel with airfield object's manageLoadingDock to measure levels of cargo at the loading dock.
	ObjInit (m)	Instantiates cargo object.
	SetCargoDestination (m)	Sets the destination for the cargo object.
	SetCargoType (m)	Defines cargo object type.
	TypeOfCargo (f)	Integer value of the cargo object's type: (1) pallet, (2) rolling, (3) passenger.
CargoHandler	UnloadAircraft (m)	Manages individual cargo handlers in the process of moving palletized cargo from aircraft objects to the loading dock.

The graphics module is home to the code allowing user-simulation interaction. The menu bar object contains the fields and methods that constitute the framework of communication between the user and the simulation. The menu bar recognizes which GUI the user requests, pauses the simulation, and allows the transfer of information. This module also draws the simulation board and updates the various statistical displays.

Table 9 Graphics Module

Field/Method	Function
BeSelected (m)	Pauses simulation and calls the corresponding GUI when a menu item has been clicked.
Button (f)	Generic graphical object inside a dialog box. Usually either "ok" or "cancel" button.
ChangeParameters (m)	Several methods using GUI's to change simulation parameters.
DialogBox (f)	Generic GUI used as a basis for all user/simulation interaction windows.
Meter (f)	Used to display changing simulation information: number of aircraft diverts, ramp space queue, motorpool availability, cargo handler availability, forklift availability, and fuel truck availability.
SetUpSimBoard (m)	Draws the airfield, menu board, and display meters at the beginning of the simulation.
ShowDestinations (m)	Called when the user clicks on the "destination statistics" button. Draws the dialog box with each destination's cargo tallies.
ShowStatistics (m)	Called when the user clicks on the "airfield statistics" button. Draws the dialog box with the current airfield statistics.
Start/stopItem (f)	Graphical menu items used to start and stop the simulation.
Update (m)	Updates a specified display meter with a new value.

The global variable module holds vital simulation variables that must be accessible to procedures in other modules at various points in the simulation. The technique of grouping global variables in one module was taught at the MODSIM III training seminar and eliminates troublesome fractured variable structure.

Verification

Verification requires confirmation that the simulation program is functioning properly.

APOM was programmed in an object-oriented fashion over the course of several stages of increasing model fidelity. At each stage, the model was debugged and its output was measured against the intuitive expectations of the thesis committee. Careful attention was paid to the documentation and structure of the code to enable easy identification of model functions. Additionally, the animation of the simulation allowed simple verification of the modeled processes.

The aircraft arrival process of the model matched the intended function of the conceptualization phase. Arrivals were random and the divert mechanism worked properly.

The service icons and the resource meters allowed simple verification of the aircraft service processes. The concept of operations for aircraft servicing was effectively translated to the model.

The handling of cargo was more difficult to verify. Several steps were taken to ensure accurate handling of this important modeling function. First, code was implemented to hard-wire the amount, type, and destination of cargo an aircraft carried. In this manner, careful accounting could be kept of the cargo as it moved from the aircraft to the loading dock and finally to its destination.

The simulation time scale was also altered to slow down the animation and aid in the verification of cargo handling. After the process was verified, the code was changed back to its full fidelity (randomness) and the process was rechecked.

As a further means to verify the model, a designed experiment was run to determine the factors that effect airfield capacity. Seven factors were chosen for the designed experiment influencing the output measure, airfield capacity. A 2^{7-2} fractional factorial design was employed in the 32-run study. Table 10 lists the design settings for the seven factors. Additionally, each run was terminated at 60.0 hours, the airflow was comprised of 25% C-5s and 75% C-17s, and aircraft were diverted if there were 15 aircraft on the ground.

Table 10 Design Settings

Factors	Hign	Low
AC interarrival time	10 minutes	20 minutes
Ramp Space	8	4
Cargo Handlers	20	10
Forklifts	10	5
Fuel Pits	4	1
Fuel Trucks	8	4
Transports	4	2

The results of the 32 runs were input to SAS JMP for analysis. A factor screening process revealed which main and factorial effects were significant.

Table 11 Airfield Capacity Effects Screening

Entered	Parameter	Estimate	DF	SS	F Ratio	Prob>F
X	**Intercept**	**76.0375**	**1**	**0**	**0.000**	**1.0000**
X	**Interarrival**	**7.3625**	**4**	**3084.54**	**8.237**	**0.0003**
X	**Ramp Space**	**8.1125**	**2**	**3123.01**	**16.679**	**0.0000**
X	**Interarrival*Ramp Space**	**5.6375**	**1**	**1017.005**	**10.863**	**0.0033**
X	**Cargo Handlers**	**3.0375**	**2**	**438.05**	**2.339**	**0.1199**
X	**Interarrival*Cargo Handlers**	**2.1125**	**1**	**142.805**	**1.525**	**0.2298**
_	Ramp Space*Cargo Handlers	?	1	68.445	0.722	0.4051
X	**Fuel Pits**	**5.2125**	**3**	**1847.615**	**6.578**	**0.0024**
X	**Interarrival*Fuel Pits**	**2.4375**	**1**	**190.125**	**2.031**	**0.1682**
_	Ramp Space*Fuel Pits	?	1	45.125	0.470	0.5003
_	Cargo Handlers*Fuel Pits	?	1	51.005	0.533	0.4733
X	**Fuel Trucks**	**4.0125**	**2**	**1303.25**	**6.960**	**0.0045**
_	Interarrival*Fuel Trucks	?	1	85.805	0.913	0.3502
_	Ramp Space*Fuel Trucks	?	1	34.445	0.357	0.5565
_	Cargo Handlers*Fuel Trucks	?	1	117.045	1.265	0.2734
X	**Fuel Pits*Fuel Trucks**	**-4.9625**	**1**	**788.045**	**8.417**	**0.0083**
_	Trucks	?	1	59.405	0.624	0.4385
_	Interarrival*Trucks	?	2	97.25	0.496	0.6165
_	Ramp Space*Trucks	?	2	185.81	0.992	0.3885
_	Cargo Handlers*Trucks	?	2	110.41	0.566	0.5764
_	Fuel Pits*Trucks	?	2	108.41	0.556	0.5823
_	Fuel Trucks*Trucks	?	2	179.53	0.955	0.4017

The results of the effects screening show **Ramp Space** as the term with the greatest effect. The interaction term **Interarrival*Ramp Space** had the next greatest effect. Logically, if an airfield had more ramp space and a greater aircraft arrival rate, the expected airfield capacity would be greater.

Of the remaining effects, **Fuel Pits**, **Fuel Trucks**, and their interaction term had the most significant impact on airfield capacity. Aircraft fueling was modeled as an independent process from other airfield activities. If fueling resources were increased, the airfield capacity was expected to increase. The negatively-valued interaction term indicates that low values of both fuel pits and fuel trucks would significantly reduce airfield capacity.

Cargo Handlers, Trucks, and Forklifts did not play a significant role in determining airfield capacity. Indeed, APOM modeled trucks as the means to move cargo from the aerial port to their final destinations. In no way could trucks impact airfield capacity. If, however, a limit were placed on the cargo capacity of the loading dock, then the inability to move cargo away from the aerial port would result in aircraft diverts. Forklifts were underutilized in every scenario of the experiment, but their only role is assisting the movement of palletized cargo from cargo handlers, to the loading dock, and onto transport trucks.

The results of the effect screening were not surprising. They followed the expectations of the thesis committee and serve to verify the function of APOM.

Validation

Validation of the model ensures that the model accurately represents the real system. The thesis study validates the APOM model by using similar simulation inputs to compare output with ACE.

An attempt was made to match inputs between ACE and APOM. Discrepancies occurred in the modeling of ground servicing of aircraft by ACE. ACE models several services beyond cargo transfer, fueling, and maintenance. The inclusion of these services increased the ground time per aircraft, and, as a result, the estimate of airfield capacity was lowered.

APOM's aircraft interarrival rate was continuously lowered until the airfield resources were saturated. Saturation was achieved when ramp space utilization went above 90% and aircraft diverts became frequent.

ACE was used in its expected value mode, and, with the reported inputs, estimated the airfield capacity at 67 aircraft. The limiting resource was aircraft servicing. APOM was run for 60 hours. After removing the first 10 hours due to transient effects and employing a batch mean technique to the remaining 50 hours, the estimated airfield capacity was approximately 69 aircraft. The similarity in capacity measures suggests the mobility activities of arrival, offload, fueling, and maintenance were modeled correctly.

Table 12 APOM Input

Area	Variable	Value
Aircraft (C-17)	Mean Interarrival Rate	15 min
	Pallet Cargo	10
	Rolling Cargo	2
	Passengers	40
	Fuel	150,000 lbs
	Con. MX (Prob, Duration)	0.075, 60 min
	Non-con MX (Prob, Duration)	0.025, 60 min
Airfield	Cargo Handlers (num, cap)	15, 2 pallets per
	Forklifts	10
	Fuel Pits (num, rate)	1, 750 gal/min
	Fuel Truck (num, cap, rate)	10, 5000 gal, 500 gal/min
	Motorpool (num, cap pal/rol/pas)	3, 10, 2, 50
	Ramp space	6

Table 13 ACE Input

Area	Variable	Value
Aircraft (C-17)	Pallet Cargo	9
	Passengers	40
	MX (prob, duration)	0.1, 60
	Fuel	150,000 lbs
Airfield	Cargo Handlers (type, num)	40k loader, 15
	Fuel Pits (num, rate)	1, 750 gal/min
	Fuel Trucks (type, num, rate)	R-9, 10, 550
	Bus (num, cap)	6, 40 passengers
Mission	Quick-turn, Offload only	

IV. Findings and Analysis

An experiment similar to the airfield capacity effects screening was run to better understand aircraft diverts. The experimental design mirrored the earlier experiment with the exception that aircraft diverts were measured instead of airfield capacity.

Table 14 Divert Effects Screening

Entered	Parameter	Estimate	DF	SS	F Ratio	Prob>F
X	Intercept	78.28125	1	0	0.000	1.0000
X	Interarrival	71.21875	4	172004.6	71.995	0.0000
X	Ramp Space	-19.28125	2	19186.81	16.062	0.0001
X	Interarrival*Ramp Space	-15.09375	1	7290.281	12.206	0.0021
X	Cargo Handlers	-7.59375	2	2780.563	2.328	0.1211
X	Interarrival*Cargo Handlers	-5.40625	1	935.2813	1.566	0.2239
_	Ramp Space*Cargo Handlers	?	1	399.0313	0.658	0.4265
X	Fuel Pits	-11.84375	3	10123.59	5.650	0.0050
X	Interarrival*Fuel Pits	-6.78125	1	1471.531	2.464	0.1308
_	Ramp Space*Fuel Pits	?	1	318.7813	0.522	0.4779
_	Cargo Handlers*Fuel Pits	?	1	357.7813	0.588	0.4518
X	Fuel Trucks	-9.34375	2	6957.063	5.824	0.0093
_	Interarrival*Fuel Trucks	?	1	731.5313	1.238	0.2784
_	Ramp Space*Fuel Trucks	?	1	148.7813	0.240	0.6289
_	Cargo Handlers*Fuel Trucks	?	1	675.2813	1.138	0.2983
X	Fuel Pits*Fuel Trucks	11.40625	1	4163.281	6.970	0.0150
_	Trucks	?	1	385.0313	0.634	0.4348
_	Interarrival*Trucks	?	2	632.5625	0.506	0.6106
_	Ramp Space*Trucks	?	2	1195.063	1.000	0.3854
_	Cargo Handlers*Trucks	?	2	857.8125	0.698	0.5091
_	Fuel Pits*Trucks	?	2	667.0625	0.535	0.5939
_	Fuel Trucks*Trucks	?	2	1060.313	0.878	0.4311

Results of the experiment showed the main effect, aircraft **Interarrival,** had the most significant effect. The magnitude of its significance was much greater than in the airfield capacity experiment (F-ratio 71.995 compared to 8.237) which suggests a much stronger link between aircraft arrival rate and aircraft diverts than aircraft arrival rate and airfield capacity. The remaining effects for aircraft diverts followed the same trend airfield capacity effects followed. **Fuel Pits, Fuel Trucks,** and their interaction term were all significant factors.

The effects screening for service MOG was drawn from the same experimental setup as the effects screening for aircraft diverts. The results showed **Interarrival, Ramp Space,** and their interaction term to be the most significant factors. These factors impact the number of aircraft parked at the airfield. When looking at service resource factors, it is interesting to note that **Fuel Pits, Fuel Trucks,** and their interaction terms with **Ramp Space** all show a negative influence on service MOG. If one or more of these factors were at a low (negative) setting, the service MOG would be positively influenced. Aircraft servicing would take longer, raising the time-averaged number of aircraft in service.

Table 15 Service MOG Effects Screening

Entered	Parameter	Estimate	DF	SS	F Ratio	Prob>F
X	**Intercept**	**4.478125**	**1**	**0**	**0.000**	**1.0000**
X	**Interarrival**	**0.471875**	**3**	**13.33094**	**19.258**	**0.0000**
X	**Ramp Space**	**1.009375**	**4**	**43.09125**	**46.687**	**0.0000**
X	**Interarrival*Ramp Space**	**0.428125**	**1**	**5.865312**	**25.419**	**0.0000**
_	Cargo Handlers	?	1	0.137812	0.587	0.4519
_	Interarrival*Cargo Handlers	?	2	0.438125	0.945	0.4047
_	Ramp Space*Cargo Handlers	?	2	0.250625	0.520	0.6017
X	**Fuel Pits**	**-0.284375**	**2**	**4.845625**	**10.500**	**0.0006**
_	Interarrival*Fuel Pits	?	1	0.227812	0.987	0.3313
X	**Ramp Space*Fuel Pits**	**-0.265625**	**1**	**2.257812**	**9.785**	**0.0047**
_	Cargo Handlers*Fuel Pits	?	2	0.163125	0.333	0.7205
X	**Fuel Trucks**	**-0.290625**	**3**	**5.408438**	**7.813**	**0.0009**
X	**Interarrival*Fuel Trucks**	**0.103125**	**1**	**0.340312**	**1.475**	**0.2369**
X	**Ramp Space*Fuel Trucks**	**-0.271875**	**1**	**2.365313**	**10.251**	**0.0040**
_	Cargo Handlers*Fuel Trucks	?	2	0.303125	0.636	0.5393
_	Fuel Pits*Fuel Trucks	?	1	0.165313	0.707	0.4094
_	Trucks	?	1	0.112812	0.478	0.4967
_	Interarrival*Trucks	?	2	0.308125	0.647	0.5336
_	Ramp Space*Trucks	?	2	0.203125	0.418	0.6638
_	Cargo Handlers*Trucks	?	3	0.550937	0.772	0.5231
_	Fuel Pits*Trucks	?	2	0.150625	0.307	0.7391
_	Fuel Trucks*Trucks	?	2	0.375625	0.800	0.4627

Verification and validation of APOM revealed no discrepancies with the model or its results. Indeed, the verification of APOM provided a means to test its usefulness in understanding the interplay of airfield resources and measures in a structured experiment. The measures of airfield capacity and service MOG provided by APOM were useful in the analysis of airfield operations.

V. Conclusions

In the search for a more useful tool in mobility planning and analysis, APOM offers the user several advantages over the ACE and BRACE models currently in the AMC inventory. The careful development of APOM enabled the author to avoid the problems associated with ACE and BRACE while including beneficial features of both.

APOM provides the user with an accurate, animated model of an air mobility base's operation and includes the added feature of a ground transportation network leading from the aerial port to four destinations. APOM also attempts to define airfield capacity with two displays. The first estimates capacity by calculating the rate of aircraft departure and extrapolating to a 24 hour time period. The airfield capacity display gives the user a run-time estimate of airfield capacity subject to user-defined parameters. The second display shows the number of aircraft in service on the ground at any given moment. By time-averaging this plot, the user can determine the average number of aircraft in service on the ground. With a service MOG less than the number of ramp space, the user can conclude resources are limiting the airfield capacity. With a service MOG near the number of ramp space, the user can conclude ramp space is limiting the airfield capacity. Both measures of capacity are unique to APOM.

With future tuning and added capabilities, APOM's utility in the field of mobility analysis will increase. Areas of future interest include:

- Greater fidelity in modeling the passenger and rolling cargo offload processes
- Capability to input an aircraft arrival pattern from the MASS model
- Expanding the transportation network to reflect a real-world network
- Using the model to test the feasibility of a planned deployment

Appendix A: Aerial Port Operations Model User's Guide

The APOM user's guide is divided into five areas: model background, graphical layout, animation explanation, parameter input, and data output.

Model Background

APOM was written in the MODSIM III simulation language developed by CACI Inc. The simulation code was written using MODSIM III's PC/Windows environment and, after compilation, yielded the executable file that runs the simulation. Requirements needed to execute the simulation are limited to 5 files: the .exe executable file, the .txt input file, the .sg2 graphics file, the .dll dynamic linked library file, and the .bmp bitmap file. All five files must be contained in the same folder. It is not necessary for the user to own a MODSIM III license in order to run the simulation. Any future adaptations to the model, however, would require recompilation and, hence, a MODSIM III license.

Graphical Layout

The features of the airfield layout include: two runways (one takeoff and one landing), one ramp for aircraft parking, an aerial port facility, and a simple ground transportation network with four end nodes.

There are several simulation meters displaying pertinent simulation data at run time. These meters are grouped into three areas of the simulation board: the bottom display, the aerial port display, and the chart area.

