

Design Notebook For Aircraft Repair Model Simulation

CS 422 Software Engineering Principles

November 8, 2000

Team X: OurColor:

Student Name

Version 1.5

Abstract

On September 9, 2000, Team OurColor was established by Dr. F. Sheldon, Manager, to develop an Aircraft Repair Model Simulation (ARMS) in support of the Earth Science Enterprise Program. The goal of Team OurColor is to provide ARMS as a fully verified and validated software package by December 11, 2000. ARMS will use a discrete, next-event time advance model to simulate the aircraft repair process. The purpose of ARMS is to compare various maintenance scheduling alternatives in order to optimize repair times and downtime costs. Specifically, seven types of commercial aircraft will be evaluated using three different scheduling techniques. The product will provide a graphical user interface, so the user can quickly and easily set parameters, run the simulation, compare scheduling alternatives, and view results. ARMS will be developed in Java using the Simlib library (provided by the customer) to perform standard simulation tasks. ARMS will also be developed to facilitate future extensibility of scheduling alternatives. (Correctness of data from individual Simlib function-calls and from user-provided aircraft parameters is assumed.)

The purpose of this document is to outline the design that Team OurColor has developed for the ARMS program described above. In compliance with IEEE standards, a function-oriented, Structured System Analysis and Structured Design (SSA/SD) methodology was used. This design methodology was selected to ensure that all customer requirements will be met and verified, and that a cohesive, loosely coupled application will be developed. Based on this design strategy, this document details the following information:

- Requirements summary
- Context diagram
- Data flow diagrams
- Process specifications (P-Specs)
- Structure chart
- Data dictionary
- Transaction analysis
- Transform analysis
- Test cases

Team OurColor members and their individual responsibilities are as follows:

- Student Name: Project Manager
- Student Name: Programmer
- Student Name: Designer

- Student Name: Designer/Programmer
- Student Name: Software Engineer
- Student Name: Requirements Engineer
- Student Name: Test Engineer
- Student Name: Test Engineer
- Student Name: Requirements Engineer

Table of Contents

1	<i>Introduction</i>	5
	1.1 Project Purpose and Goals.....	5
	1.2 Design Approach.....	6
	1.3 Traceability Approach.....	6
	1.4 Background.....	7
	1.5 Document Organization.....	8
2	<i>Requirements Analysis</i>	8
	2.1 Graphical User Interface.....	9
	2.2 Software interface.....	9
	2.3 Functional requirements.....	9
	2.3.1 Model Seven Aircraft Types.....	9
	2.3.2 Model Three Queuing Scenarios.....	9
	2.3.3 Provide Insert and Delete Capabilities.....	10
	2.3.4 Generate Final Report.....	10
	2.4 Performance requirements.....	10
	2.5 Design constraints.....	11
	2.5.1 Utilize the Simlib Library.....	11
	2.5.2 Develop ARMS Code in Java.....	11
	2.5.3 Utilize a "Middle-Pointer" Algorithm.....	11
	2.5.4 Support Extensibility.....	11
3	<i>Design</i>	11
	3.1 Context Diagram.....	12
	3.2 Level 1 Data Flow Diagram and P-Specs.....	13
	3.3 Level 2/3 Data Flow Diagrams and P-Specs.....	15
	3.4 Transaction Analysis.....	21
	3.4.1 Partitions of the System DFD.....	21
	3.4.2 Identified Transactions.....	22
	3.5 Transform Analysis and Structure Chart.....	24
	3.6 Design Decision Log.....	25
4	<i>References</i>	28
5	<i>Glossary</i>	28
	<i>APPENDIX A: Data Dictionary</i>	30
	<i>APPENDIX B: Project Schedule</i>	44
	<i>APPENDIX C: Requirements Traceability Matrix</i>	47
	<i>APPENDIX D: Identified Test Cases</i>	52
	<i>APPENDIX E: Simlib Flow Diagram</i>	55
	<i>APPENDIX F: Simlib API</i>	57
	<i>APPENDIX G: GUI Prototypes</i>	59
	<i>APPENDIX H: User-Provided Aircraft Parameters</i>	62

Table of Figures

Figure 1. ARMS context diagram.....	12
Figure 2. ARMS Level 1 data flow diagram.....	13
Figure 3. Level 2 DFD exploding Bubble 1.....	16
Figure 4. Level 2 DFD exploding Bubble 2.....	17
Figure 5. Level 2 DFD exploding Bubble 3.....	18
Figure 6. Level 3 DFD exploding Bubble 3.2.....	19
Figure 7. Partitions of the system DFD.....	22
Figure 8. ARMS structure chart.....	25
Figure 9. OurColor project schedule	45
Figure 10. Flow of control for the next-event time-advance approach.....	56
Figure 11. Input screen prototype.	60
Figure 12. Progress screen prototype.....	60
Figure 13. Report screen prototype.....	61

List of Tables

Table 1. OurColor tasks and dependencies	46
Table 2. Requirements Traceability Matrix.....	48
Table 3. Aircraft Parameter Table and corresponding value definitions.....	63

1 Introduction

On September 9, 2000, Team OurColor was established by Dr. F. Sheldon, Manager, to develop an Aircraft Repair Simulation Model (ARMS) in support of the Earth Science Enterprise Program. This notebook details the ARMS design. This section provides an overview of the ARMS software and outlines the remainder of the document. Specifically, Section 1.1 describes the project purpose and goals; Section 1.2 describes the design approach; Section 1.3 describes the traceability approach; and Section 1.4 describes the organization of the document.

1.1 Project Purpose and Goals

As requested by the client, the ARMS software will use a discrete, next-event time advance model to simulate the aircraft repair process [1]. The purpose of ARMS is to compare various maintenance scheduling alternatives in order to optimize repair times and downtime costs. Specifically, seven types of commercial aircraft will be evaluated using three different scheduling scenarios.

The product will also provide a graphical user interface, so the user can quickly and easily set input parameters, run the model, compare scheduling alternatives, delete and/or insert items, and view results. ARMS will be developed in Java using the Simlib library (provided by the customer) to perform standard simulation tasks. The ARMS application will use the user-specified set of input parameters, a client-specified set of probabilistic aircraft parameters, and the Simlib functions to estimate overall average daily downtime costs. ARMS will also be developed to facilitate future extensibility of scheduling alternatives and aircraft.

The goal of Team OurColor is to deliver a fully verified and validated software package that provides the above-mentioned functionality by December 11, 2000. To this end, Team OurColor has employed a standard Waterfall-based software engineering strategy. The tasks associated with this process are provided in Appendix B. This document represents the design portion of that approach.

1.2 Design Approach

In compliance with IEEE standards [2] and industry practice [3], Team OurColor has utilized a function-oriented, Structured System Analysis and Structured Design (SSA/SD) methodology. This design methodology was selected to ensure that all customer requirements will be met and verified, and that a cohesive, loosely coupled application will be developed. Reflexive of this design strategy, the following steps were performed:

Steps 1-2: Structured systems analysis

Step 3: Transaction analysis

Step 4: Transform analysis

Step 5: Creation of master structure chart

As a result of these steps, the following design deliverables were produced:

- Data flow diagrams (DFDs)
- Process specifications (P-specs)
- Structure chart
- Data dictionary
- Test cases

The results of these steps are provided in Section 3 and appendices A and D. The design was developed to address each of the project requirements identified in the Software Specification Requirements document [4]. Particular attention was paid to the extensibility requirement, with the goal of highly modular code that is linked to extensible data structures. The ARMS specification requirements are included in Section 2. To support verification of these requirements, a Requirements Traceability Matrix, which links requirements to design modules, is provided in Appendix C. Team OurColor traceability approach is explained in more detail in Section 1.3 below.

1.3 Traceability Approach

This section explains the approach that Team OurColor is using to verify and validate the ARMS application requirements. The first step was to number and categorize each project requirement, so they reflect a logical organization. This helped to ensure that no requirements were accidentally omitted. Next, testing methods were identified for verifying each requirement. All

non-verifiable requirements were removed. A Requirements Traceability Matrix (RTM) was then developed to provide a convenient "check-off" list. Following the Critical Design Review (CDR) [5], the DFD modules were linked to the RTM requirement categories. This association is reflected in the RTM provided in Appendix C. The final step will be to test each module against its assigned requirements. The results of this analysis will be provided in the forthcoming Test Report.

1.4 Background

As Simlib functions represent the core engine for ARMS event simulation, this section provides a brief summary of the Simlib library as a background for the design.

Simlib provides generic statistical functions and internal data structures that can be used to perform discrete, next-event time advance simulations. Simlib contains 100 lists of random numbers (called streams) which are used in conjunction with statistical distribution functions (e.g., exponential) to calculate event times. Each different random variable is assigned its own stream.

Simlib also contains 25 lists, which can be used to store internal data. Each list is a C linked list of double arrays. Each array contains ten double values which may be assigned by the user. The first element in each array is, by default, the event time, and the second is the event type. List 25 is the event list, where the calculated event times are stored. The event list is sorted in ascending time order.

Simlib functions use a transfer array to move elements between lists. Each array to be inserted must first be placed in the transfer array. Likewise, each array removed may be found in the transfer array.

In addition, Simlib provides summary functions [`sampst()` and `timest()`] that can be called as each element is added or removed from a list. These functions automatically summarize statistical information, such as the time-average delay for each element that passed through the list.

Simlib must first be initialized by calling its initialization routine. This sets the clock to zero, empties all lists, and initializes all random number streams. Next, initial events may be

scheduled by calculating event times as described above and inserting them into the event list. The Simlib timing() function is then used to receive the next event from the event list. (The timing() function removes the next event from the event list and places it in the transfer array.) After each event is handled, new events are scheduled and/or the sampst() and timest() functions are called as appropriate. This cycle continues until the simulation is complete. At this time, Simlib also provides report generator functions which return all the summary information calculated by the sampst() and timest() functions.

A flow diagram representing a generic Simlib simulation is provided as Appendix E. In addition, the Simlib API function prototypes are provided as Appendix F. This information was obtained from *Simulation Modeling and Analysis, 3rd Edition* [6].

1.5 Document Organization

This Design Notebook is divided into five sections and six appendixes. Section 1 describes the purpose and scope of the ARMS application and the overall design approach. Section 2 summarizes the project requirements, as provided in the Software Requirements Specification document [4]. Section 3 details the functional and structural design. Section 4 lists the references cited. Section 5 provides a glossary of commonly used terms and acronyms. Appendix A presents the data dictionary. Appendix B contains the project schedule. Appendix C provides the Requirements Traceability Matrix (RTM). Appendix D summarizes some test cases that were identified during the transaction analysis of the design. Appendix E provides a flowchart of the Simlib simulation process. Appendix F provides the Simlib API. Appendixes G and H provide the GUI prototypes and User-Provided Aircraft Parameters respectively.

2 Requirements Analysis

This section summarizes the requirements for the ARMS application. Specifically, it describes external interfaces, functional requirements, performance requirements, and design constraints. These requirements are listed in more detail in the ARMS Software Specifications Requirements report [4].

2.1 Graphical User Interface

The ARMS application shall provide an input screen that allows users to set the random seed, select a queuing scenario from a list of pre-defined scenarios, select the number of service stations, and start the simulation. An example of this screen is provided in Appendix G.

The ARMS application shall provide a progress screen that allows users to view simulation progress, pause the simulation, insert or delete items from the simulation, and increase or decrease the speed of the simulation. An example of this screen is provided in Appendix G.

The ARMS application shall provide an output screen that allows users to view simulation results. An example of this screen is provided in Appendix G.

2.2 Software interface

The ARMS application shall utilize Simlib library functions to calculate probabilistic event times, to increment the simulation clock, to store internal data, and to calculate time-averaged summary statistics.

2.3 Functional requirements

The ARMS application shall model the aircraft repair process using a next-event time advance simulation. As stated above, the Simlib library shall provide the probabilistic and event scheduling mechanisms required for the simulation. This section summarizes the major functional requirements associated with the aircraft repair model.

2.3.1 Model Seven Aircraft Types

The ARMS application shall support seven distinct aircraft types. The aircraft arrival and departure events shall be modeled according to the aircraft-specific attributes listed in Appendix H.

2.3.2 Model Three Queuing Scenarios

The ARMS application shall evaluate three distinct queuing scenarios and support the addition of more scenarios in the future. These initial scenarios are defined below:

Queuing Scenario 1: All waiting aircraft form a single, first-in/first-out (FIFO) queue. All stations (the number specified by the user) service all aircraft.

Queuing Scenario 2: Widebody and regular body aircraft wait in separate FIFO queues. The widebody queue is given non-preemptive priority over the regular body queue. All stations (the number specified by the user) service all aircraft.

Queuing Scenario 3: Widebody and regular body aircraft wait in separate FIFO queues. One subset of stations service only widebody aircraft, and another subset service only regular body aircraft. (The number of stations in each set is specified by the user.)

2.3.3 Provide Insert and Delete Capabilities

The ARMS application shall allow the user to insert events into the event list. The ARMS application shall also allow the user to delete events from the event list and to delete aircraft from a queue or service station.

2.3.4 Generate Final Report

The ARMS application shall calculate summary statistics using Simlib library functions. Specifically, these statistics shall include the average delay in queue for each aircraft type; the overall average delay in queue for all aircraft types; the time-average number of aircraft in the wait queue(s); the time-average number of aircraft down for each aircraft type; and the total average daily downtime cost for all aircraft added together.

The final report statistics shall reflect costs generated by aircraft prematurely removed from the maintenance process; costs generated by manually inserting aircraft into the maintenance process; and costs generated by all aircraft still in the maintenance process at the end of the simulation.

2.4 Performance requirements

The ARMS application, after receiving acceptable input and no interruptions, shall be able to complete a full simulation in no more than sixty (60) minutes. The ARMS application shall operate on a computer with 10 MB of available hard drive space and 32 MB of available RAM.

2.5 Design constraints

This section describes customer-specified requirements that have directly impacted design decisions.

2.5.1 Utilize the Simlib Library

The ARMS application shall utilize, where applicable, the functions and data structures defined in the Simlib simulation library. Specifically, Simlib list-structures shall be used to represent the event list, wait queues, and service stations.

In addition, the ARMS application shall use Simlib “streams” to provide the random numbers used in probability calculations. Specifically, streams 01 – 07 shall be used for aircraft inter-arrival times; streams 08 – 14 shall be used for engine inspection times (*first or subsequent*); streams 15 – 21 shall be used for determination of (*additional*) needed repairs; and streams 22 – 28 shall be used for engine repair times (*first or subsequent*).

2.5.2 Develop ARMS Code in Java

The ARMS application shall be developed using the Java programming language, version 1.3. The Simlib C-library shall be compiled under a Linux operating system environment and shall be accessed via the Java Native Interface.

2.5.3 Utilize a "Middle-Pointer" Algorithm

The “middle-pointer” algorithm shall be used to insert events into the event list. Records deleted from any list shall be placed in a transfer array.

2.5.4 Support Extensibility

The ARMS application shall support future extensibility of aircraft, queues, and service stations.