The bottom display contains the simulation clock (showing hours and minutes) and a divert counter (showing number of aircraft diverts).
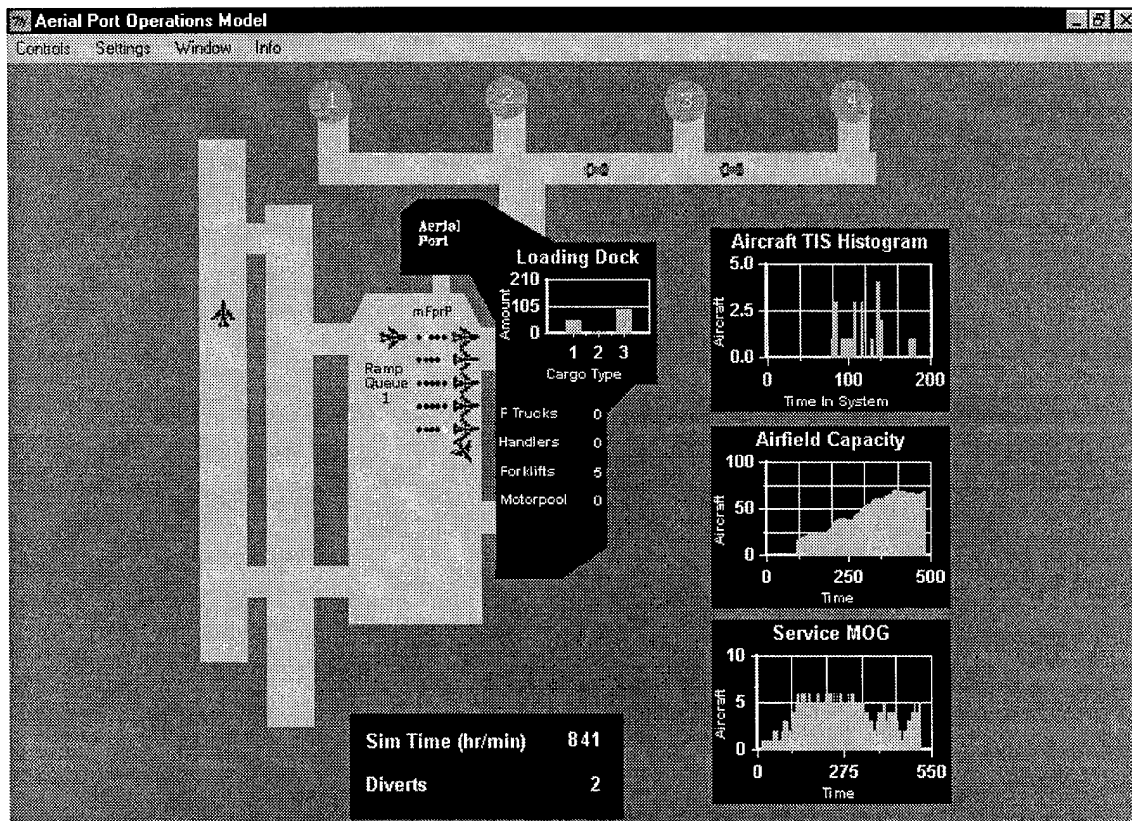
Figure 4 APOM Graphical Layout

The aerial port contains four counters showing the availability of the following airfield resources: fuel trucks, cargo handlers, forklifts, and transport trucks. Additionally, a ramp space queue meter is located on the ramp. The aerial port area also contains a dynamic, loading dock operations chart showing

the current levels of cargo on the dock organized by cargo type (1, palletized cargo; 2, rolling cargo; and 3, passengers).

The chart area contains three dynamic displays. The first is an aircraft time-in-system histogram. The second is a real-time estimate of airfield capacity measured in aircraft per 24 hours. The third display is a measure of the number of aircraft in service at a given time. This is called "service MOG (maximum on the ground)."

The menu bar at the top of the simulation window provides the user with flexible control of the simulation. There are four choices at the menu bar; the *control* button, the *settings* button, the *window* button, and the *info* button.

The *control* button gives the user start/stop options. Start simulation will begin the flow of aircraft into the airfield according to the default parameters or those specified by the user. Stop simulation halts the simulation and provides the user with the summary statistics before exiting the simulation.

The *settings* button allows the user to change parameters in four areas: airfield, aircraft, cargo, and transportation network. These options will be dealt with in detail later in the user's guide.

The *window* button allows the user to select either airfield statistics or destination statistics. The statistics kept by the simulation will be detailed later in the user's guide.

The *info* button calls a brief message regarding the origin of the simulation to the screen.

Animation

Once the user is satisfied with the initial conditions of the airfield and selects the *start* button APOM initiates the incoming flow of aircraft to the airfield. Aircraft appear as they approach and land on the runway. When an aircraft reaches the end of the runway it turns and taxis to a preset location to await assignment of ramp space. If no space is available, the aircraft remains at the preset location and the queue meter increments to indicate the number of aircraft awaiting parking. If ramp space is available, the aircraft taxis to the nearest available space to await servicing.

Three processes are performed on aircraft while parked on the ramp. The first process, unloading cargo, can be subdivided into three categories; palletized cargo, rolling cargo, and passengers. The second process is concurrent maintenance (maintenance performed simultaneously with cargo offloading). The third process is aircraft fueling.

Figure 5 Ramp Detail

Five icons appear behind each aircraft upon arriving at a ramp space to inform the user of the status of each process. The character string "mFprP" serves to remind the user which service corresponds to which icon. "m" is concurrent maintenance. "F" is aircraft
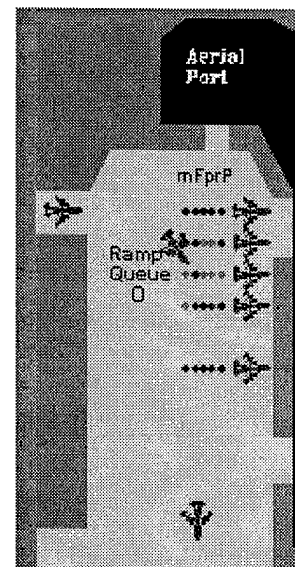
44

fueling. "p" is palletized cargo. "r" is rolling cargo. "P" is passengers. A red icon indicates the process has not yet begun. A yellow icon indicates the process has begun but has not been completed. A green icon indicates the process has been completed.

A second type of maintenance has been incorporated into APOM. Non-concurrent maintenance occurs independently of any other aircraft service. Should an aircraft require non-concurrent maintenance after completing its normal servicing it taxis to an unoccupied area of the ramp and completes its maintenance there.

As an aircraft is serviced several displays will update the availability of resources and levels of cargo at the aerial port. Fuel trucks are used to fuel aircraft parked at ramp spaces without an available fuel pit. Cargo handlers are used to offload palletized cargo. Forklifts are used to transfer cargo from cargo handlers to the loading dock and then to transport trucks. The motorpool is used to transfer cargo to their final destination. Additionally, upon departure from the airfield each aircraft will trigger an update of the two dynamic charts; the aircraft time-in-system histogram (reflecting each aircraft's time on the ground) and the airfield capacity estimator (reflecting an estimate of the airfield's 24-hour capacity).

As cargo accrues on the loading dock, the motorpool of trucks is called into action to transport the cargo to their final destination. A truck carrying

palletized cargo will be colored brown and green. A truck carrying rolling cargo will be colored black. A truck carrying passengers will be colored orange.

Parameter Input

There are four graphical user interfaces (GUI) through which a user can select APOM settings. The GUIs can be accessed by clicking the *settings* button on the main menu. Additionally, the user can reset the default settings by changing the parameter values in the *input.txt* file.

The first GUI allows the user to set airfield and simulation parameters. Stop time allows the user to set the length of the simulation. Time scale allows the user to set the animation speed. The user can specify the mean of the interarrival distribution as well as the maximum allowed number of aircraft on the ground before diverting incoming aircraft. Further, the user can specify the levels of six airfield resources. Any of these inputs may be changed mid-simulation to allow the user real-time

Figure 6 Airfield Parameters Input

impact analysis. The checkbox labeled "reset resource statistics" allows the user to control whether or not to reset the utilization statistics related to the airfield resources. Checking the box will reset the statistics.

46

The second GUI allows the user the set cargo and fuel parameters. By changing the distribution of cargo the user can alter network flow. Other cargo parameters include the delay incurred when offloading a single pallet of cargo and the number of cargo pallets a single cargo handler can carry. By changing the pallets per cargo handler parameter the user can define the "type" of cargo handler being employed at the airfield
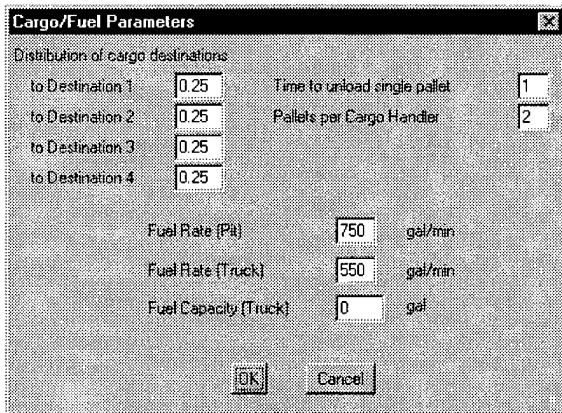
Figure 7 Cargo and Fuel Input

(25k, 40k, or 60k loaders). The three fuel parameters allow the user to define fuel transfer rates for fuel pits and fuel trucks as well as the capacity of fuel trucks.

The third GUI deals with the transportation of cargo from the aerial port across the simple network leading to the final destination. The user can set the distances to the final destinations to reflect realistic transportation network delays.
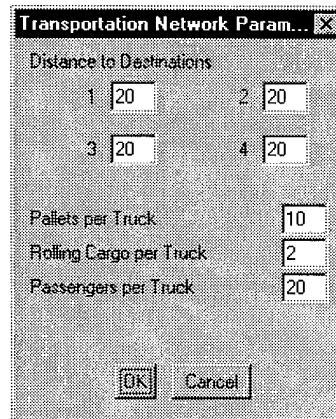
Figure 8 Transportation Input

Through this GUI the user defines the amount of cargo which can be carried by a single transport truck. Here, it assumed that the motorpool is comprised of utility trucks that can be used to transport any type of cargo.

The final GUI allows the user to see aircraft parameters according to aircraft type. At the time of thesis publication, aircraft parameters were hard-coded into the simulation and not subject to change by the user. Future improvement to the model may add the capability for user-manipulated aircraft data.
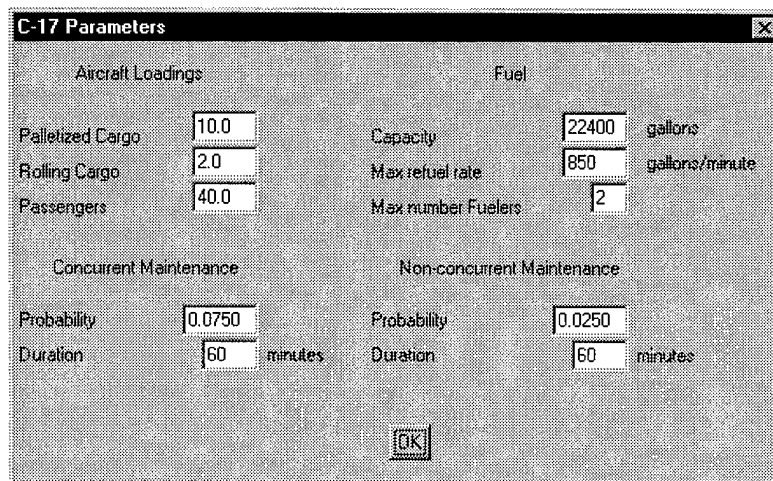
Figure 9 Aircraft Parameters

Data Output

Airfield and destination statistics can be queried at any time during the simulation run by clicking the *windows* button on the main menu. The airfield dialog box shows current levels of various statistics including the number of aircraft landed, departed, diverted, and encountered non-concurrent maintenance. Also contained in the dialog box are the utilization rates for five airfield resources.

The destination statistic dialog box presents the current amount of cargo

at each destination organized by cargo type. This information can be used to
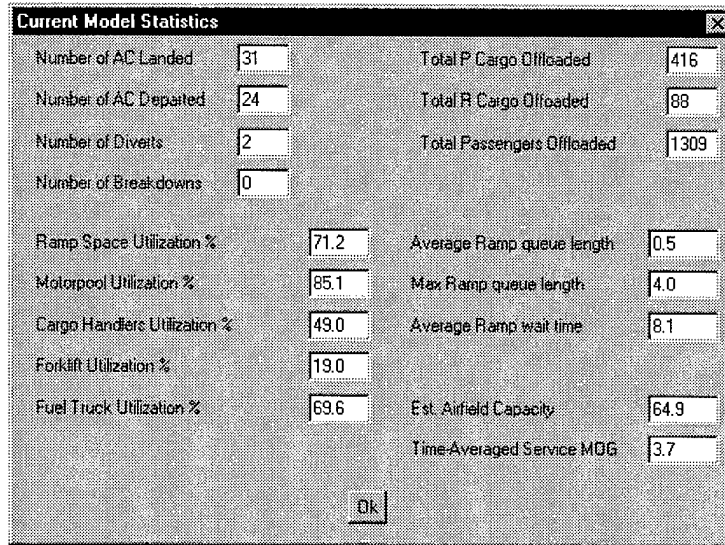
Figure 10 Model Statistics

determine a cutoff point to simulate the completion of a deployment of a specific

amount of cargo to a specific destination.

APOM creates an output file, *output.txt*, containing airfield capacity data.

A row of data is generated when an aircraft

departs the airfield. The first column is the

estimated airfield capacity (aircraft/day)

and the second column is the simulation

time of the aircraft departure (minutes).
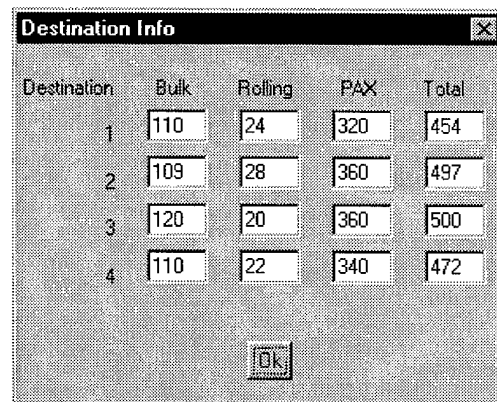
Figure 11 Destination Statistics

49

Appendix B: APOM Structure and Functional Organization

Structure

The simulation was written using MODSIM III for PC's by CACI Inc. The code is broken up into six modules. The three primary modules describe the airfield, the aircraft, and the cargo. Two functional modules handle variables used throughout the simulation and the graphics employed by the simulation. The final module, the main module, initializes the simulation.

There are three principal objects in the simulation code. The first is the *airfield* object. In the airfield object resources related to the operation of the airfield are created and managed (e.g. parking spaces, runways, ground transports, and cargo handlers). The *airfield* object also controls the arrival of aircraft, the servicing of aircraft, and the operation of the ground transports.

The second object is the *aircraft* object. The *aircraft* object determines the amount of cargo carried in each aircraft. The *aircraft* object also carries out the graphical mechanics of landing and taking off as well as acquiring a parking space.

The third object is the *cargo* object. The *cargo* object is a relatively simple object whose primary function is easing the capture of vital information on the cargo transition process.

The graphics module is home to the code allowing user-simulation interaction. This module also provides the link between simulation statistics and their graphical presentation.

The global variable module holds vital simulation variables that must be accessible to procedures in other modules at various points in the simulation. This centralized source of global variables eliminates troublesome fractured variable structure.

Simulation Run-Through

Setup

The initial phase of the simulation is the graphical setup of the model. Once the model executable file has been called the simulation instantiates a *main menu* object and asks it to *setUpSimBoard*. This function, located in the graphics module, loads and draws the graphical objects that make up the simulation board. Included in this process are: menu items, the airfield image, and various meters, charts and graphs.

At this time the simulation also instantiates an *airfield* object. The initialization of the airfield object creates various resource objects (ramp space, cargo handlers, forklifts, fuel trucks etc.) as well as the infrastructure for handling cargo at the aerial port.

After the setup is completed the simulation awaits the user go-ahead to begin aircraft arrivals. During this waiting period the user may make changes to the simulation parameters. The process of changing parameters is accomplished through the graphics module. Clicking on a menu item will open a dialog box. The simulation will present the user with the current parameter values. The user may then change any of the values. The simulation will recognize any change in parameters and save the changes in the associated global variables before closing the dialog box.

Aircraft Arrivals

The simulation will tell the *airfield* object to *generateArrivals* when the start simulation button is clicked. This method generates *aircraft* objects at random intervals based on a user-defined mean interarrival time and continues to be called throughout the simulation until the current simulation time exceeds the user-defined simulation run length. This method will also determine if a divert occurs by checking the number of aircraft currently on the ground and comparing it to the user-defined maximum allowed number of aircraft on the ground.

After the *airfield* method *generateArrivals* instantiates an *aircraft* object, it tells the *aircraft* object to *land*. The aircraft object has been initialized with the characteristics of a user-defined aircraft type (amounts of cargo, fuel capacity, fueling rate, and maintenance probabilities) and prepares to land at the airfield.

The *aircraft* object waits until the runway is free of obstructions, and then renders itself on the simulation board and rolls-out to the end of the runway.

The *aircraft* object taxis to a preset location and enters the ramp queue. If all ramp spaces are in use, the ramp queue display is incremented. Once a ramp space is available, the *aircraft* object is assigned an instance of that resource and taxis to its assigned location.

MODSIM III does not allow the programmer to determine which particular instance of a resource is assigned, only that one has been assigned. Normally this shortfall is of no concern, but in APOM's case a graphical sense of which instance of a resource becomes important. The *aircraft* object needs to know where the available ramp space is.

To circumvent this problem a parallel construct was implemented. In addition to the ramp space resource, a queue of *parkingSpace* objects is maintained by the *airfield* object. The list of *parkingSpace* objects contains the same number of objects as there are ramp spaces. Each *parkingSpace* object contains a field that describes its location and status. The *airfield* object method *nextParkingSpace* searches the queue of *parkingSpace* objects for the first available parking space.

Aircraft Servicing

The *aircraft* object is now sitting at its assigned ramp space and control of the simulation is passed back to the *airfield* object to service the aircraft. Graphical icons are drawn near the *aircraft* object to indicate the status of these services. The *airfield* object will perform three services on the *aircraft* object; cargo offload, aircraft maintenance, and aircraft fueling. Of these services, cargo offload and aircraft maintenance may take place simultaneously. Aircraft fueling begins when all other services are complete.

Cargo Offloading

The offloading of cargo is separated into three functions; *unloadPassengers*, *unloadCargo*, and *unloadRollingCargo*. All three functions may occur simultaneously and are carried out by the *airfield* object on the *aircraft* object.

The *unloadPassengers* method is relatively straightforward. A simple delay is incurred each time a passenger is removed from the aircraft. The number of passengers on an *aircraft* object is contained in the *passengerCargo* field. Each pass through a simple WHILE loop accomplishes three tasks. First, a *cargo* object is instantiated. Next, the *cargo* object is declared to be of passenger type and is given a final destination. And finally, the *cargo* object is passed to the loading dock via the method *manageLoadingDock*, where it is placed in a cargo queue awaiting transport to its final destination. When the aircraft is emptied of its

passengers, its icon color is changed and a status check is run to determine if all servicing is complete.

The offloading of rolling cargo mirrors that of passengers. A simple delay is incurred each time rolling cargo is removed from the aircraft. The number of rolling cargo aboard an *aircraft* object is contained in the *rollingCargo* field of the *aircraft* object. The same tasks are accomplished in the WHILE loop for rolling cargo as were for passengers. And again, when the aircraft is emptied of its rolling cargo, its icon color is changed and a status check is run to determine if all servicing is complete.