3 Design

As mentioned above in Section 1.2, Team OurColor has utilized a function-oriented, Structured System Analysis and Structured Design (SSA/SD) methodology for the ARMS software. The design was based on the project requirements and the next-event time advance approach discussed in Section 1.4.

This section explains the design process in detail. Specifically, Sections 3.1-3.4 present the

results of the structured systems analysis (context diagram, DFDs, P-Specs), which defines the functionality and data flows of the ARMS system. Section 3.4 describes the transaction analysis, which identifies cooperating subsystems of the software. Section 3.5 presents the transform analysis, which defines the structure of the system. Finally, Section 3.6 provides a design decision log, which documents the design evolution. In addition, descriptions of all data entities represented on the DFDs can be found in Appendix A.

3.1 Context Diagram

This section presents the context diagram for the ARMS software, shown below in Figure 1. As the first step of the SSA/SD process, it represents the highest level of abstraction and identifies any external interfaces. As indicated in the diagram, the ARMS software was designed with Simlib statistical functions at the core. (Simlib library functions are explained above in Section 1.4). All user interaction occurs through the graphical user interface, including input parameters and simulation output.

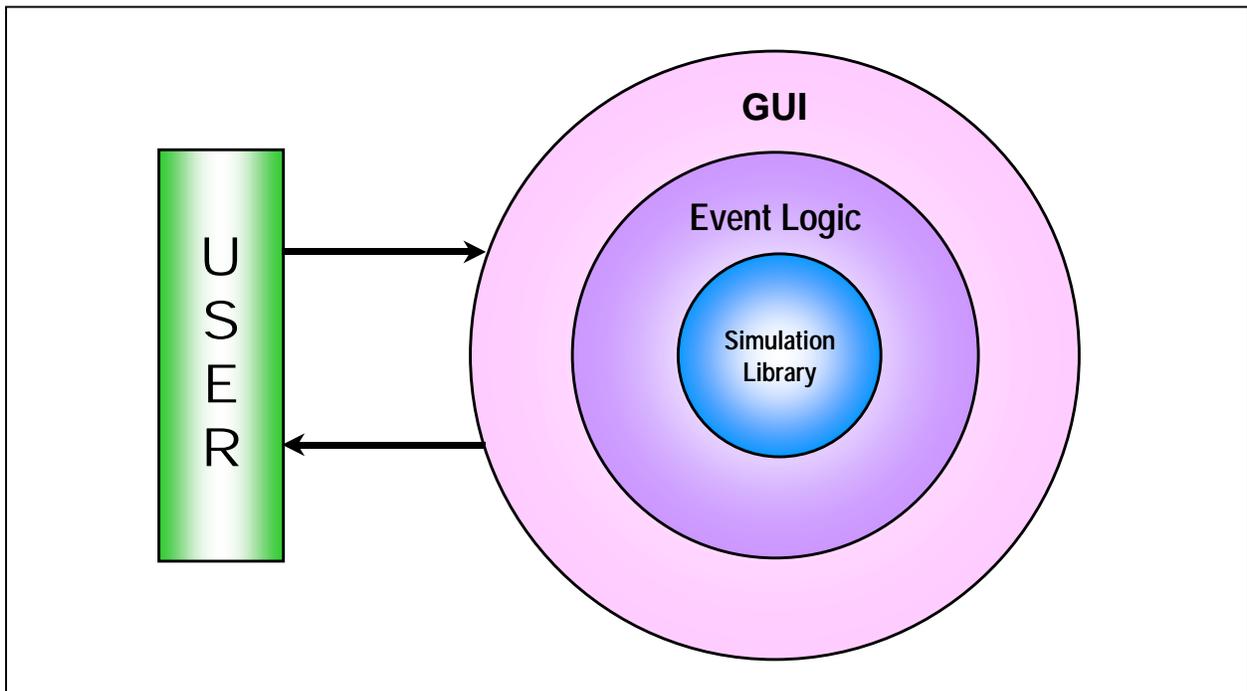


Figure 1. ARMS context diagram.

3.2 Level 1 Data Flow Diagram and P-Specs

This section describes the highest level functionality of the ARMS software. As shown below in Figure 2, three main functions are included in this diagram. First, the user input parameters must be evaluated for validity, and the simulation input data (i.e., the queuing scenario) must be determined. Next, the simulation must be initialized by setting the clock to zero and initializing the service stations, queues, and arrival events. Finally, once the simulation has successfully started, the program processes each arrival, departure, or endsim event in time order. The user also has the option to pause the simulation during this time, or to insert or delete items from the queues, service stations, or event list. Once the endsim event is reached, the summary statistics are determined and the program halts. All output is displayed to screen.

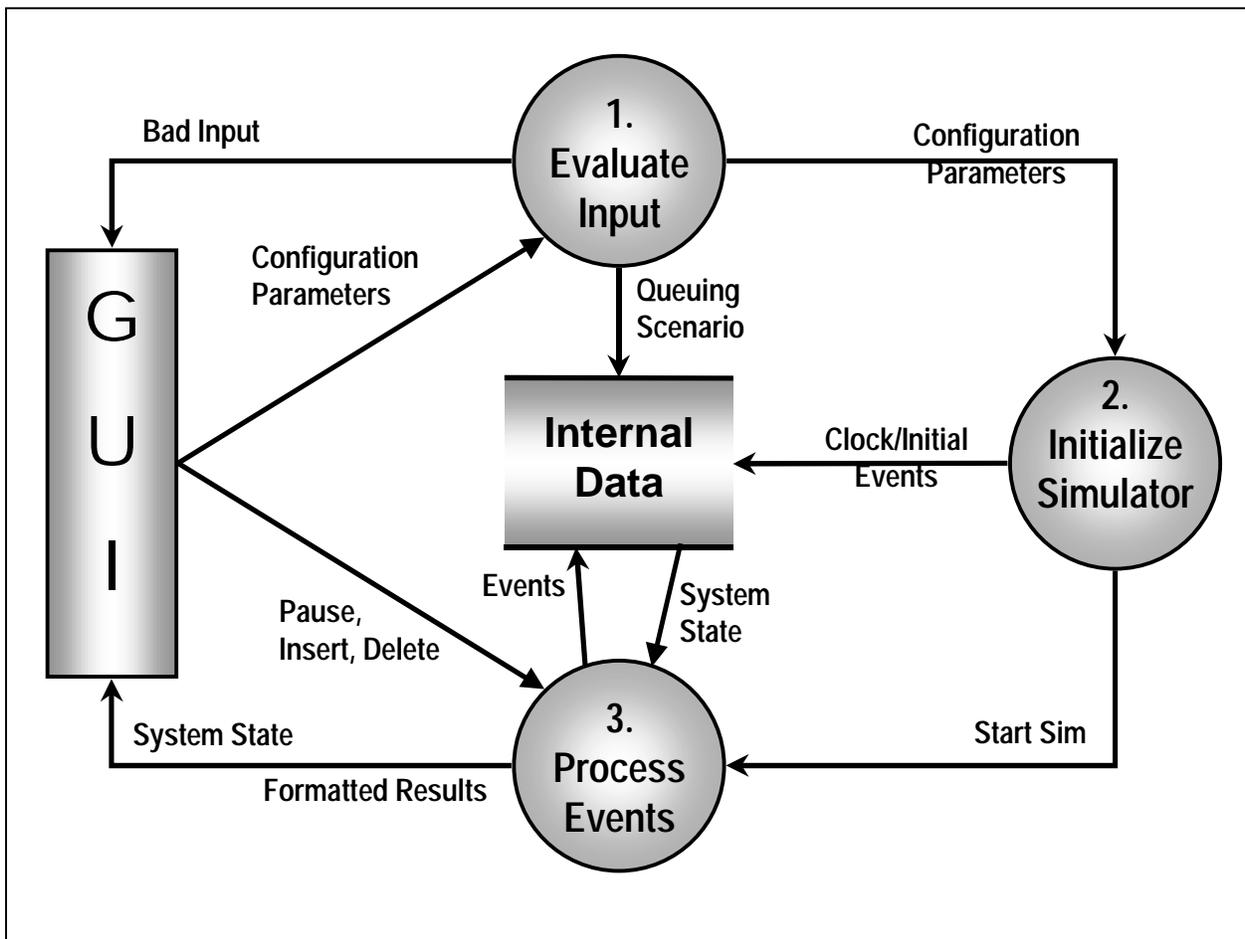


Figure 2. ARMS Level 1 data flow diagram.