The offloading of pallet cargo is a much more complicated process involving two airfield resources; cargo handlers and forklifts. The first step is the assignment of cargo handler resources to the *aircraft* object. The code for this action is contained in the *unloadCargo* method. This method will determine how many cargo handlers are needed to offload the palletized cargo from the aircraft and compare that number of cargo handlers to the number available at that time. Three possible outcomes of the comparison are accounted for in the code. The first case occurs when the number available is greater than or equal to the number required. In this case, the *aircraft* object is assigned the required number of cargo handlers. The second possible outcome occurs when the number available is less than the number required. In this case, the *aircraft* object is assigned the available cargo handlers. The third possibility is no available cargo

handlers. In this case, the *aircraft* object waits for cargo handlers to be released by the *aircraft* object occupying the first place on the allocation list of the cargo handler resource (i.e. the *aircraft* object that has held cargo handlers the longest).

The second step in the process is to remove palletized cargo from the *aircraft* object. This activity is covered by two methods: *processCargo* (airfield object) and *unloadAircraft* (*cargoHandler* object). *ProcessCargo* method creates a team of *cargoHandler* objects based on the number of cargo handlers assigned in the *unloadCargo* method. Each *cargoHandler* object is instructed to *unloadAircraft* until the *aircraft* object is emptied of its palletized cargo. The *unloadAircraft* method allows accurate modeling of the cargo offload procedure including details such as travel time to the aircraft, queuing the cargo handlers at the aircraft to allow offloading by a single handler at a time, and travel time to the loading dock. The *unloadAircraft* method also accomplishes the instantiation of *cargo* objects and defines their type and destination.

The final step is transferring the cargo objects from the *cargoHandler* objects to the loading dock. This task is accomplished by the *unloadAircraft* method and involves the airfield resource, forklifts. When the last pallet of cargo is removed from the aircraft, its service icon color is changed and a status check is run to determine if all servicing is complete.

Maintenance

Two types of maintenance are modeled by APOM; concurrent and non-concurrent maintenance. Concurrent maintenance can be accomplished at the same time as cargo offloading. Non-concurrent maintenance is performed on the aircraft after all other servicing is complete. The method *doAircraftMX* runs the concurrent maintenance on *aircraft* objects. A simple random draw is compared to the probability of concurrent maintenance field of the *aircraft* object to determine if a maintenance problem exists. If a problem exists, a simple delay of random duration based on the duration of concurrent maintenance field of the *aircraft* object is incurred. After completion, the process icon is changed to green and a status check is run to determine if all processes have been completed.

A simple random number draw determines whether non-concurrent maintenance is needed after the airfield has completed servicing the aircraft. If a non-concurrent maintenance problem exists, the aircraft taxis to an unoccupied area of the ramp and incurs a delay based on a random number draw. After the maintenance is completed the aircraft re-enters the taxi pattern and departs the airfield.

Fueling

Aircraft fueling is initiated when the cargo offload and aircraft concurrent maintenance have been completed. There are two aircraft fueling methods, *fuelAircraftByPit* and *fuelAircraftByTruck*, both of which are triggered to begin

when cargo offloading and concurrent maintenance procedures finish. Which fueling method operates on an *aircraft* object depends on the location of the *aircraft* object. If the *aircraft* object occupies a ramp space serviced by a fuel pit, the *fuelAircraftByPit* method is called. If the *aircraft* object occupies a ramp space not serviced by a fuel pit, the *fuelAircraftByTruck* method is called.

Fueling an aircraft by pit requires knowledge of three variables: the fuel transfer rate of the fuel pit, the maximum fuel take-on rate of the aircraft, and the fuel capacity of the aircraft. A simple delay is incurred according to the relationship of these three variables. After fueling is completed, the aircraft performs a status check and leaves the ramp space. The ramp space resource is returned to the *airfield* object and the availability of the parking space is changed in the *parkingSpaceQueue*.

Fueling an aircraft by fuel truck requires knowledge of six variables: the fuel transfer rate of a fuel truck (both on and off), the number of fuel hook-ups on the aircraft, the maximum fuel take-on rate of the aircraft, the capacity of a fuel truck, the fuel capacity of the aircraft, and the number of fuel trucks assigned to the aircraft.

The departure of an aircraft object from the airfield triggers two graphical outputs to change. The first is the aircraft time in system histogram. A calculation is made resulting in the *RDataPt aircraftTIS*. Upon departure, the dynamic histogram is updated with the new *RDataPt*. The second graphical

output to change is the airfield capacity estimator. In the aircraft object method, takeOff, a calculation of airfield capacity is made using the number of departures, the current simulation time, and extrapolating to find the estimated number of aircraft serviced in 24 hours. This new data point is then incorporated into the dynamic graph.

Loading Dock Management

The cargo on the loading dock is managed with two methods: the *airfield* object method *manageLoadingDock* and the *cargo* object method *incrementCargoOnDock*. Both methods are called sequentially during the offload of the three types of cargo. The method *incrementCargoOnDock* keeps a count of the current levels of cargo on the loading dock and is linked to the loading dock dynamic graphical display. The method *manageLoadingDock* places *cargo* objects into various queues based on their type and destination. When levels of cargo heading to a specific destination reach a user-defined point a transport method is invoked to take the cargo to their destination.

Transportation Network

The *airfield* object governs the movement of cargo over the transportation network by employing the *sendTruck* method. When the amount of any type of cargo headed to a specific destination reaches a user-defined level (the truck capacity of that type of cargo) the *sendTruck* method is called. The *sendTruck* method encompasses all activities from loading of cargo onboard the truck, to

graphically driving the truck to its destination, to unloading the cargo at the final destination, to returning the truck to the airfield motorpool.

The first step is acquiring a truck from the *airfield* object's motorpool resource. Next, cargo objects are placed into a queue representing the hold of the truck. The animation of the truck is carried out in the *driveTruckTo* (and *driveTruckFrom*) method. Finally, the cargo is offloaded from the trucks, the destination statistics are updated, and the truck is driven back to the airfield for further service.

Appendix C: APOM Code

Aircraft Modules (Definition and Implementation)

```
DEFINITION MODULE aircraftMod;

FROM globalMod        IMPORT  airfield,
                              library,
                              window,
                              mainMenu,
                              chart1,
                              chart2,
                              chart4,

                              numberOfFuelPits,

                              maxWaitingTime,
                              totalArrivals,
                              totalDepartures,
                              totalBreakdowns,
                              totalWaitingTime,
                              totalWaits,

                              mog,
                              service,
                              timeAvg,
                              outputStream,

                              currentOnGround,
                              numInQueue,

                              stream2,
                              stream3;

FROM SimMod           IMPORT  SimTime, TriggerObj;
FROM Animate          IMPORT  DynImageObj;
FROM Image            IMPORT  ImageObj;
FROM Graph            IMPORT  RDataPt, RDataPtMObj, IDataPt, IDataPtMObj;
FROM GTypes           IMPORT ALL ColorType;
FROM MathMod          IMPORT  ATAN2;

TYPE
    aircraftObj = OBJECT (DynImageObj)

        beginWaitTime   :   REAL;
        endWaitTime     :   REAL;
        aircraftTIS     :   RDataPt;
        palletCargo     :   INTEGER;
        rollingCargo    :   INTEGER;
        passengerCargo  :   INTEGER;
        fuel            :   REAL;
        fuelRate        :   REAL;
        maxFuelers      :   INTEGER;
```

61

```
        concurMXprob    :   REAL;
        nonConMXprob    :   REAL;
        concurMXdur     :   REAL;
        nonConMXdur     :   REAL;

        location,
        oldState,
        newState        :       INTEGER;
        fuelBegin,
        palletBegin,
        rollingBegin,
        passengerBegin,
        mxBegin,
        fuelFinish,
        palletFinish,
        rollingFinish,
        passengerFinish,
        mxFinish        :       BOOLEAN;
        fuelIcon,
        palletIcon,
        rollingIcon,
        passengerIcon,
        mxIcon          :       ImageObj;
        finishedTrigger,
        fuelTrigger     :   TriggerObj;

        ASK METHOD decPalletCargo;
        ASK METHOD decrementFuel (IN amount : REAL);
        ASK METHOD decPassengerCargo;
        ASK METHOD getPalletCargo : INTEGER;
        ASK METHOD decRollingCargo;
        ASK METHOD setLocation (IN number : INTEGER);
        ASK METHOD getLocation : REAL;
        ASK METHOD setBeginWaitTime;
        ASK METHOD setEndWaitTime;
        ASK METHOD initIcons;
        ASK METHOD beginIcon (IN which : INTEGER);
        ASK METHOD endIcon (IN which : INTEGER);
        ASK METHOD checkForCompletion;
        TELL METHOD land;
        TELL METHOD takeOff;
        TELL METHOD repairAircraft;
    OVERRIDE
        ASK METHOD ObjInit;

    END OBJECT;

END MODULE.
```

```
IMPLEMENTATION MODULE aircraftMod;

    OBJECT aircraftObj;

        ASK METHOD ObjInit;
        VAR
            temp    :   INTEGER;

        BEGIN

            INHERITED ObjInit;

            temp := stream2.UniformInt(1,100);

            CASE temp
                WHEN 1..75:
                        palletCargo := 10;
                        rollingCargo := 2;
                        passengerCargo := 40;
                        fuel := 22400.0;{gallons}
                        fuelRate := 850.0; {gal/min}
                        maxFuelers := 2;
                        concurMXprob := 0.075;
                        nonConMXprob := 0.025;
                        concurMXdur := 60.0;
                        nonConMXdur := 60.0;
                WHEN 76..100:
                        palletCargo := 30;
                        rollingCargo := 6;
                        passengerCargo := 60;
                        fuel := 49500.0;
                        fuelRate := 850.0;
                        maxFuelers := 2;
                        concurMXprob := 0.3;
                        nonConMXprob := 0.1;
                        concurMXdur := 100.0;
                        nonConMXdur := 100.0;
            END CASE;

            NEW (finishedTrigger);
            NEW (fuelTrigger);

        END METHOD;

        ASK METHOD decPalletCargo;
        BEGIN
            palletCargo := palletCargo - 1;
        END METHOD;

        ASK METHOD decrementFuel (IN amount : REAL);
        BEGIN
            fuel := fuel - amount;
        END METHOD;

        ASK METHOD decPassengerCargo;
        BEGIN
```

```
            passengerCargo := passengerCargo - 1;
      END METHOD;


      ASK METHOD decRollingCargo;
      BEGIN
            rollingCargo := rollingCargo - 1;
      END METHOD;


      ASK METHOD getPalletCargo : INTEGER;

      VAR
            amount : INTEGER;
      BEGIN
            amount := palletCargo;
            RETURN amount;
      END METHOD;


      TELL METHOD land;

      BEGIN

            WAIT FOR airfield.runway TO Give (SELF,1);

                  INC (totalArrivals);
                  INC (currentOnGround);
                  LoadFromLibrary (library, "plane");
                  ASK window TO AddGraphic (SELF);
                  DisplayAt(1.32,0.33);

                  SetSpeed (400.0/60.0);
                  WAIT FOR SELF TO MoveTo (1.25,3.36);
                  END WAIT;
                  SetSpeed (150.0/60.0);
                  WAIT FOR SELF TO MoveTo (1.25,6.0);
                  END WAIT;
                  SetSpeed (30.0/60.0);
                  WAIT FOR SELF TO MoveTo (1.25,6.33);
                  END WAIT;
                  SetRotationSpeed (-1.3);
                  WAIT FOR SELF TO RotateTo (-1.57);
                  END WAIT;

                  ASK airfield.runway TO TakeBack (SELF,1);
            END WAIT;

            SetSpeed (15.0/60.0);
            WAIT FOR SELF TO MoveTo (2.55,6.33);
            END WAIT;

            INC (numInQueue);
            ASK mainMenu TO updateQueue;
            setBeginWaitTime;

            WAIT FOR airfield.loadingDock TO Give (SELF,1);
                  setEndWaitTime;
                  totalWaitingTime := totalWaitingTime + endWaitTime -
      beginWaitTime;
```

```
                INC (totalWaits);

                DEC (numInQueue);
                ASK mainMenu TO updateQueue;

                setLocation(airfield.nextParkingSpace);

                IF location > 1
                    WAIT FOR SELF TO RotateTo (-1.57-
(ATAN2((getLocation-1.0)*0.3,0.95)));
                    END WAIT;
                END IF;

                WAIT FOR SELF TO MoveTo (3.5, 6.63 -
(getLocation*0.3));
                    END WAIT;

                IF location > 1
                    ASK SELF TO SetRotationSpeed (1.3);
                    WAIT FOR SELF TO RotateTo (-1.57);
                    END WAIT;
                END IF;

                initIcons;

                TELL airfield TO unloadPassengers (SELF);

                TELL airfield TO unloadCargo (SELF);

                TELL airfield TO unloadRollingCargo (SELF);

                TELL airfield TO doAircraftMX(SELF);

                WAIT FOR fuelTrigger TO Fire;
                END WAIT;

                IF location <= TRUNC(numberOfFuelPits)
                    TELL airfield TO fuelAircraftByPit (SELF);
                ELSE
                    TELL airfield TO fuelAircraftByTruck (SELF);
                END IF;

                WAIT FOR finishedTrigger TO Fire;
                END WAIT;

                DISPOSE (fuelIcon);
                DISPOSE (palletIcon);
                DISPOSE (rollingIcon);
                DISPOSE (passengerIcon);
                DISPOSE (mxIcon);

                WAIT FOR SELF TO MoveTo (3.0, 6.63 -
(getLocation*0.3));
                    END WAIT;
                ASK SELF TO SetRotationSpeed (-1.3);
                WAIT FOR SELF TO RotateTo (-3.1415);
                END WAIT;
```

```
        ASK airfield TO updateParkingSpace (location);

        ASK airfield.loadingDock TO TakeBack (SELF,1);
    END WAIT;

    IF stream3.UniformReal(0.0,1.0) < nonConMXprob
        INC (totalBreakdowns);
        WAIT FOR SELF TO repairAircraft;
        END WAIT;
    ELSE
    END IF;

    WAIT FOR SELF TO takeOff;

        aircraftTIS := SimTime - beginWaitTime;

        DISPOSE (SELF);

    END WAIT;

END METHOD;

TELL METHOD takeOff;

BEGIN
    WAIT FOR SELF TO MoveTo (3.0,3.20);
    END WAIT;
    WAIT FOR SELF TO RotateTo (-4.712);
    END WAIT;
    WAIT FOR SELF TO MoveTo (1.68,3.20);
    END WAIT;

    WAIT FOR airfield.runway TO Give (SELF,1);

        WAIT FOR SELF TO MoveTo (0.38,3.20);
        END WAIT;

        ASK airfield.runway TO TakeBack (SELF,1);
    END WAIT;

    DEC (currentOnGround);

    WAIT FOR SELF TO RotateTo (0.0);
    END WAIT;
    SetSpeed (150.0/60.0);
    WAIT FOR SELF TO MoveTo (0.38,4.10);
    END WAIT;
    SetSpeed (400.0/60.0);
    WAIT FOR SELF TO MoveTo (0.38,9.85);
    END WAIT;

    INC (totalDepartures);

    mog := FLOAT(totalDepartures)*(1440.0/SimTime);
    ASK GETMONITOR (mog, RDataPtMObj) TO SetGraph(chart4);
```

```
        ASK outputStream TO WriteReal (mog, 10, 3);
        ASK outputStream TO WriteReal (SimTime, 10, 3);
        ASK outputStream TO WriteLn;


END METHOD;

TELL METHOD repairAircraft;

BEGIN

        WAIT FOR SELF TO MoveTo (3.0,4.0);
        END WAIT;
        WAIT FOR SELF TO RotateTo (-4.712);
        END WAIT;
        WAIT FOR SELF TO MoveTo (2.55,4.0);
        END WAIT;

        WAIT DURATION (stream3.Exponential(nonConMXdur));
        END WAIT;

        WAIT FOR SELF TO RotateTo (-1.57);
        END WAIT;
        WAIT FOR SELF TO MoveTo (3.0,4.0);
        END WAIT;
        WAIT FOR SELF TO RotateTo (-3.1415);
        END WAIT;

END METHOD;

ASK METHOD setLocation (IN number : INTEGER);

BEGIN
        location := number;
END METHOD;

ASK METHOD getLocation : REAL;

VAR
        temp : REAL;
BEGIN
        temp := FLOAT (location);
        RETURN temp;
END METHOD;

ASK METHOD setBeginWaitTime;
BEGIN
        beginWaitTime := SimTime;
END METHOD;

ASK METHOD setEndWaitTime;
BEGIN
        endWaitTime := SimTime;
END METHOD;

ASK METHOD initIcons;
```

```
BEGIN

        NEW(fuelIcon);
        ASK fuelIcon TO LoadFromLibrary (library, "icon");
        ASK window TO AddGraphic (fuelIcon);
        ASK fuelIcon TO SetColor (Red);
        ASK fuelIcon TO DisplayAt (3.02, 6.65 - (getLocation*0.3));

        NEW(palletIcon);
        ASK palletIcon TO LoadFromLibrary (library, "icon");
        ASK window TO AddGraphic (palletIcon);
        ASK palletIcon TO SetColor (Red);
        ASK palletIcon TO DisplayAt (3.1, 6.65 -
(getLocation*0.3));

        NEW(rollingIcon);
        ASK rollingIcon TO LoadFromLibrary (library, "icon");
        ASK window TO AddGraphic (rollingIcon);
        ASK rollingIcon TO SetColor (Red);
        ASK rollingIcon TO DisplayAt (3.18, 6.65 -
(getLocation*0.3));

        NEW(passengerIcon);
        ASK passengerIcon TO LoadFromLibrary (library, "icon");
        ASK window TO AddGraphic (passengerIcon);
        ASK passengerIcon TO SetColor (Red);
        ASK passengerIcon TO DisplayAt (3.27, 6.65 -
(getLocation*0.3));

        NEW(mxIcon);
        ASK mxIcon TO LoadFromLibrary (library, "icon");
        ASK window TO AddGraphic (mxIcon);
        ASK mxIcon TO SetColor (Green);
        ASK mxIcon TO DisplayAt (2.93, 6.65 - (getLocation*0.3));

    END METHOD;

    ASK METHOD beginIcon (IN which : INTEGER);

    BEGIN
        CASE which
            WHEN 1:
                fuelBegin := TRUE;
                ASK fuelIcon TO SetColor (Yellow);
                ASK fuelIcon TO Draw;
            WHEN 2:
                palletBegin := TRUE;
                ASK palletIcon TO SetColor (Yellow);
                ASK palletIcon TO Draw;
            WHEN 3:
                rollingBegin := TRUE;
                ASK rollingIcon TO SetColor (Yellow);
                ASK rollingIcon TO Draw;
            WHEN 4:
                passengerBegin := TRUE;
                ASK passengerIcon TO SetColor (Yellow);
                ASK passengerIcon TO Draw;
```

```
                WHEN 5:
                    mxBegin := TRUE;
                    ASK mxIcon TO SetColor (Yellow);
                    ASK mxIcon TO Draw;
            END CASE;
        END METHOD;

        ASK METHOD endIcon (IN which : INTEGER);

        BEGIN
            CASE which
                WHEN 1:
                    fuelFinish := TRUE;
                    fuelBegin := FALSE;
                    ASK fuelIcon TO SetColor (Green);
                    ASK fuelIcon TO Draw;
                WHEN 2:
                    palletFinish := TRUE;
                    palletBegin := FALSE;
                    ASK palletIcon TO SetColor (Green);
                    ASK palletIcon TO Draw;
                WHEN 3:
                    rollingFinish := TRUE;
                    rollingBegin := FALSE;
                    ASK rollingIcon TO SetColor (Green);
                    ASK rollingIcon TO Draw;
                WHEN 4:
                    passengerFinish := TRUE;
                    passengerBegin := FALSE;
                    ASK passengerIcon TO SetColor (Green);
                    ASK passengerIcon TO Draw;
                WHEN 5:
                    mxFinish := TRUE;
                    mxBegin := FALSE;
                    ASK mxIcon TO SetColor (Green);
                    ASK mxIcon TO Draw;
            END CASE;
        END METHOD;

        ASK METHOD checkForCompletion;

        BEGIN
            IF
((palletFinish)AND(rollingFinish)AND(passengerFinish)AND(fuelFinish)AND
(mxFinish))
                ASK finishedTrigger TO Release;
            ELSIF
((palletFinish)AND(rollingFinish)AND(passengerFinish)AND(mxFinish))
                ASK fuelTrigger TO Release;
            END IF;

            IF
((palletBegin)OR(rollingBegin)OR(passengerBegin)OR(fuelBegin)OR(mxBegin
))
                newState := 1;
            ELSE
                newState := 0;
```

69

```
                END IF;

                IF oldState < newState
                    INC(service);
                    INC(timeAvg);
                ELSIF oldState > newState
                    DEC(service);
                    DEC(timeAvg);
                ELSE
                END IF;
                ASK GETMONITOR (service, IDataPtMObj) TO SetGraph(chart2);
                oldState := newState;

        END METHOD;

    END OBJECT;
END MODULE.
```

Airfield Modules (Definition and Implementation)

```
DEFINITION MODULE airfieldMod;

    FROM globalMod        IMPORT  cargo,

                                  palletsHeadedToA,
                                  palletsHeadedToB,
                                  palletsHeadedToC,
                                  palletsHeadedToD,

                                  rollingHeadedToA,
                                  rollingHeadedToB,
                                  rollingHeadedToC,
                                  rollingHeadedToD,

                                  passengersHeadedToA,
                                  passengersHeadedToB,
                                  passengersHeadedToC,
                                  passengersHeadedToD,

                                  palletsOnDock,
                                  rollingOnDock,
                                  passengersOnDock,

                                  cargoAtDestA,  rollingAtDestA,
passengersAtDestA,
                                  cargoAtDestB,  rollingAtDestB,
passengersAtDestB,
                                  cargoAtDestC,  rollingAtDestC,
passengersAtDestC,
                                  cargoAtDestD,  rollingAtDestD,
passengersAtDestD,

                                  totalAtDestA,
                                  totalAtDestB,
                                  totalAtDestC,
                                  totalAtDestD,

                                  meanInterArrTime,
                                  totalPalletCargo,
                                  totalRollingCargo,
                                  totalPassengers,
                                  numberOfLoadingDocks,
                                  numberOfTrucks,
                                  numberOfCargoHandlers,
                                  numberOfForklifts,
                                  numberOfFuelTrucks,

                                  cargoTimeToUnloadPallet,
                                  cargoPalletsPerHandler,
                                  fuelRatePit,
                                  fuelRateTruck,
                                  fuelTruckCapacity,
```

71

```
                        palletPerTruck,
                        rollingPerTruck,
                        passengerPerTruck,
                        netDistanceToA,
                        netDistanceToB,
                        netDistanceToC,
                        netDistanceToD,

                        incLDocks,
                        decLDocks,
                        incTrucks,
                        decTrucks,
                        incHandlers,
                        decHandlers,
                        incForklifts,
                        decForklifts,
                        incFuelTrucks,
                        decFuelTrucks,

                        currentOnGround,
                        maxOnGround,
                        cargoCounter,
                        totalDiverts,
                        runLength,

                        mainMenu,
                        chart1,
                        chart3,
                        window,
                        library,
                        timeAvgStats,
                        timeAvg,

                        stream1,
                        stream3;

FROM aircraftMod      IMPORT   aircraftObj;
FROM cargoMod         IMPORT   cargoObj, cargoHandlerObj;
FROM ResMod           IMPORT   ResourceObj, EntryObj;
FROM GrpMod           IMPORT   StatQueueObj;
FROM SimMod           IMPORT   SimTime, StopSimulation;
FROM Graph            IMPORT   RDataPtMObj, RDataPt;
FROM MathMod          IMPORT   CEIL;
FROM GTypes           IMPORT ALL ColorType;
FROM Animate          IMPORT   DynImageObj;

TYPE
    airfieldObj =  OBJECT

        loadingDock      : ResourceObj;
        motorPool        : ResourceObj;
        cargoHandler     : ResourceObj;
        forklifts        : ResourceObj;
        runway           : ResourceObj;
        fuelTrucks       : ResourceObj;
        palletQueueA,
```

72

```
                palletQueueB,
                palletQueueC,
                palletQueueD,
                rollingQueueA,
                rollingQueueB,
                rollingQueueC,
                rollingQueueD,
                passengerQueueA,
                passengerQueueB,
                passengerQueueC,
                passengerQueueD,
                parkingQueue      :     StatQueueObj;
                parkingSpace      :     parkingSpaceObj;

                ASK METHOD ObjInit;
                TELL METHOD generateArrivals;
                TELL METHOD unloadCargo (IN plane : aircraftObj);
                TELL METHOD processCargo (IN plane : aircraftObj; IN
handlers : INTEGER);
                TELL METHOD manageLoadingDock (IN pieceOfCargo : cargoObj);
                TELL METHOD unloadPassengers (IN plane : aircraftObj);
                TELL METHOD unloadRollingCargo (IN plane : aircraftObj);
                TELL METHOD fuelAircraftByPit (IN plane : aircraftObj);
                TELL METHOD fuelAircraftByTruck (IN plane : aircraftObj);
                TELL METHOD refillFuelTrucks (IN amount : REAL; IN number :
INTEGER; IN plane : aircraftObj);
                TELL METHOD doAircraftMX (IN plane : aircraftObj);
                TELL METHOD sendTruck (IN where : INTEGER; IN whatType :
INTEGER);
                TELL METHOD driveTruckTo (IN where : INTEGER; IN whatType :
INTEGER);
                TELL METHOD driveTruckFrom (IN where : INTEGER);
                ASK METHOD incrementLDocks (IN number : REAL);
                TELL METHOD decrementLDocks (IN number : REAL);
                ASK METHOD incrementTrucks (IN number : REAL);
                TELL METHOD decrementTrucks (IN number : REAL);
                ASK METHOD incrementHandlers (IN number : REAL);
                TELL METHOD decrementHandlers (IN number : REAL);
                ASK METHOD incrementForklifts (IN number : REAL);
                TELL METHOD decrementForklifts (IN number : REAL);
                ASK METHOD incrementFuelTrucks (IN number : REAL);
                TELL METHOD decrementFuelTrucks (IN number : REAL);
                ASK METHOD nextParkingSpace : INTEGER;
                ASK METHOD updateParkingSpace (IN number : INTEGER);
                ASK METHOD incrementParkingSpace (IN number : INTEGER);
                TELL METHOD decrementParkingSpace (IN number : INTEGER);

        END OBJECT;

        parkingSpaceObj = OBJECT

                id        :     INTEGER;
                inUse     :     BOOLEAN;

                ASK METHOD setId (IN number : INTEGER);
                ASK METHOD getId : INTEGER;
                ASK METHOD setInUse (IN use : BOOLEAN);
```

```
        END OBJECT;

END MODULE.
```

```
IMPLEMENTATION MODULE airfieldMod;

    OBJECT airfieldObj;

        ASK METHOD ObjInit;

        VAR
            i :   INTEGER;

        BEGIN

            NEW (loadingDock);
            ASK loadingDock TO Create (TRUNC(numberOfLoadingDocks));
            ASK loadingDock TO SetPendStats      (TRUE);
            ASK loadingDock TO SetAllocationStats   (TRUE);

            NEW (motorPool);
            ASK motorPool TO Create (TRUNC(numberOfTrucks));
            ASK motorPool TO SetPendStats         (TRUE);
            ASK motorPool TO SetAllocationStats (TRUE);

            NEW (cargoHandler);
            ASK cargoHandler TO Create (TRUNC(numberOfCargoHandlers));
            ASK cargoHandler TO SetPendStats      (TRUE);
            ASK cargoHandler TO SetAllocationStats (TRUE);

            NEW (forklifts);
            ASK forklifts TO Create (TRUNC(numberOfForklifts));
            ASK forklifts TO SetPendStats (TRUE);
            ASK forklifts TO SetAllocationStats (TRUE);

            NEW (fuelTrucks);
            ASK fuelTrucks TO Create (TRUNC(numberOfFuelTrucks));
            ASK fuelTrucks TO SetPendStats (TRUE);
            ASK fuelTrucks TO SetAllocationStats (TRUE);

            NEW (runway);
            ASK runway TO Create (1);

            NEW (palletQueueA); NEW (rollingQueueA); NEW
(passengerQueueA);
            NEW (palletQueueB); NEW (rollingQueueB); NEW
(passengerQueueB);
            NEW (palletQueueC); NEW (rollingQueueC); NEW
(passengerQueueC);
            NEW (palletQueueD); NEW (rollingQueueD); NEW
(passengerQueueD);

            NEW (parkingQueue);

            FOR i := 1 TO TRUNC(numberOfLoadingDocks)
                NEW (parkingSpace);
                ASK parkingSpace TO setId (i);
                ASK parkingSpace TO setInUse (FALSE);
                ASK parkingQueue TO Add (parkingSpace);
            END FOR;
```

```
END METHOD;

TELL METHOD generateArrivals;

    VAR
        aircraft : aircraftObj;
        interArrTime : REAL;
    BEGIN
        NEW (timeAvgStats);
        ADDMONITOR (timeAvg, timeAvgStats);

        WHILE SimTime < runLength*60.0
            ASK mainMenu TO updateTrucks;
            ASK mainMenu TO updateHandlers;
            ASK mainMenu TO updateForklifts;
            ASK mainMenu TO updateFuelTrucks;

            interArrTime := stream1.Exponential
(meanInterArrTime);
            IF SimTime + interArrTime > runLength*60.0
                WAIT DURATION (runLength*60.0 - SimTime);
                END WAIT;
                ASK mainMenu TO showStatistics;
                REMOVEMONITOR (timeAvg, timeAvgStats);
                HALT;
            ELSE
            END IF;

            WAIT DURATION (interArrTime);

                IF FLOAT(currentOnGround) >= maxOnGround      {AC
divert logic}
                    ASK mainMenu TO updateDiverts;
                ELSE
                    NEW (aircraft);
                    TELL aircraft TO land;
                    ASK GETMONITOR (aircraft.aircraftTIS,
RDataPtMObj) TO SetGraph (chart1);
                    ASK GETMONITOR (aircraft.aircraftTIS,
RDataPtMObj) TO SetHistMode (TRUE);
                END IF;

            END WAIT;
        END WHILE;
        ASK mainMenu TO showStatistics;
        REMOVEMONITOR (timeAvg, timeAvgStats);
        HALT;

END METHOD;

TELL METHOD unloadPassengers (IN plane : aircraftObj);
VAR
    cargo : cargoObj;

BEGIN
    ASK plane TO beginIcon(4);
    ASK plane TO checkForCompletion;
```

```
            WHILE plane.passengerCargo > 0
                ASK plane TO decPassengerCargo;
                NEW(cargo);
                INC(totalPassengers);
                ASK cargo TO setCargoType(3);
                ASK cargo TO setCargoDestination;
                ASK cargo TO incrementCargoOnDock;
                manageLoadingDock(cargo);
                WAIT DURATION (0.5);
                END WAIT;
            END WHILE;

            ASK plane TO endIcon(4);
            ASK plane TO checkForCompletion;

    END METHOD;

    TELL METHOD fuelAircraftByPit (IN plane : aircraftObj);

    BEGIN
        ASK plane TO beginIcon (1);
        ASK plane TO checkForCompletion;
        WAIT DURATION (plane.fuel/fuelRatePit);
        END WAIT;
        ASK plane TO endIcon (1);
        ASK plane TO checkForCompletion;
    END METHOD;

    TELL METHOD fuelAircraftByTruck (IN plane : aircraftObj);
    VAR
        temp : EntryObj;
        attachedTrucks : INTEGER;
        fuelRate      : REAL;
    BEGIN
        attachedTrucks := 0;
        WHILE plane.fuel > 0.0
            IF fuelTrucks.Resources < plane.maxFuelers
                WAIT FOR fuelTrucks TO Give (plane, 1);
                END WAIT;
                INC (attachedTrucks);
                fuelRate := fuelRateTruck;
                ASK plane TO beginIcon (1);
                ASK plane TO checkForCompletion;
                ASK mainMenu TO updateFuelTrucks;
            ELSE
                WAIT FOR fuelTrucks TO Give (plane,
plane.maxFuelers);
                END WAIT;
                attachedTrucks := plane.maxFuelers;
                fuelRate := FLOAT(plane.maxFuelers)*fuelRateTruck;
                ASK plane TO beginIcon (1);
                ASK plane TO checkForCompletion;
                ASK mainMenu TO updateFuelTrucks;
            END IF;
```

```
                    IF plane.fuel >=
fuelTruckCapacity*FLOAT(attachedTrucks)
                        IF fuelRate <= plane.fuelRate
                            WAIT DURATION
(fuelTruckCapacity*FLOAT(attachedTrucks)/fuelRate);
                            END WAIT;
                        ELSE
                            WAIT DURATION
(fuelTruckCapacity*FLOAT(attachedTrucks)/plane.fuelRate);
                            END WAIT;
                        END IF;
                        ASK plane TO
decrementFuel(fuelTruckCapacity*FLOAT(attachedTrucks));

refillFuelTrucks(fuelTruckCapacity,attachedTrucks,plane);
                        attachedTrucks := 0;
                    ELSE
                        IF fuelRate <= plane.fuelRate
                            WAIT DURATION (plane.fuel/fuelRate);
                            END WAIT;
                        ELSE
                            WAIT DURATION (plane.fuel/plane.fuelRate);
                            END WAIT;
                        END IF;
                        refillFuelTrucks(plane.fuel,attachedTrucks,plane);
                        ASK plane TO decrementFuel(plane.fuel);
                        attachedTrucks := 0;
                    END IF;
                END WHILE;

                ASK plane TO endIcon (1);
                ASK plane TO checkForCompletion;

        END METHOD;

        TELL METHOD refillFuelTrucks (IN amount : REAL; IN number :
INTEGER; IN plane : aircraftObj);
        BEGIN
            WAIT DURATION (amount/500.0);
            END WAIT;
            ASK fuelTrucks TO TakeBack (plane, number);
            ASK mainMenu TO updateFuelTrucks;
        END METHOD;

        TELL METHOD doAircraftMX (IN plane : aircraftObj);

        BEGIN
            IF stream3.UniformReal(0.0,1.0) < plane.concurMXprob
                ASK plane TO beginIcon (5);
                ASK plane TO checkForCompletion;
                WAIT DURATION (stream3.Exponential(plane.concurMXdur));
                END WAIT;
                ASK plane TO endIcon (5);
            ELSE
                ASK plane TO endIcon (5);
            END IF;
```

```
              ASK plane TO checkForCompletion;
        END METHOD;


        TELL METHOD unloadRollingCargo (IN plane : aircraftObj);
        VAR
              cargo : cargoObj;

        BEGIN
              ASK plane TO beginIcon (3);
              ASK plane TO checkForCompletion;
              WHILE plane.rollingCargo > 0
                    ASK plane TO decRollingCargo;
                    NEW(cargo);
                    INC(totalRollingCargo);
                    ASK cargo TO setCargoType(2);
                    ASK cargo TO setCargoDestination;
                    ASK cargo TO incrementCargoOnDock;
                    manageLoadingDock(cargo);
                    WAIT DURATION (2.0);
                    END WAIT;
              END WHILE;
              ASK plane TO endIcon (3);
              ASK plane TO checkForCompletion;
        END METHOD;


        TELL METHOD unloadCargo (IN plane : aircraftObj);
        VAR
              temp : EntryObj;
              need : INTEGER;
        BEGIN

              need :=
CEIL(FLOAT(plane.palletCargo)/cargoPalletsPerHandler);
              WHILE (cargoHandler.Resources > 0) AND
(cargoHandler.NumberAllocatedTo(plane)
                    < need)
                    WAIT FOR cargoHandler TO Give (plane, 1);
                    END WAIT;
              END WHILE;

              IF cargoHandler.NumberAllocatedTo(plane) = 0
                    temp := cargoHandler.AllocationList.First;
                    IF need <= temp.Number
                        WAIT FOR cargoHandler TO Give (plane, need);
                        END WAIT;
                    ELSE
                        WAIT FOR cargoHandler TO Give (plane, temp.Number);
                        END WAIT;
                    END IF;
              END IF;

              ASK mainMenu TO updateHandlers;

              WAIT FOR SELF TO processCargo (plane,
cargoHandler.NumberAllocatedTo(plane));
              END WAIT;
```

```
        END METHOD;

        TELL METHOD processCargo (IN plane : aircraftObj; IN handlers :
INTEGER);

        VAR
            kLoader         :   cargoHandlerObj;
            cargoTeam       :   ResourceObj;
            loadingPriority :   ResourceObj;

        BEGIN
            NEW (cargoTeam);
            ASK cargoTeam TO Create (handlers);
            NEW (loadingPriority);
            ASK loadingPriority TO Create (1);

            ASK plane TO beginIcon(2);
            ASK plane TO checkForCompletion;

            WHILE plane.getPalletCargo > 0
                WAIT FOR cargoTeam TO Give (plane, 1);
                    NEW (kLoader);
                    TELL kLoader TO unloadAircraft (plane, cargoTeam,
loadingPriority);
                END WAIT;
            END WHILE;

            ASK plane TO endIcon(2);
            ASK plane TO checkForCompletion;

            ASK cargoHandler TO TakeBack (plane, handlers);
            ASK mainMenu TO updateHandlers;

        END METHOD;

        TELL METHOD manageLoadingDock(IN pieceOfCargo : cargoObj);

        BEGIN

            CASE pieceOfCargo.destination
                WHEN 1:
                    CASE pieceOfCargo.typeOfCargo
                        WHEN 1:
                            ASK palletQueueA TO Add (pieceOfCargo);
                            IF palletsHeadedToA >=
TRUNC(palletPerTruck)
                                sendTruck(1,1);
                                palletsHeadedToA := palletsHeadedToA -
TRUNC(palletPerTruck);
                            END IF;
                        WHEN 2:
                            ASK rollingQueueA TO Add (pieceOfCargo);
                            IF rollingHeadedToA >=
TRUNC(rollingPerTruck)
                                sendTruck(1,2);
                                rollingHeadedToA := rollingHeadedToA -
TRUNC(rollingPerTruck);
```