Level 1 P-Specs, which further explain the top-level DFD bubbles in pseudocode, are provided below:

1. Evaluate Input

/* Customer presses “start” button */

1.1 Load selected queuing scenario

1.1.1 Set information about service stations

1.1.2 Set information about aircraft

1.1.2 Set information about queues

1.2 Evaluate number of service stations

1.2.1 If number of service stations entered is invalid for scenario, re-prompt for input

1.3 Evaluate random seed

1.3.1 If random seed not a number or blank,
Use default random seed

2. Initialize Simulator

/* Customer presses “start” button and input data is valid */

2.1 Initialize Simlib

2.1.1 Initialize random number streams

2.1.2 Initialize event List

2.1.3 Set system clock to 0

2.2 Initialize delay time to default value of one second.

2.3 Initialize pause flag to default value of true.

3. Process Events (*infinite loop*)

/* Always occurs after system is initialized */

- 3.1 If “pause” mode is activated, process
 - external events
 - 3.1.1 If delete selected, process delete
 - 3.1.2 If insert selected, process insert
 - 3.2 Else, process internal events
 - 3.2.1 Get next event from event list (via Simlib transfer array)
 - 3.2.2 If arrive, process arrival
 - 3.2.3 If depart, process departure
 - 3.2.4 If endsim, process endsim
 - 3.2.5 Wait the delay time
 - 3.3 Update progress GUI
 - 3.3.1 Print event list
 - 3.3.2 Print queue lists
 - 3.3.3 Print service station lists
 - 3.3.4 Print current simulation time

3.3 Level 2/3 Data Flow Diagrams and P-Specs

This section provides a more detailed look at Bubbles 1-3 from the high level DFD. Figure 3 below explodes Bubble 1. As indicated in this figure, the Evaluate Input bubble entails: 1) initializing the queuing scenario (an internal data store which describes the queues, service stations, and aircraft being used in the simulation); 2) verifying the user-specified number of bays (i.e., making sure it is appropriate for the selected queuing scenario); and 3) selecting a random seed (a default value is used if none is entered by the user).

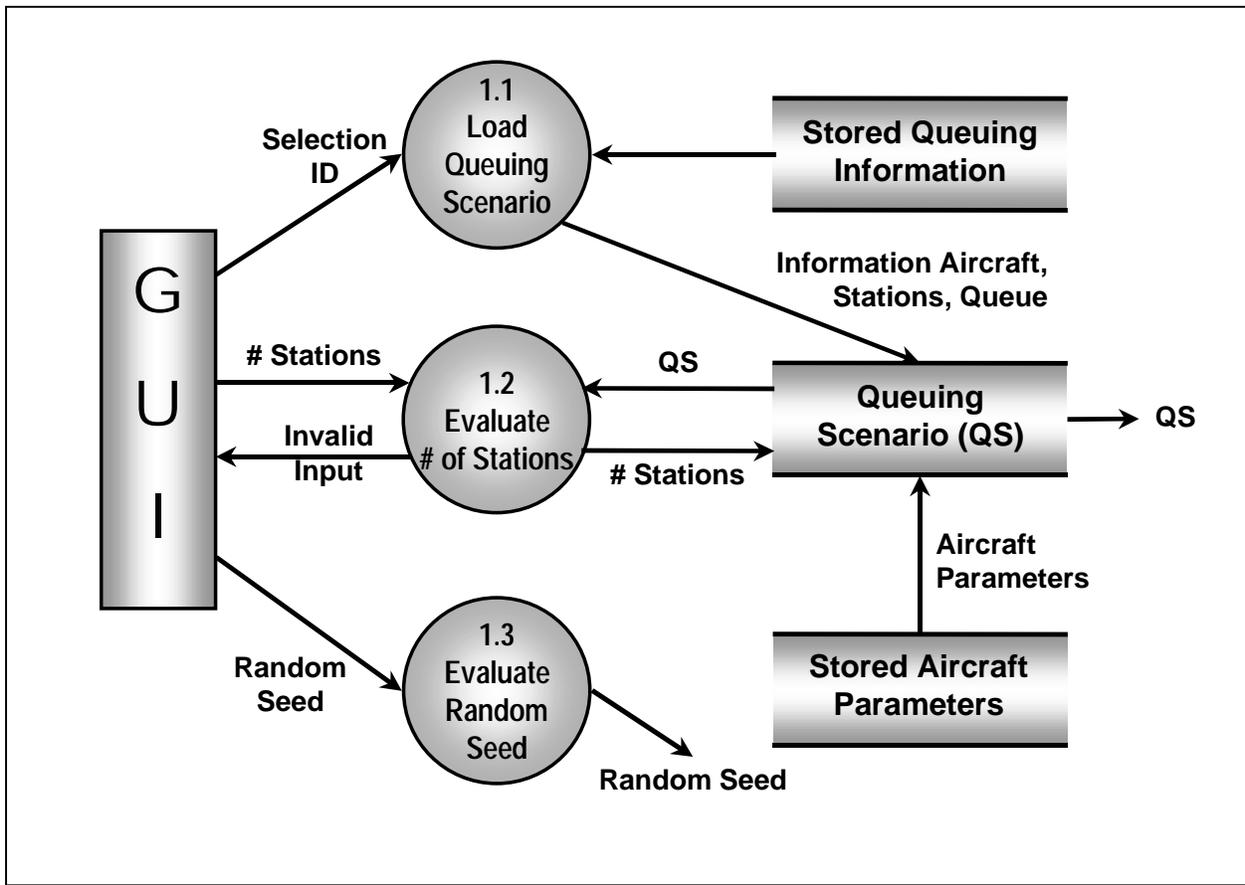


Figure 3. Level 2 DFD exploding Bubble 1.

Figure 4 below explodes Bubble 2. As indicated in this figure, the Initialize Simulator bubble entails: 1) initializing the Simlib library (including the clock and its internal data stores); 2) initializing the delay time (the internal "throttle" value that controls the speed of the simulation; and 3) initializing the pause flag to true (i.e., the simulation will start in paused mode).