80

```
                                END IF;
                     WHEN 3:
                        ASK passengerQueueA TO Add (pieceOfCargo);
                        IF passengersHeadedToA >=
TRUNC(passengerPerTruck)
                                sendTruck(1,3);
                                passengersHeadedToA :=
passengersHeadedToA - TRUNC(passengerPerTruck);
                                END IF;
                  END CASE;
            WHEN 2:
                  CASE pieceOfCargo.typeOfCargo
                     WHEN 1:
                        ASK palletQueueB TO Add (pieceOfCargo);
                        IF palletsHeadedToB >=
TRUNC(palletPerTruck)
                                sendTruck(2,1);
                                palletsHeadedToB := palletsHeadedToB -
TRUNC(palletPerTruck);
                                END IF;
                     WHEN 2:
                        ASK rollingQueueB TO Add (pieceOfCargo);
                        IF rollingHeadedToB >=
TRUNC(rollingPerTruck)
                                sendTruck(2,2);
                                rollingHeadedToB := rollingHeadedToB -
TRUNC(rollingPerTruck);
                                END IF;
                     WHEN 3:
                        ASK passengerQueueB TO Add (pieceOfCargo);
                        IF passengersHeadedToB >=
TRUNC(passengerPerTruck)
                                sendTruck(2,3);
                                passengersHeadedToB :=
passengersHeadedToB - TRUNC(passengerPerTruck);
                                END IF;
                  END CASE;
            WHEN 3:
                  CASE pieceOfCargo.typeOfCargo
                     WHEN 1:
                        ASK palletQueueC TO Add (pieceOfCargo);
                        IF palletsHeadedToC >=
TRUNC(palletPerTruck)
                                sendTruck(3,1);
                                palletsHeadedToC := palletsHeadedToC -
TRUNC(palletPerTruck);
                                END IF;
                     WHEN 2:
                        ASK rollingQueueC TO Add (pieceOfCargo);
                        IF rollingHeadedToC >=
TRUNC(rollingPerTruck)
                                sendTruck(3,2);
                                rollingHeadedToC := rollingHeadedToC -
TRUNC(rollingPerTruck);
                                END IF;
                     WHEN 3:
                        ASK passengerQueueC TO Add (pieceOfCargo);
```

```
                                        IF passengersHeadedToC >=
TRUNC(passengerPerTruck)
                                                sendTruck(3,3);
                                                passengersHeadedToC :=
passengersHeadedToC - TRUNC(passengerPerTruck);
                                        END IF;
                                END CASE;
                        WHEN 4:
                                CASE pieceOfCargo.typeOfCargo
                                        WHEN 1:
                                                ASK palletQueueD TO Add (pieceOfCargo);
                                                IF palletsHeadedToD >=
TRUNC(palletPerTruck)
                                                        sendTruck(4,1);
                                                        palletsHeadedToD := palletsHeadedToD -
TRUNC(palletPerTruck);
                                                END IF;
                                        WHEN 2:
                                                ASK rollingQueueD TO Add (pieceOfCargo);
                                                IF rollingHeadedToD >=
TRUNC(rollingPerTruck)
                                                        sendTruck(4,2);
                                                        rollingHeadedToD := rollingHeadedToD -
TRUNC(rollingPerTruck);
                                                END IF;
                                        WHEN 3:
                                                ASK passengerQueueD TO Add (pieceOfCargo);
                                                IF passengersHeadedToD >=
TRUNC(passengerPerTruck)
                                                        sendTruck(4,3);
                                                        passengersHeadedToD :=
passengersHeadedToD - TRUNC(passengerPerTruck);
                                                END IF;
                                END CASE;
                        END CASE;
                END METHOD;

        TELL METHOD sendTruck (IN where : INTEGER; IN whatType :
INTEGER);
                VAR
                        tempA,
                        tempB,
                        tempC,
                        tempD      :   cargoObj;
                        i, n       :   INTEGER;

        BEGIN
                WAIT FOR motorPool TO Give (SELF, 1);
                        ASK mainMenu TO updateTrucks;

                        CASE where
                                WHEN 1:
                                        CASE whatType
                                                WHEN 1:
                                                        FOR i := 1 TO TRUNC(palletPerTruck)
                                                                WAIT FOR forklifts TO Give (SELF,
1);
```

```
                                        END WAIT;
                                        WAIT DURATION (0.5);
                                        END WAIT;
                                        palletsOnDock:= palletsOnDock - 1;
                                        ASK mainMenu TO updateForklifts;
                                        ASK forklifts TO TakeBack (SELF,
1);
                                    END FOR;

                                    WAIT FOR SELF TO driveTruckTo(1,1);
                                    END WAIT;

                                    FOR n := 1 TO TRUNC(palletPerTruck)
                                        tempA := palletQueueA.Remove();
                                        WAIT DURATION (0.5);
                                        END WAIT;
                                        INC (cargoAtDestA);
                                        INC (totalAtDestA);
                                    END FOR;

                                    WAIT FOR SELF TO driveTruckFrom(1);
                                    END WAIT;
                                WHEN 2:
                                    FOR i := 1 TO TRUNC(rollingPerTruck)
                                        WAIT DURATION (0.5);
                                        END WAIT;
                                        rollingOnDock := rollingOnDock - 1;
                                    END FOR;

                                    WAIT FOR SELF TO driveTruckTo(1,2);
                                    END WAIT;

                                    FOR n := 1 TO TRUNC(rollingPerTruck)
                                        tempA := rollingQueueA.Remove();
                                        WAIT DURATION (0.5);
                                        END WAIT;
                                        INC (rollingAtDestA);
                                        INC (totalAtDestA);
                                    END FOR;

                                    WAIT FOR SELF TO driveTruckFrom(1);
                                    END WAIT;
                                WHEN 3:
                                    WAIT DURATION(passengerPerTruck/20.0);
                                    END WAIT;

                                    passengersOnDock := passengersOnDock -
TRUNC(passengerPerTruck);

                                    WAIT FOR SELF TO driveTruckTo(1,3);
                                    END WAIT;

                                    FOR n := 1 TO TRUNC(passengerPerTruck)
                                        tempA := passengerQueueA.Remove();
                                        INC (passengersAtDestA);
                                        INC (totalAtDestA);
                                    END FOR;
```

```
                                    WAIT DURATION (passengerPerTruck/20.0);
                                    END WAIT;

                                    WAIT FOR SELF TO driveTruckFrom(1);
                                    END WAIT;
                        END CASE;
                WHEN 2:
                    CASE whatType
                        WHEN 1:
                            FOR i := 1 TO TRUNC(palletPerTruck)
                                WAIT FOR forklifts TO Give (SELF,
1);
                                END WAIT;
                                WAIT DURATION (0.5);
                                END WAIT;
                                palletsOnDock:= palletsOnDock - 1;
                                ASK mainMenu TO updateForklifts;
                                ASK forklifts TO TakeBack (SELF,
1);
                            END FOR;

                            WAIT FOR SELF TO driveTruckTo(2,1);
                            END WAIT;

                            FOR n := 1 TO TRUNC(palletPerTruck)
                                tempA := palletQueueB.Remove();
                                WAIT DURATION (0.5);
                                END WAIT;
                                INC (cargoAtDestB);
                                INC (totalAtDestB);
                            END FOR;

                            WAIT FOR SELF TO driveTruckFrom(2);
                            END WAIT;
                        WHEN 2:
                            FOR i := 1 TO TRUNC(rollingPerTruck)
                                WAIT DURATION (0.5);
                                END WAIT;
                                rollingOnDock := rollingOnDock - 1;
                            END FOR;

                            WAIT FOR SELF TO driveTruckTo(2,2);
                            END WAIT;

                            FOR n := 1 TO TRUNC(rollingPerTruck)
                                tempA := rollingQueueB.Remove();
                                WAIT DURATION (0.5);
                                END WAIT;
                                INC (rollingAtDestB);
                                INC (totalAtDestB);
                            END FOR;

                            WAIT FOR SELF TO driveTruckFrom(2);
                            END WAIT;
                        WHEN 3:
                            WAIT DURATION (passengerPerTruck/20.0);
                            END WAIT;
```

```
                                  passengersOnDock := passengersOnDock -
    TRUNC(passengerPerTruck);

                                  WAIT FOR SELF TO driveTruckTo(2,3);
                                  END WAIT;

                                  FOR n := 1 TO TRUNC(passengerPerTruck)
                                      tempA := passengerQueueB.Remove();
                                      INC (passengersAtDestB);
                                      INC (totalAtDestB);
                                  END FOR;
                                  WAIT DURATION (passengerPerTruck/20.0);
                                  END WAIT;

                                  WAIT FOR SELF TO driveTruckFrom(2);
                                  END WAIT;
                          END CASE;
                      WHEN 3:
                          CASE whatType
                              WHEN 1:
                                  FOR i := 1 TO TRUNC(palletPerTruck)
                                      WAIT FOR forklifts TO Give (SELF,
    1);

                                      END WAIT;
                                      WAIT DURATION (0.5);
                                      END WAIT;
                                      palletsOnDock:= palletsOnDock - 1;
                                      ASK mainMenu TO updateForklifts;
                                      ASK forklifts TO TakeBack (SELF,
    1);

                                  END FOR;

                                  WAIT FOR SELF TO driveTruckTo(3,1);
                                  END WAIT;

                                  FOR n := 1 TO TRUNC(palletPerTruck)
                                      tempA := palletQueueC.Remove();
                                      WAIT DURATION (0.5);
                                      END WAIT;
                                      INC (cargoAtDestC);
                                      INC (totalAtDestC);
                                  END FOR;

                                  WAIT FOR SELF TO driveTruckFrom(3);
                                  END WAIT;
                              WHEN 2:
                                  FOR i := 1 TO TRUNC(rollingPerTruck)
                                      WAIT DURATION (0.5);
                                      END WAIT;
                                      rollingOnDock:= rollingOnDock - 1;
                                  END FOR;

                                  WAIT FOR SELF TO driveTruckTo(3,2);
                                  END WAIT;

                                  FOR n := 1 TO TRUNC(rollingPerTruck)
```

```
                                tempA := rollingQueueC.Remove();
                                WAIT DURATION (0.5);
                                END WAIT;
                                INC (rollingAtDestC);
                                INC (totalAtDestC);
                            END FOR;

                            WAIT FOR SELF TO driveTruckFrom(3);
                            END WAIT;
                    WHEN 3:
                            WAIT DURATION (passengerPerTruck/20.0);
                            END WAIT;

                            passengersOnDock := passengersOnDock -
TRUNC(passengerPerTruck);

                            WAIT FOR SELF TO driveTruckTo(3,3);
                            END WAIT;

                            FOR n := 1 TO TRUNC(passengerPerTruck)
                                tempA := passengerQueueC.Remove();
                                INC (passengersAtDestC);
                                INC (totalAtDestC);
                            END FOR;
                            WAIT DURATION (passengerPerTruck/20.0);
                            END WAIT;

                            WAIT FOR SELF TO driveTruckFrom(3);
                            END WAIT;
                    END CASE;
                WHEN 4:
                    CASE whatType
                        WHEN 1:
                            FOR i := 1 TO TRUNC(palletPerTruck)
                                WAIT FOR forklifts TO Give (SELF,
1);
                                END WAIT;
                                WAIT DURATION (0.5);
                                END WAIT;
                                palletsOnDock:= palletsOnDock - 1;
                                ASK mainMenu TO updateForklifts;
                                ASK forklifts TO TakeBack (SELF,
1);
                            END FOR;

                            WAIT FOR SELF TO driveTruckTo(4,1);
                            END WAIT;

                            FOR n := 1 TO TRUNC(palletPerTruck)
                                tempA := palletQueueD.Remove();
                                WAIT DURATION (0.5);
                                END WAIT;
                                INC (cargoAtDestD);
                                INC (totalAtDestD);
                            END FOR;

                            WAIT FOR SELF TO driveTruckFrom(4);
```

86

```
                                END WAIT;
                        WHEN 2:
                            FOR i := 1 TO TRUNC(rollingPerTruck)
                                WAIT DURATION (0.5);
                                END WAIT;
                                rollingOnDock:= rollingOnDock - 1;
                            END FOR;

                            WAIT FOR SELF TO driveTruckTo(4,2);
                            END WAIT;

                            FOR n := 1 TO TRUNC(rollingPerTruck)
                                tempA := rollingQueueD.Remove();
                                WAIT DURATION (0.5);
                                END WAIT;
                                INC (rollingAtDestD);
                                INC (totalAtDestD);
                            END FOR;

                            WAIT FOR SELF TO driveTruckFrom(4);
                            END WAIT;
                        WHEN 3:
                            WAIT DURATION (passengerPerTruck/20.0);
                            END WAIT;

                            passengersOnDock := passengersOnDock -
TRUNC(passengerPerTruck);

                            WAIT FOR SELF TO driveTruckTo(4,3);
                            END WAIT;

                            FOR n := 1 TO TRUNC(passengerPerTruck)
                                tempA := passengerQueueD.Remove();
                                INC (passengersAtDestD);
                                INC (totalAtDestD);
                            END FOR;
                            WAIT DURATION (passengerPerTruck/20.0);
                            END WAIT;

                            WAIT FOR SELF TO driveTruckFrom(4);
                            END WAIT;
                    END CASE;
                END CASE;

                ASK motorPool TO TakeBack (SELF, 1);
                ASK mainMenu TO updateTrucks;
                ASK mainMenu TO updateForklifts;
            END WAIT;

        END METHOD;

        ASK METHOD incrementLDocks (IN number : REAL);
        BEGIN
            ASK loadingDock TO IncrementResourcesBy (TRUNC(number));
        END METHOD;

        TELL METHOD decrementLDocks (IN number : REAL);
```

87

```
BEGIN
    TELL loadingDock TO DecrementResourcesBy (TRUNC(number));
END METHOD;


ASK METHOD incrementTrucks (IN number : REAL);
BEGIN
    ASK motorPool TO IncrementResourcesBy (TRUNC(number));
END METHOD;


TELL METHOD decrementTrucks (IN number : REAL);
BEGIN
    TELL motorPool TO DecrementResourcesBy (TRUNC(number));
END METHOD;


ASK METHOD incrementHandlers (IN number : REAL);
BEGIN
    ASK cargoHandler TO IncrementResourcesBy (TRUNC(number));
END METHOD;


TELL METHOD decrementHandlers (IN number : REAL);
BEGIN
    TELL cargoHandler TO DecrementResourcesBy (TRUNC(number));
END METHOD;


ASK METHOD incrementForklifts (IN number : REAL);
BEGIN
    ASK forklifts TO IncrementResourcesBy (TRUNC(number));
END METHOD;


TELL METHOD decrementForklifts (IN number : REAL);
BEGIN
    TELL forklifts TO DecrementResourcesBy (TRUNC(number));
END METHOD;


ASK METHOD incrementFuelTrucks (IN number : REAL);
BEGIN
    ASK fuelTrucks TO IncrementResourcesBy (TRUNC(number));
END METHOD;


TELL METHOD decrementFuelTrucks (IN number : REAL);
BEGIN
    TELL fuelTrucks TO DecrementResourcesBy (TRUNC(number));
END METHOD;


ASK METHOD nextParkingSpace : INTEGER;
VAR
    number   : INTEGER;
    lowNumber: INTEGER;
    temp     : parkingSpaceObj;
BEGIN
    lowNumber := parkingQueue.numberIn;

    FOREACH temp IN parkingQueue
        IF NOT(temp.inUse)
            number := temp.getId;
            IF number < lowNumber
                lowNumber := number;
```

```
                    END IF;
               END IF;
          END FOREACH;

          FOREACH temp IN parkingQueue
               IF temp.id = lowNumber
                    ASK parkingQueue TO RemoveThis (temp);
                    NEW (temp);
                    ASK temp TO setId (lowNumber);
                    ASK temp TO setInUse (TRUE);
                    ASK parkingQueue TO Add (temp);
               END IF;
          END FOREACH;

          RETURN lowNumber;
END METHOD;


ASK METHOD updateParkingSpace (IN number : INTEGER);

VAR
     temp : parkingSpaceObj;
BEGIN
     FOREACH temp IN parkingQueue
          IF temp.id = number
               ASK parkingQueue TO RemoveThis (temp);
               NEW (temp);
               ASK temp TO setInUse (FALSE);
               ASK temp TO setId (number);
               ASK parkingQueue TO Add (temp);
          END IF;
     END FOREACH;
END METHOD;


ASK METHOD incrementParkingSpace (IN number : INTEGER);

VAR
     temp : parkingSpaceObj;
BEGIN
     WHILE number > 0
          NEW (temp);
          ASK temp TO setId (parkingQueue.numberIn + 1);
          ASK temp TO setInUse (FALSE);
          ASK parkingQueue TO Add (temp);
          number := number - 1;
     END WHILE;
END METHOD;


TELL METHOD decrementParkingSpace (IN number : INTEGER);

VAR
     temp     : parkingSpaceObj;
BEGIN
     WHILE number > 0
          temp := parkingQueue.First;
          WHILE temp.id <> parkingQueue.numberIn
               temp := parkingQueue.Next(temp);
          END WHILE;
```

```
                    ASK parkingQueue TO RemoveThis (temp);
                    number := number - 1;
               END WHILE;
          END METHOD;

          TELL METHOD driveTruckTo(IN where : INTEGER; IN whatType :
INTEGER);

          VAR
               truck : DynImageObj;

          BEGIN
               NEW (truck);
               ASK truck TO LoadFromLibrary (library, "truck");
               ASK window TO AddGraphic (truck);

               CASE whatType
                    WHEN 2:
                         ASK truck TO SetColor (Black);
                    WHEN 3:
                         ASK truck TO SetColor (Orange);
                    OTHERWISE;
               END CASE;

               ASK truck TO DisplayAt (4.30,8.06);
               ASK truck TO SetSpeed (2.0);

               WAIT FOR truck TO MoveTo (4.30,8.57)
               END WAIT;

               CASE where

                    WHEN 1:
                         ASK truck TO SetRotationSpeed (2.0);
                         WAIT FOR truck TO RotateTo (1.57);
                         END WAIT;
                         WAIT FOR truck TO MoveTo (1.79,8.57);
                         END WAIT;
                         ASK truck TO SetRotationSpeed (-2.0);
                         WAIT FOR truck TO RotateTo (0.0);
                         END WAIT;
                         WAIT FOR truck TO MoveTo (1.79,9.48);
                         END WAIT;
                         DISPOSE (truck);
                    WHEN 2:
                         ASK truck TO SetRotationSpeed (2.0);
                         WAIT FOR truck TO RotateTo (1.57);
                         END WAIT;
                         WAIT FOR truck TO MoveTo (4.13,8.57);
                         END WAIT;
                         ASK truck TO SetRotationSpeed (-2.0);
                         WAIT FOR truck TO RotateTo (0.0);
                         END WAIT;
                         WAIT FOR truck TO MoveTo (4.13,9.48);
                         END WAIT;
                         DISPOSE (truck);
                    WHEN 3:
```

```
            ASK truck TO SetRotationSpeed (-2.0);
            WAIT FOR truck TO RotateTo (-1.57);
            END WAIT;
            WAIT FOR truck TO MoveTo (6.51,8.57);
            END WAIT;
            ASK truck TO SetRotationSpeed (2.0);
            WAIT FOR truck TO RotateTo (0.0);
            END WAIT;
            WAIT FOR truck TO MoveTo (6.51,9.48);
            END WAIT;
            DISPOSE (truck);
        WHEN 4:
            ASK truck TO SetRotationSpeed (-2.0);
            WAIT FOR truck TO RotateTo (-1.57);
            END WAIT;
            WAIT FOR truck TO MoveTo (8.72,8.57);
            END WAIT;
            ASK truck TO SetRotationSpeed (2.0);
            WAIT FOR truck TO RotateTo (0.0);
            END WAIT;
            WAIT FOR truck TO MoveTo (8.72,9.48);
            END WAIT;
            DISPOSE (truck);
    END CASE;

END METHOD;

TELL METHOD driveTruckFrom (IN where : INTEGER);

VAR
    truck : DynImageObj;
BEGIN
    NEW (truck);
    ASK truck TO LoadFromLibrary (library, "truck");
    ASK window TO AddGraphic (truck);
    ASK truck TO SetRotation (3.1415);
    ASK truck TO SetSpeed (2.0);

    CASE where
        WHEN 1:
            ASK truck TO DisplayAt (1.79,9.48);
            WAIT FOR truck TO MoveTo (1.79,8.57);
            END WAIT;
            ASK truck TO SetRotationSpeed(2.0);
            WAIT FOR truck TO RotateTo (-1.57);
            END WAIT;
            WAIT FOR truck TO MoveTo (4.30,8.57);
            END WAIT;
            ASK truck TO SetRotationSpeed (-2.0);
            WAIT FOR truck TO RotateTo (-3.1415);
            END WAIT;
        WHEN 2:
            ASK truck TO DisplayAt (4.13,9.48);
            WAIT FOR truck TO MoveTo (4.13,8.57);
            END WAIT;
            ASK truck TO SetRotationSpeed(2.0);
            WAIT FOR truck TO RotateTo (-1.57);
```

```
                        END WAIT;
                        WAIT FOR truck TO MoveTo (4.30,8.57);
                        END WAIT;
                        ASK truck TO SetRotationSpeed (-2.0);
                        WAIT FOR truck TO RotateTo (-3.1415);
                        END WAIT;
                    WHEN 3:
                        ASK truck TO DisplayAt (6.51,9.48);
                        WAIT FOR truck TO MoveTo (6.51,8.57);
                        END WAIT;
                        ASK truck TO SetRotationSpeed(-2.0);
                        WAIT FOR truck TO RotateTo (-4.712);
                        END WAIT;
                        WAIT FOR truck TO MoveTo (4.30,8.57);
                        END WAIT;
                        ASK truck TO SetRotationSpeed (2.0);
                        WAIT FOR truck TO RotateTo (-3.1415);
                        END WAIT;
                    WHEN 4:
                        ASK truck TO DisplayAt (8.72,9.48);
                        WAIT FOR truck TO MoveTo (8.72,8.57);
                        END WAIT;
                        ASK truck TO SetRotationSpeed(-2.0);
                        WAIT FOR truck TO RotateTo (-4.712);
                        END WAIT;
                        WAIT FOR truck TO MoveTo (4.30,8.57);
                        END WAIT;
                        ASK truck TO SetRotationSpeed (2.0);
                        WAIT FOR truck TO RotateTo (-3.1415);
                        END WAIT;
                END CASE;

                WAIT FOR truck TO MoveTo (4.30,8.06);
                END WAIT;

                DISPOSE (truck);

            END METHOD;

    END OBJECT;

    OBJECT parkingSpaceObj;

        ASK METHOD setId (IN number : INTEGER);

        BEGIN
            id := number;
        END METHOD;

        ASK METHOD setInUse (IN use : BOOLEAN);

        BEGIN
            inUse := use;
        END METHOD;

        ASK METHOD getId : INTEGER;
        VAR
```

```
            number : INTEGER;
        BEGIN
            number := id;
            RETURN number;
        END METHOD;

    END OBJECT;

END MODULE.
```

Cargo Modules (Definition and Implementation)

```
DEFINITION MODULE cargoMod;

    FROM globalMod   IMPORT  totalPalletCargo,
                             totalRollingCargo,

                             palletsHeadedToA,
                             palletsHeadedToB,
                             palletsHeadedToC,
                             palletsHeadedToD,

                             rollingHeadedToA,
                             rollingHeadedToB,
                             rollingHeadedToC,
                             rollingHeadedToD,

                             passengersHeadedToA,
                             passengersHeadedToB,
                             passengersHeadedToC,
                             passengersHeadedToD,

                             palletsOnDock,
                             rollingOnDock,
                             passengersOnDock,

                             cargoProbToA,
                             cargoProbToB,
                             cargoProbToC,
                             cargoProbToD,

                             mainMenu,
                             airfield,

                             chart3,

                             stream3;

    FROM aircraftMod IMPORT aircraftObj;

    FROM SimMod       IMPORT  SimTime;
    FROM Graph        IMPORT  IDataPtMObj;
    FROM GrpMod       IMPORT  StatQueueObj;
    FROM ResMod       IMPORT  ResourceObj;

    TYPE

        cargoObj = OBJECT

            typeOfCargo :   INTEGER;
            destination :   INTEGER;
            beginWaitTime : REAL;

            ASK METHOD ObjInit;
```

```
        ASK METHOD setCargoType (IN type : INTEGER);
        ASK METHOD setCargoDestination;
        ASK METHOD incrementCargoOnDock;

    END OBJECT;

    cargoHandlerObj = OBJECT

        TELL METHOD unloadAircraft (IN plane : aircraftObj; IN team
: ResourceObj; IN loadingPriority : ResourceObj);

    END OBJECT;

END MODULE.
```

```
IMPLEMENTATION MODULE cargoMod;

    OBJECT cargoObj;

        ASK METHOD ObjInit;
        BEGIN
            beginWaitTime := SimTime;
        END METHOD;

        ASK METHOD setCargoType (IN type : INTEGER);
        BEGIN
            CASE type
                WHEN 1:
                    typeOfCargo := 1;
                WHEN 2:
                    typeOfCargo := 2;
                WHEN 3:
                    typeOfCargo := 3;
            END CASE;

        END METHOD;

        ASK METHOD setCargoDestination;

        VAR
            temp,
            tempToA,
            tempToB,
            tempToC,
            tempToD     : INTEGER;

        BEGIN
            temp :=  stream3.UniformInt (1,100);

            tempToA := TRUNC (100.0*cargoProbToA);
            tempToB := TRUNC (100.0*cargoProbToB) + tempToA;
            tempToC := TRUNC (100.0*cargoProbToC) + tempToB;
            tempToD := TRUNC (100.0*cargoProbToD) + tempToC;

            IF temp < tempToA
                destination := 1;
            ELSIF temp < tempToB
                destination := 2;
            ELSIF temp < tempToC
                destination := 3;
            ELSIF temp < tempToD
                destination := 4;
            ELSE
                destination := 4;
            END IF;

        END METHOD;

        ASK METHOD incrementCargoOnDock;

        BEGIN
```

```
CASE destination
    WHEN 1:
        CASE typeOfCargo
            WHEN 1:
                INC(palletsHeadedToA);
                INC(palletsOnDock);
            WHEN 2:
                INC(rollingHeadedToA);
                INC(rollingOnDock);
            WHEN 3:
                INC(passengersHeadedToA);
                INC(passengersOnDock);
        END CASE;
    WHEN 2:
        CASE typeOfCargo
            WHEN 1:
                INC(palletsHeadedToB);
                INC(palletsOnDock);
            WHEN 2:
                INC(rollingHeadedToB);
                INC(rollingOnDock);
            WHEN 3:
                INC(passengersHeadedToB);
                INC(passengersOnDock);
        END CASE;
    WHEN 3:
        CASE typeOfCargo
            WHEN 1:
                INC(palletsHeadedToC);
                INC(palletsOnDock);
            WHEN 2:
                INC(rollingHeadedToC);
                INC(rollingOnDock);
            WHEN 3:
                INC(passengersHeadedToC);
                INC(passengersOnDock);
        END CASE;
    WHEN 4:
        CASE typeOfCargo
            WHEN 1:
                INC(palletsHeadedToD);
                INC(palletsOnDock);
            WHEN 2:
                INC(rollingHeadedToD);
                INC(rollingOnDock);
            WHEN 3:
                INC(passengersHeadedToD);
                INC(passengersOnDock);
        END CASE;
END CASE;

ASK GETMONITOR (palletsOnDock, IDataPtMObj) TO SetGraph
(chart3);
ASK GETMONITOR (palletsOnDock, IDataPtMObj) TO SetElement
(0);
ASK GETMONITOR (rollingOnDock, IDataPtMObj) TO SetGraph
(chart3);
```

97

```
                ASK GETMONITOR (rollingOnDock, IDataPtMObj) TO SetElement
(1);
                ASK GETMONITOR (passengersOnDock, IDataPtMObj) TO SetGraph
(chart3);
                ASK GETMONITOR (passengersOnDock, IDataPtMObj) TO
SetElement (2);

        END METHOD;

    END OBJECT;

    OBJECT cargoHandlerObj;

    TELL METHOD unloadAircraft (IN plane : aircraftObj; IN team :
ResourceObj; IN loadingPriority : ResourceObj);

        VAR
            temp : REAL;
            cargoQueue : StatQueueObj;
            cargo : cargoObj;

        BEGIN
            temp := cargoPalletsPerHandler;
            NEW (cargoQueue);

            WAIT DURATION (plane.getLocation*0.5);
            END WAIT;

            WAIT FOR loadingPriority TO Give (SELF,1);
            END WAIT;

            WHILE (plane.getPalletCargo > 0) AND (temp > 0.0)
                ASK plane TO decPalletCargo;

                WAIT DURATION (cargoTimeToUnloadPallet);
                END WAIT;

                NEW (cargo);
                ASK cargo TO setCargoDestination;
                ASK cargo TO setCargoType(1);

                ASK cargoQueue TO Add (cargo);
                temp := temp - 1.0;
            END WHILE;

            ASK loadingPriority TO TakeBack (SELF, 1);

            WAIT DURATION (plane.getLocation*0.5);
            END WAIT;

            WAIT FOR airfield.forklifts TO Give
(SELF,cargoQueue.numberIn);
                ASK mainMenu TO updateForklifts;
                WAIT DURATION (cargoTimeToUnloadPallet);
                END WAIT;

                FOREACH cargo IN cargoQueue
```

```
                        INC(totalPalletCargo);
                        ASK cargo TO incrementCargoOnDock;
                        TELL airfield TO manageLoadingDock(cargo);
                        ASK airfield.forklifts TO TakeBack (SELF,1);
                        ASK mainMenu TO updateForklifts;
                        ASK cargoQueue TO RemoveThis (cargo);
                    END FOREACH;
                END WAIT;

                ASK team TO TakeBack (plane,1);
                ASK mainMenu TO updateHandlers;

            END METHOD;

        END OBJECT;

    END MODULE.
```

Global Modules (Definition and Implementation)

```
DEFINITION MODULE globalMod;

     FROM airfieldMod      IMPORT airfieldObj;
     FROM cargoMod         IMPORT cargoObj;
     FROM RandMod          IMPORT RandomObj;
     FROM Chart            IMPORT ChartObj;
     FROM Graphic          IMPORT GraphicLibObj;
     FROM Graph            IMPORT IDataPt, RDataPt;
     FROM StatMod          IMPORT ITimedStatObj;
     FROM Window           IMPORT WindowObj;
     FROM graphicsMod      IMPORT mainMenuObj;
     FROM IOMod            IMPORT  StreamObj;

     VAR
          airfield         :  airfieldObj;
          stream1          :  RandomObj;
          stream2          :  RandomObj;
          stream3          :  RandomObj;
          cargo            :  cargoObj;
          outputStream     :  StreamObj;
          inputStream      :  StreamObj;

          chart1,
          chart2,
          chart3,
          chart4           :  ChartObj;
          library          :  GraphicLibObj;
          window           :  WindowObj;
          mainMenu         :  mainMenuObj;

          totalArrivals    :  INTEGER;
          totalDepartures  :  INTEGER;
          totalDiverts     :  INTEGER;
          totalBreakdowns  :  INTEGER;
          currentOnGround  :  INTEGER;
          numInQueue       :  INTEGER;
          maxWaitingTime   :  REAL;
          totalWaitingTime :  REAL;
          totalWaits       :  INTEGER;
          totalPalletCargo :  INTEGER;
          totalRollingCargo :  INTEGER;
          totalPassengers  :  INTEGER;
          cargoCounter     :  INTEGER;

          palletsHeadedToA    :  INTEGER;
          palletsHeadedToB    :  INTEGER;
          palletsHeadedToC    :  INTEGER;
          palletsHeadedToD    :  INTEGER;

          rollingHeadedToA,
          rollingHeadedToB,
          rollingHeadedToC,
```

```
rollingHeadedToD          :   INTEGER;

passengersHeadedToA,
passengersHeadedToB,
passengersHeadedToC,
passengersHeadedToD       :   INTEGER;

palletsOnDock             :   IDataPt;
rollingOnDock             :   IDataPt;
passengersOnDock          :   IDataPt;

mog                 :   RDataPt;
service             :   IDataPt;
timeAvg             :   LMONITORED INTEGER;
timeAvgStats        :   ITimedStatObj;

cargoAtDestA, rollingAtDestA, passengersAtDestA :   INTEGER;
cargoAtDestB, rollingAtDestB, passengersAtDestB :   INTEGER;
cargoAtDestC, rollingAtDestC, passengersAtDestC :   INTEGER;
cargoAtDestD, rollingAtDestD, passengersAtDestD :   INTEGER;

totalAtDestA,
totalAtDestB,
totalAtDestC,
totalAtDestD              :   INTEGER;

numberOfLoadingDocks      :   REAL;      {airfield parameters}
numberOfTrucks            :   REAL;
numberOfCargoHandlers     :   REAL;
numberOfForklifts         :   REAL;
numberOfFuelPits          :   REAL;
numberOfFuelTrucks        :   REAL;
runLength                 :   REAL;
timeScale                 :   REAL;
meanInterArrTime          :   REAL;
maxOnGround               :   REAL;

cargoProbToA,                            {cargo parameters   }
cargoProbToB,
cargoProbToC,
cargoProbToD,
cargoTimeToUnloadPallet :   REAL;
cargoPalletsPerHandler  :   REAL;
fuelRatePit,
fuelRateTruck,
fuelTruckCapacity         :   REAL;

netDistanceToA,                          {network parameters   }
netDistanceToB,
netDistanceToC,
netDistanceToD,
palletPerTruck,
rollingPerTruck,
passengerPerTruck         :   REAL;

incLDocks                 :   REAL;
decLDocks                 :   REAL;
```

```
        incTrucks                :   REAL;
        decTrucks                :   REAL;
        incHandlers,
        decHandlers              :   REAL;
        incForklifts,
        decForklifts             :   REAL;
        incFuelTrucks,
        decFuelTrucks            :   REAL;

END MODULE.

IMPLEMENTATION MODULE globalMod;

END MODULE.
```

Graphics Modules (Definition and Implementation)

```
DEFINITION MODULE graphicsMod;

FROM Animate     IMPORT DynDClockObj;
FROM Form        IMPORT DialogBoxObj;
FROM Button      IMPORT ButtonObj;
FROM Value       IMPORT ValueBoxObj;
FROM Check       IMPORT CheckBoxObj;
FROM Image       IMPORT ImageObj;
FROM GTypes      IMPORT ALL ColorType;
FROM Menu        IMPORT MenuBarObj, MenuItemObj;
FROM Meter       IMPORT DigitalDisplayObj;
FROM Graph       IMPORT RDataPtMObj, IDataPtMObj;


FROM SimMod      IMPORT StartSimulation,
                        StopSimulation,
                        SimTime,
                        Timescale;


FROM Dynamic     IMPORT RealTimeAnimation;

FROM globalMod   IMPORT totalArrivals,     {pulling in global variables  }
                        totalDepartures,   {will allow menu driven       }
                        totalDiverts,      {displays to show statistics  }
                        numInQueue,        {and parameters held globally }
                        totalWaitingTime,
                        totalWaits,
                        totalBreakdowns,

                        totalPalletCargo,
                        totalRollingCargo,
                        totalPassengers,

                        cargoAtDestA, rollingAtDestA, passengersAtDestA,
                        cargoAtDestB, rollingAtDestB, passengersAtDestB,
                        cargoAtDestC, rollingAtDestC, passengersAtDestC,
                        cargoAtDestD, rollingAtDestD, passengersAtDestD,

                        totalAtDestA,
                        totalAtDestB,
                        totalAtDestC,
                        totalAtDestD,

                        runLength,                {airfield parameters}
                        timeScale,
                        meanInterArrTime,
                        numberOfLoadingDocks,
                        numberOfTrucks,
                        numberOfCargoHandlers,
                        numberOfForklifts,
                        numberOfFuelPits,
                        numberOfFuelTrucks,
                        maxOnGround,
```

```
                    cargoProbToA,              {cargo parameters   }
                    cargoProbToB,
                    cargoProbToC,
                    cargoProbToD,
                    cargoTimeToUnloadPallet,
                    cargoPalletsPerHandler,
                    fuelRatePit,
                    fuelRateTruck,
                    fuelTruckCapacity,

                    netDistanceToA,            {network parameters   }
                    netDistanceToB,
                    netDistanceToC,
                    netDistanceToD,
                    palletPerTruck,
                    rollingPerTruck,
                    passengerPerTruck,

                    incLDocks,
                    decLDocks,
                    incTrucks,
                    decTrucks,
                    incHandlers,
                    decHandlers,
                    incForklifts,
                    decForklifts,
                    incFuelTrucks,
                    decFuelTrucks,

                    airfield,
                    mog,
                    service,
                    timeAvgStats,
                    outputStream,

                    chart1,
                    chart2,
                    chart3,
                    chart4,
                    library,
                    window;

    TYPE
        mainMenuObj = OBJECT (MenuBarObj)

        dialogBox    : DialogBoxObj;
        button       : ButtonObj;
        startItem,
        stopItem     : MenuItemObj;
        divertMeter,
        queueMeter,
        trucksAvailMeter,
        handlerMeter,
        forkliftMeter,
        fuelTruckMeter        : DigitalDisplayObj;
```