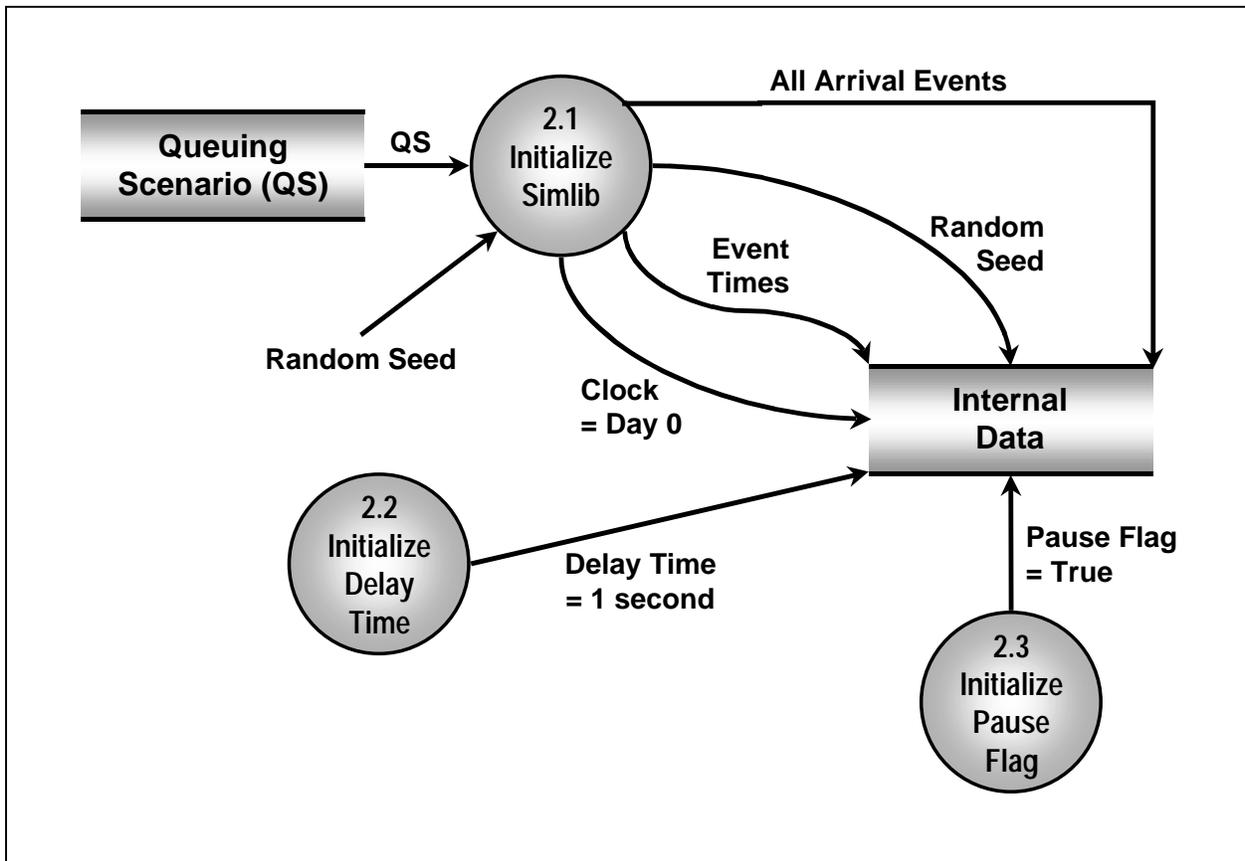


Figure 4. Level 2 DFD exploding Bubble 2.

Figure 5 below explodes Bubble 3. As indicated in this figure, the Process Events bubble entails: 1) processing external events (i.e., Insert or Delete, as requested by the user via the Progress GUI); 2) processing internal events (i.e., Arrival, Departure, or Endsime, as issued by Simlib); and 3) updating the GUI with the current state of the system. Note that the user will only be allowed to insert or delete items when the system is in paused mode. Otherwise, the internal events cycle will be processed automatically, with no user intervention.

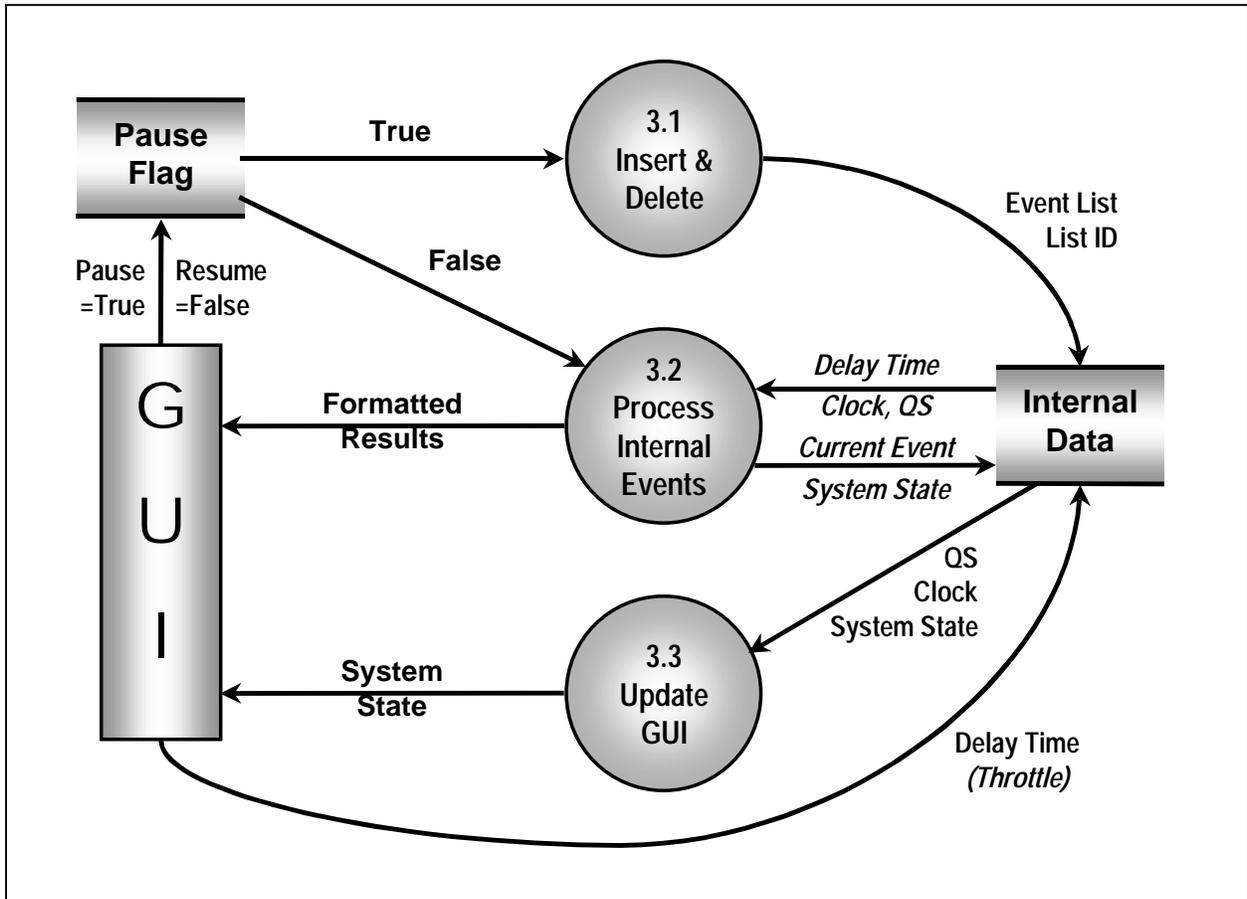


Figure 5. Level 2 DFD exploding Bubble 3

Because the processing of internal events (which loops indefinitely until Endsime occurs) represents the main engine of the ARMS software, Bubble 3.2 has been further exploded into a level 3 DFD (Figure 6 below). This DFD shows how the throttle value is used to delay the simulation progress and how the Endsime event results in the output statistics being displayed to screen.