104

```
        ASK METHOD showStatistics;
        ASK METHOD changeAirfieldParameters;
        ASK METHOD aircraftParameters(IN type : INTEGER);
        ASK METHOD changeCargoParameters;
        ASK METHOD changeNetworkParameters;
        ASK METHOD showDestinations;
        ASK METHOD setUpSimBoard;
        ASK METHOD updateDiverts;
        ASK METHOD updateQueue;
        ASK METHOD updateTrucks;
        ASK METHOD updateHandlers;
        ASK METHOD updateForklifts;
        ASK METHOD updateFuelTrucks;
    OVERRIDE
        ASK METHOD BeSelected;
    END OBJECT;

END MODULE.
```

```
IMPLEMENTATION MODULE graphicsMod;

    OBJECT mainMenuObj;

        ASK METHOD BeSelected;

        VAR
            picture : ImageObj;

        BEGIN
            CASE ASK LastPicked Id
                WHEN 1:
                    ASK startItem TO Deactivate;
                    ASK stopItem TO Activate;
                    StartSimulation;
                WHEN 2:
                    StopSimulation;
                    showStatistics;
                    ASK outputStream TO Close;
                    DISPOSE (outputStream);
                    HALT;
                WHEN 3:
                    showStatistics;
                WHEN 4:
                    showDestinations;
                WHEN 10:
                    changeAirfieldParameters;
                WHEN 11:
                    aircraftParameters(1);    {C-17}
                WHEN 12:
                    changeCargoParameters;
                WHEN 13:
                    changeNetworkParameters;
                WHEN 14:
                    aircraftParameters(2);   {C-5}
                WHEN 99:
                    NEW (dialogBox);
                    ASK dialogBox TO LoadFromLibrary (library,
"AboutBox");
                    ASK window TO AddGraphic (dialogBox);
                    ASK dialogBox TO Draw;

                    NEW (picture);
                    ASK picture TO LoadFromLibrary (library, "bitmap");
                    ASK window TO AddGraphic (picture);
                    ASK picture TO Draw;

                    button := ASK dialogBox TO AcceptInput();
                    DISPOSE (dialogBox);
                    DISPOSE (picture);
                OTHERWISE;
            END CASE;
        END METHOD;

        ASK METHOD changeAirfieldParameters;

        VAR
```

106

```
                valStop,
                valScale,
                valMeanTime,
                valNumDocks,
                valNumTrucks,
                valMOG,
                valNumHandlers,
                valNumForklifts,
                valNumFuelPits,
                valNumFuelTrucks    :    ValueBoxObj;
                resetStats          :    CheckBoxObj;
                tempLDocks,
                tempTrucks,
                tempHandlers,
                tempForklifts,
                tempFuelTrucks      :    REAL;

         BEGIN
                NEW (dialogBox);
                ASK dialogBox TO LoadFromLibrary(library, "ParmBox");
                ASK window TO AddGraphic (dialogBox);

                valStop := ASK dialogBox Child ("stopTime", 1);
                ASK valStop TO SetValue (runLength);

                valScale := ASK dialogBox Child ("timeScale", 2);
                ASK valScale TO SetValue (timeScale);

                valMeanTime := ASK dialogBox Child ("mean", 3);
                ASK valMeanTime TO SetValue (meanInterArrTime);

                valNumDocks := ASK dialogBox Child ("numLDocks", 4);
                ASK valNumDocks TO SetValue (numberOfLoadingDocks);

                valNumTrucks := ASK dialogBox Child ("numTrucks", 5);
                ASK valNumTrucks TO SetValue (numberOfTrucks);

                valMOG := ASK dialogBox Child ("maxOnGround", 6);
                ASK valMOG TO SetValue (maxOnGround);

                valNumHandlers := ASK dialogBox Child ("numHandlers", 7);
                ASK valNumHandlers TO SetValue (numberOfCargoHandlers);

                valNumForklifts := ASK dialogBox Child ("numForklifts", 8);
                ASK valNumForklifts TO SetValue (numberOfForklifts);

                valNumFuelPits := ASK dialogBox Child ("numFuelPits", 9);
                ASK valNumFuelPits TO SetValue (numberOfFuelPits);

                valNumFuelTrucks := ASK dialogBox Child ("numFuelTrucks",
         10);
                ASK valNumFuelTrucks TO SetValue (numberOfFuelTrucks);

                resetStats := ASK dialogBox Child ("resetStats", 150);
                ASK resetStats TO SetCheck (FALSE);

                button := ASK dialogBox TO AcceptInput();
```

107

```
IF ASK button ReferenceName = "Ok"

        runLength := ASK valStop Value();
        timeScale := ASK valScale Value();
        Timescale := 60.0/timeScale;
        meanInterArrTime := ASK valMeanTime Value();
        maxOnGround := ASK valMOG Value();
        numberOfFuelPits := ASK valNumFuelPits Value();

        tempLDocks := ASK valNumDocks Value();
        IF tempLDocks < numberOfLoadingDocks
            decLDocks := numberOfLoadingDocks - tempLDocks;
            TELL airfield TO decrementLDocks(decLDocks);
            TELL airfield TO
decrementParkingSpace(TRUNC(decLDocks));
            numberOfLoadingDocks := tempLDocks;
        ELSIF tempLDocks > numberOfLoadingDocks
            incLDocks := tempLDocks - numberOfLoadingDocks;
            ASK airfield TO incrementLDocks(incLDocks);
            ASK airfield TO
incrementParkingSpace(TRUNC(incLDocks));
            numberOfLoadingDocks := tempLDocks;
        ELSE numberOfLoadingDocks := tempLDocks;
        END IF;

        tempTrucks := ASK valNumTrucks Value();
        IF tempTrucks < numberOfTrucks
            decTrucks := numberOfTrucks - tempTrucks;
            TELL airfield TO decrementTrucks(decTrucks);
            numberOfTrucks := tempTrucks;
            updateTrucks;
        ELSIF tempTrucks > numberOfTrucks
            incTrucks := tempTrucks - numberOfTrucks;
            ASK airfield TO incrementTrucks(incTrucks);
            numberOfTrucks := tempTrucks;
            updateTrucks;
        ELSE numberOfTrucks := tempTrucks;
        END IF;

        tempHandlers := ASK valNumHandlers Value();
        IF tempHandlers < numberOfCargoHandlers
            decHandlers := numberOfCargoHandlers -
tempHandlers;
            TELL airfield TO decrementHandlers(decHandlers);
            numberOfCargoHandlers := tempHandlers;
            updateHandlers;
        ELSIF tempHandlers > numberOfCargoHandlers
            incHandlers := tempHandlers -
numberOfCargoHandlers;
            ASK airfield TO incrementHandlers(incHandlers);
            numberOfCargoHandlers := tempHandlers;
            updateHandlers;
        ELSE numberOfCargoHandlers := tempHandlers;
        END IF;

        tempForklifts := ASK valNumForklifts Value();
```

```
            IF tempForklifts < numberOfForklifts
                decForklifts := numberOfForklifts - tempForklifts;
                TELL airfield TO decrementForklifts(decForklifts);
                numberOfForklifts := tempForklifts;
                updateForklifts;
            ELSIF tempForklifts > numberOfForklifts
                incForklifts := tempForklifts - numberOfForklifts;
                ASK airfield TO incrementForklifts(incForklifts);
                numberOfForklifts := tempForklifts;
                updateForklifts;
            ELSE numberOfForklifts := tempForklifts;
            END IF;

            tempFuelTrucks := ASK valNumFuelTrucks Value();
            IF tempFuelTrucks < numberOfFuelTrucks
                decFuelTrucks := numberOfFuelTrucks -
tempFuelTrucks;
                TELL airfield TO
decrementFuelTrucks(decFuelTrucks);
                numberOfFuelTrucks := tempFuelTrucks;
                updateFuelTrucks;
            ELSIF tempFuelTrucks > numberOfFuelTrucks
                incFuelTrucks := tempFuelTrucks -
numberOfFuelTrucks;
                ASK airfield TO incrementFuelTrucks(incFuelTrucks);
                numberOfFuelTrucks := tempFuelTrucks;
                updateFuelTrucks;
            ELSE numberOfFuelTrucks := tempFuelTrucks;
            END IF;

            IF resetStats.Checked = TRUE
                ASK airfield.loadingDock TO ResetAllocationStats;
                ASK airfield.loadingDock TO ResetPendingStats;
                ASK airfield.motorPool TO ResetAllocationStats;
                ASK airfield.cargoHandler TO ResetAllocationStats;
                ASK airfield.forklifts  TO ResetAllocationStats;
                ASK airfield.fuelTrucks TO ResetAllocationStats;
            ELSE
            END IF;

            DISPOSE (dialogBox);

        ELSE
            DISPOSE (dialogBox);
        END IF;

    END METHOD;

    ASK METHOD showStatistics;

    VAR
        valBox  :   ValueBoxObj;

    BEGIN
        NEW (dialogBox);
        ASK dialogBox TO LoadFromLibrary (library, "StatsBox");
        ASK window TO AddGraphic (dialogBox);
```

```
        valBox := ASK dialogBox Child ("numLand", 1);
        ASK valBox TO SetValue (FLOAT (totalArrivals));

        valBox := ASK dialogBox Child ("numDepart", 2);
        ASK valBox TO SetValue (FLOAT (totalDepartures));

        valBox := ASK dialogBox Child ("numDivert", 3);
        ASK valBox TO SetValue (FLOAT (totalDiverts));

        valBox := ASK dialogBox Child ("numBreakdown", 17);
        ASK valBox TO SetValue (FLOAT (totalBreakdowns));

        valBox := ASK dialogBox Child ("utilLD", 4);
        ASK valBox TO SetValue
((airfield.loadingDock.AllocWtdMean()/numberOfLoadingDocks)*100.0);

        valBox := ASK dialogBox Child ("utilMP", 5);
        ASK valBox TO SetValue
((airfield.motorPool.AllocWtdMean()/numberOfTrucks)*100.0);

        valBox := ASK dialogBox Child ("utilCH", 16);
        ASK valBox TO SetValue
((airfield.cargoHandler.AllocWtdMean()/numberOfCargoHandlers)*100.0);

        valBox := ASK dialogBox Child ("utilFork", 12);
        ASK valBox TO SetValue
((airfield.forklifts.AllocWtdMean()/numberOfForklifts)*100.0);

        valBox := ASK dialogBox Child ("utilFuel", 13);
        ASK valBox TO SetValue
((airfield.fuelTrucks.AllocWtdMean()/numberOfFuelTrucks)*100.0);

        valBox := ASK dialogBox Child ("lengthQ", 6);
        ASK valBox TO SetValue
(airfield.loadingDock.PendWtdMean());

        valBox := ASK dialogBox Child ("maxLengthQ", 7);
        ASK valBox TO SetValue
(FLOAT(airfield.loadingDock.PendingMaximum()));

        valBox := ASK dialogBox Child ("meanWaitQ", 8);
        ASK valBox TO SetValue
(totalWaitingTime/FLOAT(totalWaits));

        valBox := ASK dialogBox Child ("totPCargo", 9);
        ASK valBox TO SetValue (FLOAT (totalPalletCargo));

        valBox := ASK dialogBox Child ("totRCargo", 10);
        ASK valBox TO SetValue (FLOAT (totalRollingCargo));

        valBox := ASK dialogBox Child ("totalPass", 11);
        ASK valBox TO SetValue (FLOAT (totalPassengers));

        valBox := ASK dialogBox Child ("estAirfieldCap", 18);
        ASK valBox TO SetValue
(FLOAT(totalDepartures)*(1440.0/SimTime));
```

```
            valBox := ASK dialogBox Child ("serviceMOG", 19);
            ASK valBox TO SetValue (timeAvgStats.Mean);

            valBox := ASK dialogBox Child ("simTime", 20);
            ASK valBox TO SetValue (SimTime/60.0);

            ASK dialogBox TO Draw;

            button := ASK dialogBox TO AcceptInput ();
            DISPOSE (dialogBox);

        END METHOD;

        ASK METHOD showDestinations;

        VAR
            valBox  :    ValueBoxObj;

        BEGIN
            NEW(dialogBox);
            ASK dialogBox TO LoadFromLibrary (library,
"destinationBox");
            ASK window TO AddGraphic (dialogBox);

            valBox := ASK dialogBox Child ("bulk1", 1);
            ASK valBox TO SetValue (FLOAT (cargoAtDestA));

            valBox := ASK dialogBox Child ("bulk2", 2);
            ASK valBox TO SetValue (FLOAT (cargoAtDestB));

            valBox := ASK dialogBox Child ("bulk3", 3);
            ASK valBox TO SetValue (FLOAT (cargoAtDestC));

            valBox := ASK dialogBox Child ("bulk4", 4);
            ASK valBox TO SetValue (FLOAT (cargoAtDestD));

            valBox := ASK dialogBox Child ("rolling1", 5);
            ASK valBox TO SetValue (FLOAT (rollingAtDestA));

            valBox := ASK dialogBox Child ("rolling2", 6);
            ASK valBox TO SetValue (FLOAT (rollingAtDestB));

            valBox := ASK dialogBox Child ("rolling3", 7);
            ASK valBox TO SetValue (FLOAT (rollingAtDestC));

            valBox := ASK dialogBox Child ("rolling4", 8);
            ASK valBox TO SetValue (FLOAT (rollingAtDestD));

            valBox := ASK dialogBox Child ("pass1", 9);
            ASK valBox TO SetValue (FLOAT (passengersAtDestA));

            valBox := ASK dialogBox Child ("pass2", 10);
            ASK valBox TO SetValue (FLOAT (passengersAtDestB));

            valBox := ASK dialogBox Child ("pass3", 11);
            ASK valBox TO SetValue (FLOAT (passengersAtDestC));
```

```
        valBox := ASK dialogBox Child ("pass4", 12);
        ASK valBox TO SetValue (FLOAT (passengersAtDestD));

        valBox := ASK dialogBox Child ("totalAtA", 13);
        ASK valBox TO SetValue (FLOAT (totalAtDestA));

        valBox := ASK dialogBox Child ("totalAtB", 14);
        ASK valBox TO SetValue (FLOAT (totalAtDestB));

        valBox := ASK dialogBox Child ("totalAtC", 15);
        ASK valBox TO SetValue (FLOAT (totalAtDestC));

        valBox := ASK dialogBox Child ("totalAtD", 16);
        ASK valBox TO SetValue (FLOAT (totalAtDestD));

        ASK dialogBox TO Draw;

        button := ASK dialogBox TO AcceptInput ();
        DISPOSE (dialogBox);

END METHOD;

ASK METHOD aircraftParameters(IN type : INTEGER);

VAR
        valPallet,
        valRolling,
        valPass,
        valFuelCap,
        valFuelRate,
        valFuelNum,
        valConProb,
        valConDur,
        valNonProb,
        valNonDur         :   ValueBoxObj;

BEGIN
        NEW (dialogBox);
        ASK dialogBox TO LoadFromLibrary(library, "acParamBox");

        IF type = 1
            ASK dialogBox TO SetLabel ("C-17 Parameters");
        ELSE
            ASK dialogBox TO SetLabel ("C-5 Parameters");
        END IF;

        ASK window TO AddGraphic (dialogBox);

        valPallet := ASK dialogBox Child ("amtPallet", 1);
        valRolling := ASK dialogBox Child ("amtRolling", 2);
        valPass := ASK dialogBox Child ("amtPass", 3);
        valFuelCap := ASK dialogBox Child ("fuelCap", 4);
        valFuelRate := ASK dialogBox Child ("fuelRate", 5);
        valFuelNum := ASK dialogBox Child ("fuelNum", 6);
        valConProb := ASK dialogBox Child ("cProb", 7);
        valConDur := ASK dialogBox Child ("cDur", 8);
```

112

```
        valNonProb := ASK dialogBox Child ("nProb", 9);
        valNonDur := ASK dialogBox Child ("nDur", 10);

        CASE type
            WHEN 1:
                ASK valPallet TO SetValue (10.0);
                ASK valRolling TO SetValue (2.0);
                ASK valPass TO SetValue (40.0);
                ASK valFuelCap TO SetValue (22400.0);
                ASK valFuelRate TO SetValue (850.0);
                ASK valFuelNum TO SetValue (2.0);
                ASK valConProb TO SetValue (0.075);
                ASK valConDur TO SetValue (60.0);
                ASK valNonProb TO SetValue (.025);
                ASK valNonDur TO SetValue (60.0);
            WHEN 2:
                ASK valPallet TO SetValue (30.0);
                ASK valRolling TO SetValue (6.0);
                ASK valPass TO SetValue (60.0);
                ASK valFuelCap TO SetValue (49500.0);
                ASK valFuelRate TO SetValue (850.0);
                ASK valFuelNum TO SetValue (2.0);
                ASK valConProb TO SetValue (0.3);
                ASK valConDur TO SetValue (100.0);
                ASK valNonProb TO SetValue (.1);
                ASK valNonDur TO SetValue (100.0);
        END CASE;

        button := ASK dialogBox TO AcceptInput ();

        DISPOSE (dialogBox);

END METHOD;

ASK METHOD changeCargoParameters;

VAR
    valToA,
    valToB,
    valToC,
    valToD,
    valTimeUnload,
    valPallets,
    valFuelRatePit,
    valFuelRateTruck,
    valFuelCapTruck     :   ValueBoxObj;

BEGIN
    NEW (dialogBox);
    ASK dialogBox TO LoadFromLibrary(library, "cargoParamBox");
    ASK window TO AddGraphic (dialogBox);

    valToA := ASK dialogBox Child ("cargoDistA", 1);
    ASK valToA TO SetValue (cargoProbToA);

    valToB := ASK dialogBox Child ("cargoDistB", 2);
    ASK valToB TO SetValue (cargoProbToB);
```