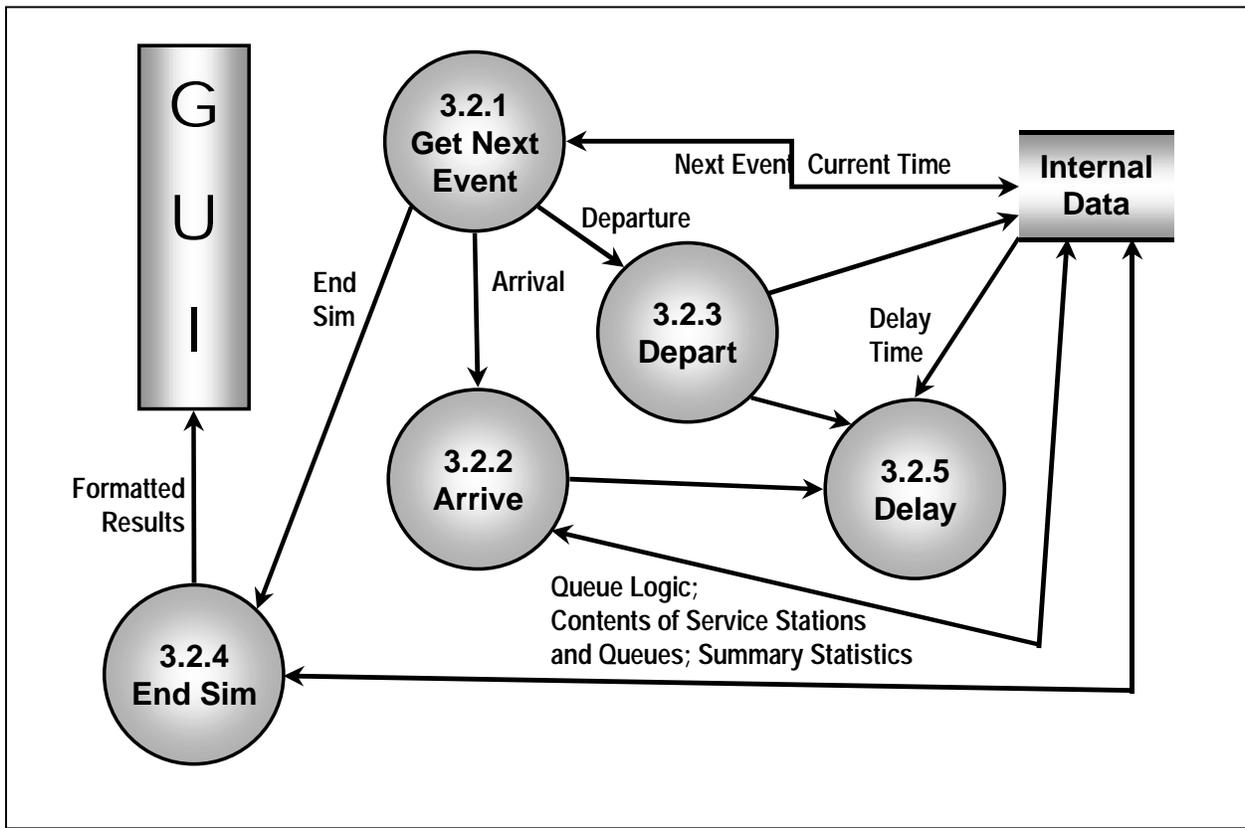


Figure 6. Level 3 DFD exploding Bubble 3.2

Also because of the importance of the event processing (Insert, Delete, Arrive, Depart, Endsime), these functions are explained in more detail in the P-Specs below.

3.1.1 Process Delete

/ Occurs if user selects pause, then delete option */*

3.1.1.1 Find selected event from list using middle pointer algorithm.

3.1.1.2 Delete event from list

3.1.1.3 If deleting from bay, force aircraft departure.

3.1.1.4 If deleting from queue, update statistics.

3.1.2 Process Insert

/ Occurs if user selects pause, then insert option */*

3.1.2.1 Find location to insert in event list using middle

pointer algorithm.

3.1.2.2 Insert event into event list.

3.2.2 Process Arrival

/ Occurs if first event in event list is an arrival */*

3.2.2.1 If any service stations that can take this aircraft are empty, move aircraft into station and schedule next departure time.

3.2.2.2 Else, move aircraft into least filled queue that will accept it and update queue statistics.

3.2.3 Process Departure

/ Occurs if first event in event list is a departure */*

3.2.3.1 Remove aircraft from appropriate station list and update service station statistics. (Ignore departure if no such aircraft in that station.)

3.2.3.2 Choose next aircraft from appropriate queue, move aircraft into station, and schedule next departure time.

3.2.4 Process Endsim

/ Occurs if first event in event list is endsim*/*

3.2.4.1 For each bay, update statistics for that service station (assuming aircraft have just departed).

3.2.4.2 Generate Simlib summary data (average time spent in queue, average downtime, etc.).

3.2.4.3 Calculate average downtime costs for all aircraft types.

3.2.4.4 Display results on output GUI.

3.2.4.5 Exit application.

3.4 Transaction Analysis

The purpose of the transaction analysis is to break down the system into a network of cooperating subsystems. This helps to identify test cases that will verify different parts of the system. It also leads to the transform analysis step, where structure is applied to each subsystem partition. Section 3.4.1 presents the subsystem partitions for the ARMS design, and Section 3.4.2 lists the identified transactions.

3.4.1 Partitions of the System DFD

To partition the system, the five DFDs presented above were first combined into one master DFD. Then transactions (identified in Section 3.4.2) were used to separate the cooperating subsystems. This partitioning is shown below in Figure 7. Note that because of the relative simplicity of the ARMS design, the DFD bubbles were naturally placed in a somewhat hierarchical order during the transaction analysis step. Also note that in most cases, the ARMS application will perform as a single subsystem, traversing each node in the DFD (the same process will occur no matter which queuing scenario is selected). Only in the cases of: 1) invalid input or 2) the user requests an insert or delete event, will additional bubbles of the DFD be activated. These transactions are reflected in the system partitioning of Figure 7.

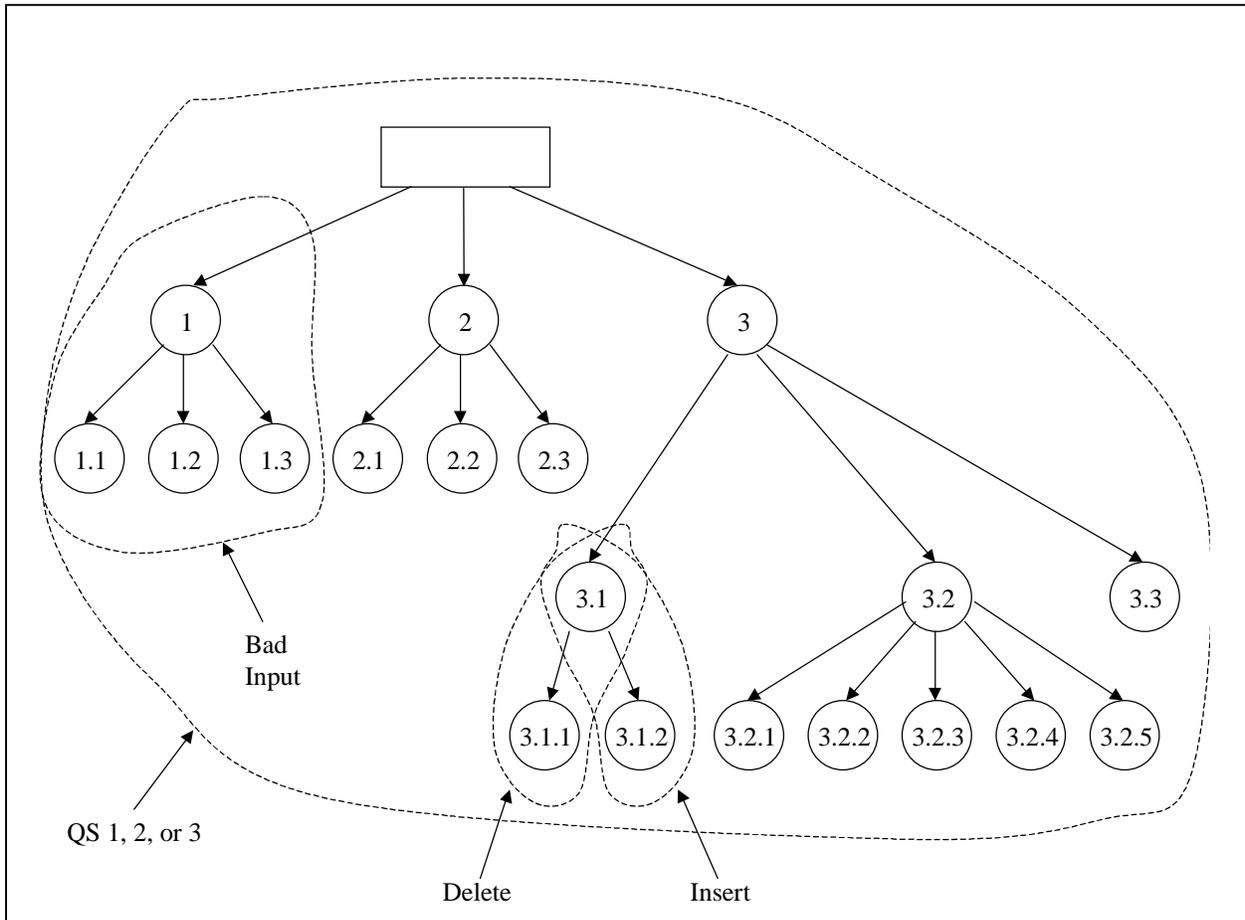


Figure 7. Partitions of the system DFD

3.4.2 Identified Transactions

This section presents transactions intended to trigger all possible flow paths of the ARMS system. Each transaction is identified by the following five items, as defined in Budgen [3]:

- The event in the systems environment that causes the transaction to occur.
- The stimulus that is applied to the system to inform it about the event.
- The activity that is performed by the system as a result of the stimulus.
- The response that this generates in terms of output from the system.
- The effect that this has upon the environment.

Transaction 1:

Event: Running stored Queuing Scenario #1 (QS1).

Stimulus: User selects QS1, enters a valid number of service stations, and presses start.

Activity: ARMS runs to completion using the logic defined by QS1.

Response: Results are displayed to output GUI.

Effect: Verifying the description and logic associated with QS1.

Transaction 2:

Event: Running stored Queuing Scenario #2 (QS2).

Stimulus: User selects QS2, enters a valid number of service stations, and presses start.

Activity: ARMS runs to completion using the logic defined by QS2.

Response: Results are displayed to output GUI.

Effect: Verifying the description and logic associated with QS2.

Transaction 3:

Event: Running stored Queuing Scenario #3 (QS3).

Stimulus: User selects QS3, enters valid numbers of service stations, and presses start.

Activity: ARMS runs to completion using the logic defined by QS3.

Response: Results are displayed to output GUI.

Effect: Verifying the description and logic associated with QS3.

Transaction 4:

Event: Running stored Queuing Scenario #3.

Stimulus: User selects QS3, enters invalid numbers of service stations, and presses start.

Activity: ARMS prompts user for re-entry until valid parameters are entered.

Response: Error message is displayed to GUI.

Effect: Verifying the error checking of input parameters.

Transaction 5:

Event: Running stored Queuing Scenario #2.

Stimulus: User presses Pause button on progress GUI, presses button to Delete an aircraft from the queue, and presses Resume.

Activity: ARMS deletes that aircraft from the selected queue and then runs to completion.

Response: Progress GUI shows that the aircraft has been removed from the queue.

Effect: Verifying the Delete function.

Transaction 6:

Event: Running stored Queuing Scenario #1.

Stimulus: User presses Pause button on progress GUI, presses button to Insert an event to the event list, and presses Resume.

Activity: ARMS prompts the user for a description of the event, inserts the event to the event list, and then runs to completion.

Response: Progress GUI shows that the event has been inserted into the event list.

Effect: Verifying the Insert function.

3.5 Transform Analysis and Structure Chart

The transform analysis marks the final phase of the SSA/SD design process. During the transform analysis, the subsystem DFDs determined in the Transaction Analysis are each arranged in a hierarchical structure chart, and then all subsystems are combined into one hierarchical structure chart that reflects the entire system.

Given the relative simplicity of the ARMS design, each DFD flows right into the next without overlap; so a hierarchical progression emerged naturally from the DFDs of Sections 3.2 and 3.3. Similarly, given the small number of cooperating subsystems, combining all the structure charts into one master chart was a relatively straightforward task. As a result, the intermediate step of producing a non-hierarchical DFD with central transform (NHDFD) was not necessary. Figure 8 below represents the overall structure of the ARMS system, showing control flow between modules. Note that the GUI components are not reflected in this chart. This is because they represent only an interface for providing data to or receiving data from the system.

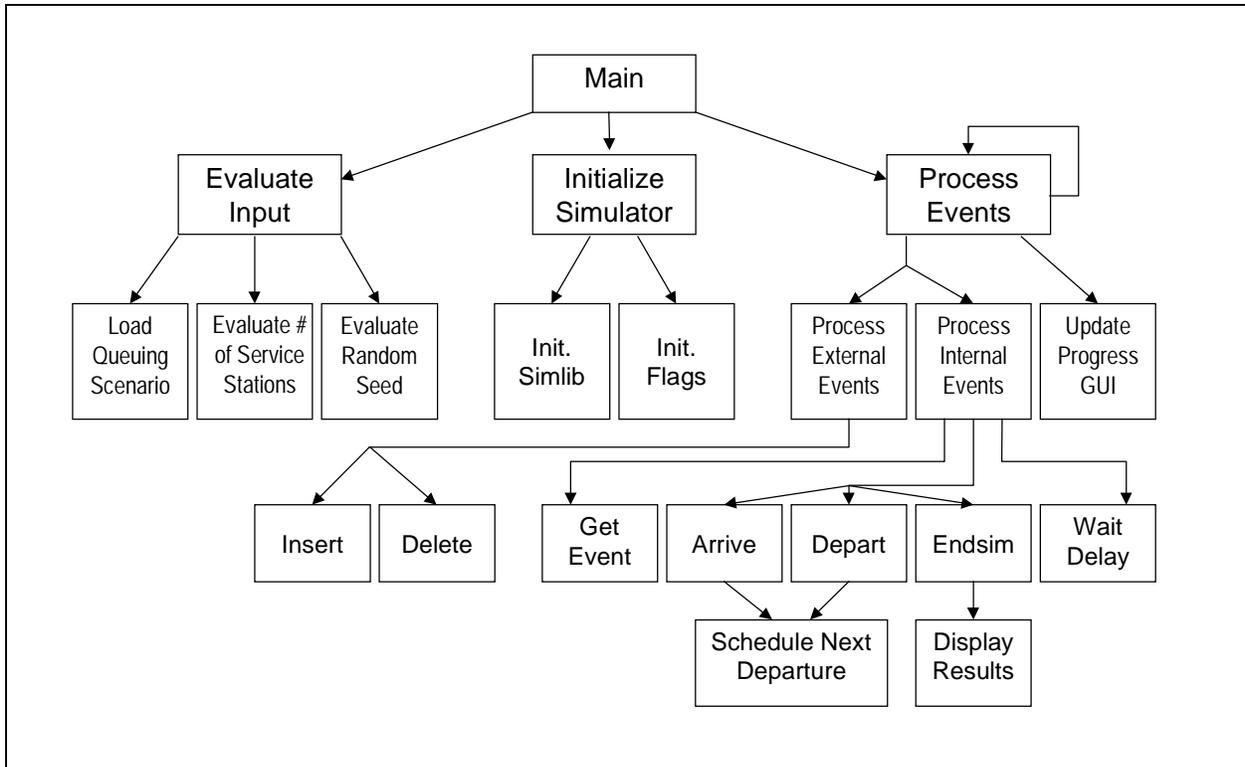


Figure 8. ARMS structure chart

3.6 Design Decision Log

This section documents the evolution of the ARMS design. Significant decisions or events that have impacted the design are listed below. Decisions are generally categorized on a week-by-week basis.