113

```
            valToC := ASK dialogBox Child ("cargoDistC", 3);
            ASK valToC TO SetValue (cargoProbToC);

            valToD := ASK dialogBox Child ("cargoDistD", 4);
            ASK valToD TO SetValue (cargoProbToD);

            valTimeUnload := ASK dialogBox Child ("timeUnloadPallet",
5);

            ASK valTimeUnload TO SetValue (cargoTimeToUnloadPallet);

            valPallets := ASK dialogBox Child ("palletPerHandler", 6);
            ASK valPallets TO SetValue (cargoPalletsPerHandler);

            valFuelRatePit := ASK dialogBox Child ("fuelRatePit", 7);
            ASK valFuelRatePit TO SetValue (fuelRatePit);

            valFuelRateTruck := ASK dialogBox Child ("fuelRateTruck",
8);

            ASK valFuelRateTruck TO SetValue (fuelRateTruck);

            valFuelCapTruck  := ASK dialogBox Child ("fuelCapTruck",
9);

            ASK valFuelCapTruck TO SetValue (fuelTruckCapacity);

            button := ASK dialogBox TO AcceptInput ();

            IF ASK button ReferenceName = "ok"

                cargoProbToA := ASK valToA Value();
                cargoProbToB := ASK valToB Value();
                cargoProbToC := ASK valToC Value();
                cargoProbToD := ASK valToD Value();
                cargoTimeToUnloadPallet := ASK valTimeUnload Value();
                cargoPalletsPerHandler := ASK valPallets Value();
                fuelRatePit := ASK valFuelRatePit Value();
                fuelRateTruck := ASK valFuelRateTruck Value();
                fuelTruckCapacity := ASK valFuelCapTruck Value();

                DISPOSE (dialogBox);
            ELSE
                DISPOSE (dialogBox);
            END IF;

        END METHOD;

        ASK METHOD changeNetworkParameters;

        VAR
            valToA,
            valToB,
            valToC,
            valToD,
            valPalletPerTruck,
            valRollingPerTruck,
            valPassPerTruck       :   ValueBoxObj;
```

114

```
        BEGIN
                NEW (dialogBox);
                ASK dialogBox TO LoadFromLibrary(library,
"networkParamBox");
                ASK window TO AddGraphic (dialogBox);

                valToA := ASK dialogBox Child ("distToA", 1);
                ASK valToA TO SetValue (netDistanceToA);

                valToB := ASK dialogBox Child ("distToB", 2);
                ASK valToB TO SetValue (netDistanceToB);

                valToC := ASK dialogBox Child ("distToC", 3);
                ASK valToC TO SetValue (netDistanceToC);

                valToD := ASK dialogBox Child ("distToD", 4);
                ASK valToD TO SetValue (netDistanceToD);

                valPalletPerTruck := ASK dialogBox Child ("palletPerTruck",
5);
                ASK valPalletPerTruck TO SetValue (palletPerTruck);

                valRollingPerTruck := ASK dialogBox Child
("rollingPerTruck", 6);
                ASK valRollingPerTruck TO SetValue (rollingPerTruck);

                valPassPerTruck := ASK dialogBox Child ("passPerTruck", 7);
                ASK valPassPerTruck TO SetValue (passengerPerTruck);

                button := ASK dialogBox TO AcceptInput ();

                IF ASK button ReferenceName = "ok"

                        netDistanceToA := ASK valToA Value();
                        netDistanceToB := ASK valToB Value();
                        netDistanceToC := ASK valToC Value();
                        netDistanceToD := ASK valToD Value();
                        palletPerTruck := ASK valPalletPerTruck Value();
                        rollingPerTruck := ASK valRollingPerTruck Value();
                        passengerPerTruck := ASK valPassPerTruck Value();

                        DISPOSE (dialogBox);
                ELSE
                        DISPOSE (dialogBox);
                END IF;

        END METHOD;

        ASK METHOD setUpSimBoard;

        VAR
                airfield    : ImageObj;
                clock       : DynDClockObj;

        BEGIN

                NEW (window);
```

```
        ASK window TO SetTitle ("Aerial Port Operations Model");
        ASK window TO SetSize(100.0,100.0);
        ASK window TO ShowWorld(0.0,0.0,10.0,10.0);
        ASK window TO SetColor(ForestGreen);

        ASK window TO Draw;

        NEW (library);
        ASK library TO ReadFromFile ("graphics.sg2");

        NEW (airfield);
        ASK airfield TO LoadFromLibrary (library, "airfield");
        ASK window TO AddGraphic (airfield);

        NEW (clock);
        ASK clock TO LoadFromLibrary (library, "clock");
        ASK window TO AddGraphic (clock);
        ASK clock TO SetTimeScale (1.0/60.0);
        ASK clock TO SetTime (0,0,0);
        Timescale := 60.0/timeScale;
        ASK clock TO StartMotion;

        ASK SELF TO LoadFromLibrary (library, "menubar");
        ASK window TO AddGraphic (SELF);
        startItem := ASK SELF Descendant ("Start", 1);
        stopItem := ASK SELF Descendant ("Stop", 2);

        NEW (chart1);
        ASK chart1 TO LoadFromLibrary (library, "ACTimeChart");
        ASK window TO AddGraphic (chart1);

        NEW (chart2);
        ASK chart2 TO LoadFromLibrary (library, "mogChart");
        ASK window TO AddGraphic (chart2);
        ASK GETMONITOR (service, IDataPtMObj) TO SetGraph(chart2);

        NEW (chart3);
        ASK chart3 TO LoadFromLibrary (library, "LDockChart");
        ASK window TO AddGraphic (chart3);

        NEW (chart4);
        ASK chart4 TO LoadFromLibrary (library, "capChart");
        ASK window TO AddGraphic (chart4);
        ASK GETMONITOR (mog, RDataPtMObj) TO SetGraph(chart4);

        NEW (divertMeter);
        ASK divertMeter TO LoadFromLibrary (library,
"divertMeter");
        ASK window TO AddGraphic (divertMeter);

        NEW (queueMeter);
        ASK queueMeter TO LoadFromLibrary (library, "queueMeter");
        ASK window TO AddGraphic (queueMeter);

        NEW (trucksAvailMeter);
        ASK trucksAvailMeter TO LoadFromLibrary (library,
"truckMeter");
```

116

```
                ASK window TO AddGraphic (trucksAvailMeter);

                NEW (handlerMeter);
                ASK handlerMeter TO LoadFromLibrary (library,
"handlerMeter");
                ASK window TO AddGraphic (handlerMeter);

                NEW (forkliftMeter);
                ASK forkliftMeter TO LoadFromLibrary (library,
"forkliftMeter");
                ASK window TO AddGraphic (forkliftMeter);

                NEW (fuelTruckMeter);
                ASK fuelTruckMeter TO LoadFromLibrary (library,
"fuelTruckMeter");
                ASK window TO AddGraphic (fuelTruckMeter);

                ASK SELF TO Draw;
                ASK airfield TO Draw;
                ASK clock TO Draw;
                ASK chart1 TO Draw;
                ASK chart2 TO Draw;
                ASK chart3 TO Draw;
                ASK chart4 TO Draw;
                ASK divertMeter TO Draw;
                ASK queueMeter TO Draw;
                ASK trucksAvailMeter TO Draw;
                ASK handlerMeter TO Draw;
                ASK forkliftMeter TO Draw;
                ASK fuelTruckMeter TO Draw;

                RealTimeAnimation := TRUE;

        END METHOD;

        ASK METHOD updateDiverts;

        BEGIN
                INC (totalDiverts);
                ASK divertMeter TO DisplayValue (FLOAT(totalDiverts));
        END METHOD;

        ASK METHOD updateQueue;

        BEGIN
                ASK queueMeter TO DisplayValue (FLOAT(numInQueue));
        END METHOD;

        ASK METHOD updateTrucks;

        BEGIN
                ASK trucksAvailMeter TO DisplayValue
(FLOAT(airfield.motorPool.Resources));
        END METHOD;

        ASK METHOD updateHandlers;
```

```
        BEGIN
            ASK handlerMeter TO DisplayValue
(FLOAT(airfield.cargoHandler.Resources));
        END METHOD;

        ASK METHOD updateForklifts;

        BEGIN
            ASK forkliftMeter TO DisplayValue
(FLOAT(airfield.forklifts.Resources));
        END METHOD;

        ASK METHOD updateFuelTrucks;

        BEGIN
            ASK fuelTruckMeter TO DisplayValue
(FLOAT(airfield.fuelTrucks.Resources));
        END METHOD;

    END OBJECT;

END MODULE.
```

Main Module

```
MAIN MODULE final;

FROM IOMod          IMPORT  ALL FileUseType;
FROM RandMod        IMPORT  FetchSeed;
FROM Menu           IMPORT  MenuItemObj;
FROM globalMod      IMPORT  runLength,              {airfield
parameters}

                            timeScale,
                            meanInterArrTime,
                            numberOfLoadingDocks,
                            numberOfTrucks,
                            numberOfCargoHandlers,
                            numberOfForklifts,
                            numberOfFuelPits,
                            numberOfFuelTrucks,
                            maxOnGround,

                            cargoProbToA,           {cargo parameters}
                            cargoProbToB,
                            cargoProbToC,
                            cargoProbToD,
                            cargoTimeToUnloadPallet,
                            cargoPalletsPerHandler,
                            fuelRatePit,
                            fuelRateTruck,
                            fuelTruckCapacity,

                            netDistanceToA,         {network parameters}
                            netDistanceToB,
                            netDistanceToC,
                            netDistanceToD,
                            palletPerTruck,
                            rollingPerTruck,
                            passengerPerTruck,

                            mog,
                            outputStream,
                            inputStream,

                            mainMenu,
                            airfield,

                            stream1,
                            stream2,
                            stream3;

VAR
    item      :   MenuItemObj;
    parameter :   REAL;

BEGIN
```

```
    NEW (inputStream);                                  {open IO stream to
read}
    ASK inputStream TO Open ("input.txt", Input); {initial parameters
}
                                                        {from input.txt file
}
    ASK inputStream TO ReadReal (parameter);
    runLength := parameter;
    ASK inputStream TO ReadReal (parameter);
    timeScale := parameter;
    ASK inputStream TO ReadReal (parameter);
    meanInterArrTime := parameter;
    ASK inputStream TO ReadReal (parameter);          {sim/airfield
parameters}
    numberOfLoadingDocks := parameter;
    ASK inputStream TO ReadReal (parameter);
    numberOfTrucks := parameter;
    ASK inputStream TO ReadReal (parameter);
    numberOfCargoHandlers := parameter;
    ASK inputStream TO ReadReal (parameter);
    numberOfForklifts := parameter;
    ASK inputStream TO ReadReal (parameter);
    numberOfFuelPits := parameter;
    ASK inputStream TO ReadReal (parameter);
    numberOfFuelTrucks := parameter;
    ASK inputStream TO ReadReal (parameter);
    maxOnGround := parameter;
    ASK inputStream TO ReadReal (parameter);
    mog := parameter;

    ASK inputStream TO ReadReal (parameter);
    cargoProbToA := parameter;
    ASK inputStream TO ReadReal (parameter);
    cargoProbToB := parameter;
    ASK inputStream TO ReadReal (parameter);          {cargo/fuel
parameters}
    cargoProbToC := parameter;
    ASK inputStream TO ReadReal (parameter);
    cargoProbToD := parameter;
    ASK inputStream TO ReadReal (parameter);
    cargoTimeToUnloadPallet := parameter;
    ASK inputStream TO ReadReal (parameter);
    cargoPalletsPerHandler := parameter;
    ASK inputStream TO ReadReal (parameter);
    fuelRatePit := parameter;
    ASK inputStream TO ReadReal (parameter);
    fuelRateTruck := parameter;
    ASK inputStream TO ReadReal (parameter);
    fuelTruckCapacity := parameter;

    ASK inputStream TO ReadReal (parameter);
    netDistanceToA := parameter;
    ASK inputStream TO ReadReal (parameter);
    netDistanceToB := parameter;
    ASK inputStream TO ReadReal (parameter);          {trans network   }
    netDistanceToC := parameter;                      {parameters      }
    ASK inputStream TO ReadReal (parameter);
```

```
    netDistanceToD := parameter;
    ASK inputStream TO ReadReal (parameter);
    palletPerTruck := parameter;
   'ASK inputStream TO ReadReal (parameter);
    rollingPerTruck := parameter;
    ASK inputStream TO ReadReal (parameter);
    passengerPerTruck := parameter;

    ASK inputStream TO Close;
    DISPOSE (inputStream);

    NEW (stream1);                          {rand num generators for  }
    NEW (stream2);                          {AC arrival, cargo amts,   }
    NEW (stream3);                          {and AC maintenance         }

    ASK stream2 TO SetSeed (FetchSeed (2));      {seed assignments   }
    ASK stream3 TO SetSeed (FetchSeed (3));

    NEW (outputStream);
    ASK outputStream TO Open ("output.txt", Output);

    NEW (airfield);
    TELL airfield TO generateArrivals;

    NEW (mainMenu);
    ASK mainMenu TO setUpSimBoard;

    REPEAT
        item := ASK mainMenu TO AcceptInput();
    UNTIL (ASK item ReferenceName = "Start");

END MODULE.
```

Bibliography

"A material handling solution for an air cargo facility," Industrial Engineering, 24-25 (November 1992).

"JFK airport cargo system will be U.S. first," Civil Engineering, 12-13 (May 1992).

Banks, Jerry, John S. Carson II, and Barry L. Nelson. Discrete-Event System Simulation (Second Edition). New Jersey: Prentice-Hall, 1996.

Battaglioli, Victor J. Throughput Capacity Estimation. Army Command and General Staff College, June 1975.

CACI Products Company. Modeling and Simulation with MODSIM III. Training Seminar at Arlington VA: CACI Products Co., 3-7 August 1998.

CACI Products Company. MODSIM III Reference Manual. LaJolla CA: CACI Products Co., August 1997.

CACI Products Company. SIMGRAPHICS II User's Manual. LaJolla CA: CACI Products Co., July 1995.

Cusick, Travis. Base Resource and Capability Estimator User's Manual and Model Code. Washington University, December 1997.

Department of the Air Force. Military Airlift Policy for Aerial Port Operations. AMC Policy Directive 24-1, April 1995.

Department of the Air Force. Military Airlift Transportation. AMC Instruction 24-101, Vol 1, May 1995.

Department of the Air Force. Military Airlift: Aerial Port Aircraft Loading Certification and Aircraft Load Planning and Loading Program. AMC Instruction 24-101, Vol 7, January 1996.

Department of the Air Force. Military Airlift: Aerial Port Mobility Units and Aerial Delivery Flights. AMC Instruction 24-101, Vol 18, May 1996.

Department of the Air Force. <u>Military Airlift: Air Terminal Operations Center</u>. AMC Instruction 24-101, Vol 9, August 1996.

Department of the Air Force. <u>Military Airlift: Cargo and Mail</u>. AMC Instruction 24-101, Vol 11, March 1996.

Department of the Army. <u>Strategic Mobility Sensitivity Analysis of Selected Alternatives Tactical Wheeled Vehicle Fleet Study</u>. Army Training and Doctrine Command, Feb 1981.

Gordon, Geoffrey. <u>System Simulation</u>. New Jersey: Prentice-Hall, 1969.

Johnson, Randall G. <u>A SLAM Airfield Model For Airlift Operations</u>. MS thesis, AFIT/GST/OS/84M-12. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, March 1984 .

Killingsworth, Paul S. and Laura Melody. <u>Should C-17s Be Used to Carry In-Theater Cargo During Major Deployments?</u>. RAND DB-171-AF/OSD, 1997.

Law, Averill M. and Michael G. McComas. "Secrets of Successful Simulation Studies," <u>Industrial Engineering</u>, 22:47-48, 51-53, 72 (May 1990).

Law, Averill M. and W. David Kelton. <u>Simulation Modeling and Analysis</u> (Second Edition). New York: McGraw-Hill, 1991.

Mattock, Michael G. and others. <u>New Capabilities for Strategic Mobility Analysis Using Mathematical Programming</u>. RAND MR-296-JS, 1995.

Mattock, Michael G. and others. <u>New Capabilities for Strategic Mobility Analysis: Executive Summary</u>. RAND MR-294-JS, 1994.

McCanne, Randy. <u>The Airlift Capabilities Estimation Prototype: A Case Study in Model Validation</u>. MS thesis, AFIT/GOR/ENS/93M-13. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, March 1993.

Morris, William T. "On the Art of Modeling," <u>Management Science</u>, 13:B707-B717 (August 1967).

Nickles, Keith E. <u>Global Reach and Air Cargo Operations: A Study in Materials Handling Equipment Requirements</u>. Graduate Research Paper, AFIT/GMO/LAL/96J-7. School of Logistics and Acquisition Management, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, May 1996.

Nobert, Yves and Jacques Roy. "Freight handling personnel scheduling at air cargo terminals," <u>Transportation Science</u>, 32:295-301 (August 1998).

Park, Chan S. and Yong Deok Noh. "A port simulation model for bulk cargo operations," <u>Simulation</u>, 236-246 (Jun 1987).

Post, David C. <u>Air Force Reserve Aerial Port Contingency Training.</u> Graduate Research Paper, AFIT/GMO/LAL/96N-12. School of Logistics and Acquisition Management, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, November 1996.

Pritsker, A. Alan B. and others. <u>Simulation with Visual SLAM and AweSim</u>. New York: John Wiley and Sons, 1997.

Schank, John and others. <u>A Review of Strategic Mobility Models and Analysis</u>. RAND R-3926-JS, 1991.

Shannon, Robert E. <u>Systems Simulation: the Art and Science</u>. New Jersey: Prentice-Hall, 1975.

Stucker, James P. and Ruth T. Berg. <u>Understanding Airfield Capacity for Airlift Operations.</u> RAND MR-700-AF/OSD, 1998.

Thompson, Tom E. <u>Mobility Requirements Studies: Time for a New Approach.</u> USAWC Strategy Research Project, Army War College, February 1997.

Williams, David. <u>Estimation of Airfield Capacity</u>. MS thesis, AFIT/GOA/ENS/99M-12. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, March 1999.

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE March 1999	3. REPORT TYPE AND DATES COVERED Master's Thesis	

4. TITLE AND SUBTITLE
MODELING AND ANALYSIS OF AERIAL PORT OPERATIONS

5. FUNDING NUMBERS

6. AUTHOR(S)
Timothy W. Albrecht, Captain, USAF

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Air Force Institute of Technology
WPAFB OH 45433-6583

8. PERFORMING ORGANIZATION REPORT NUMBER

AFIT/GOR/ENS/99M-1

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
HQ AMC/XPY
402 Scott Drive Unit 3L3
Scott AFB IL 62225-5307

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

The focus of this thesis effort is gaining useful insight into aerial port operations by employing an animated simulation. Understanding airfield capacity, resources, and functioning allows greater accuracy and efficiency in both planning for future force structures and matching mobility assets with commanders' objectives. Two current simulations, ACE (Airfield Capacity Estimator) and BRACE (Base Resource Allocation and Capabilities Estimator), model mobility activities at the base level with some deficiencies. The model proposed by this thesis, APOM (Aerial Port Operations Model), will provide the mobility analyst an animated simulation with two, new measures of aerial port operations; a real-time estimate of airfield capacity subject to changing levels of airfield resources, and an instantaneous count of serviced aircraft (service MOG). Additionally, APOM will offer an expanded utility to the mobility analyst by modeling a ground transportation network associated with the aerial port.

14. SUBJECT TERMS
Airlift Operations; Airlift; Simulation; Simulation Languages

15. NUMBER OF PAGES
133

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UL