Date	Decisions
9/8	<ul style="list-style-type: none"> – Decided team member roles and responsibilities. – Selected Java as the programming language for ARMS development.
9/16	<ul style="list-style-type: none"> – Started to understand the problem description. – Decided on an input screen for selecting max number of service stations and queuing style. – Decided to vary the number of stations for the user in a multi-run simulation and select the optimal number of bays for him.

Date	Decisions
	<ul style="list-style-type: none"> <li data-bbox="402 268 1437 352">– Decided to superimpose the results of the various runs using multi-colored line graphs. <li data-bbox="402 394 1437 430">– Decided to have input and output screens, but no progress screen.
9/22	<ul style="list-style-type: none"> <li data-bbox="402 472 1437 556">– Decided to add a progress screen that shows contents of event list, queues, and service stations. <li data-bbox="402 598 1437 634">– Decided progress screen will not be a moving process flow diagram. <li data-bbox="402 676 1437 711">– Decided to allow insert/delete capabilities only for the event list. <li data-bbox="402 753 1437 837">– Decided to indirectly handle insert/delete using an "edit" button for the event list. <li data-bbox="402 879 1437 1018">– Decided to not show the details of the repair cycle inside a service station -- only to show when an aircraft enters and leaves the station (although the repair cycle will be used to calculate the departure time). <li data-bbox="402 1060 1437 1207">– Decided to base the simulation engine on Simlib's internal process: 1) initialize, 2) cycle through events in time-order, adding new events where appropriate, and 3) generate summary report.
9/30	<ul style="list-style-type: none"> <li data-bbox="402 1249 1437 1333">– Learned how to call C functions from Java using the Java Native Interface. Identified Simlib extensions required for this to work. <li data-bbox="402 1375 1437 1411">– Decided to call Simlib C code from Java instead of porting it to Java. <li data-bbox="402 1453 1437 1537">– Decided to add a "throttle" value to the progress screen so user can increase or decrease the simulation speed. <li data-bbox="402 1579 1437 1663">– Decided to add a random seed input parameter to the input screen and to use a default value if none is entered.
10/7	<ul style="list-style-type: none"> <li data-bbox="402 1705 1437 1789">– Decided to add a "delete" button for the queues and service stations on the progress screen, but still kept the "edit" button for the event list.

Date	Decisions
	<ul style="list-style-type: none"> - Decided how to implement the "middle-pointer" algorithm for inserting to Simlib lists. Identified Simlib extensions needed for this to work.
10/14	<ul style="list-style-type: none"> - Decided to remove "edit" button from the progress screen's event list (as this was confusing). Instead, the event list will have an "insert" and "delete" button, consistent with the "delete" buttons on the queue and service station lists. The "insert" button will trigger small box to prompt for event details. - Decided that the user will have the power to insert or delete any event, but ARMS will not check for illogical uses of events. - Decided that extra departure events (if inserted illogically) will just be ignored. - Decided that any aircraft still in a service station when the simulation ends will be included in the summary statistics. - Decided that deleting an aircraft from a service station will force the departure of that aircraft. - Decided to add extensibility to the ARMS code. The input screen will have a placeholder button for future functionality. Eventually, user will be able add and describe new aircraft, service stations, or queues. - Decided to schedule all arrival events up front in the initialization routine, instead of scheduling the next arrival after each arrival is removed from the event list.
10/24	<ul style="list-style-type: none"> - Decided to implement extensibility using a "Queuing Scenario" (QS) data structure. This structure holds all the information about the aircraft, queues, and service stations in the simulation. It includes information such as what aircraft a queue will accept, what queues get higher priority, and what aircraft a service station will accept. This allows all scheduling decisions to be based on this structure alone. As a result, all decisions are based on "extensible"

Date	Decisions
-------------	------------------

parameters that can easily be changed. In addition, only one piece of logic is required for any queuing scenario, instead of one function for each queuing scenario, as was initially thought.

10/30 – Decided to save all QS data in a Java-formatted file, which is easier to read into the simulation. As a result, new queuing scenarios can only be added via the GUI input screen.

4 References

1. Sheldon, F.T., "Project Requirements, CS422 Software Engineering Principles" <http://www.eecs.wsu.edu/~sheldon/cs422.html>. Fall 2000.
2. IEEE-SA Standards Board, IEEE Guide to Software Design Descriptions. IEEE Std 1016.1-1993. USA, March 18, 1998.
3. Budgen, David, Software Design. Addison-Wesley 1994.
4. Team OurColor, "Software Requirement Specifications for Aircraft Repair Modeling Simulation - Final Draft" October, , 2000.
5. Team OurColor "Critical Design Review for Aircraft Repair Modeling Simulation." October 25, 2000.
6. Law, Averill M. and David Kelton, Simulation Modeling and Analysis, 3rd Edition. New York: McGraw-Hill Companies, October 1999

5 Glossary

This section defines the terms and acronyms that are used throughout this document.

API – Application Programming Interface

ARMS – Aircraft Repair Model Simulation

C – the programming language ANSI-C

DFD – Data Flow Diagram. Provides an indication of how data transforms occur as the data moves through the system. DFDs also depict the functions that transform the data.

ESEP – Earth Science Enterprise Program

GUI – Graphical User Interface

MB – megabyte, as in 1 million bytes of data storage

NHDFD – Non-hierarchical Data-Flow Diagram with Central Transform

PC – Personal Computer

QS – Queuing Scenario

RAM – Random Access Memory

RTM – Requirements Traceability Matrix

P-Spec – Process Specifications

SSA/SD – Structured Systems Analysis/Structured Design

SRS – Software Requirements Specification

APPENDIX A: Data Dictionary

This appendix defines the data entities represented on the data flow diagrams of Section 3. Each dictionary entry contains the following information:

Name: The name of the entity as it appears in the design documents and code.

Description: A brief description of the variable.

Used In: The functional units using this variable.

Units: The unit of measure for the variable's data.

Range: The acceptable range of data values for the variable.

Data Type: The data type to be used when declaring the variable during coding.

Attribute: Indicates whether the variable contains data, control information, or a data condition.

Data Store

Location: The common region where the variable must be stored.

Accuracy: The degree of accuracy required for output comparisons between implementations. N/A means not applicable. TBD means to be to be determined later.

Name: **Aircraft**

Description: An abstract data type that describes aircraft details. Specifically, it contains the following variables:

Plane Type (unique identifier)

Priority level (can be used as a widebody indicator)

Engines

Mean time between arrivals

Min inspection time

Max inspection time

Repair probability

Repair time

Daily cost

Used In: All modules utilize this information, as it is part of the queuing scenario.

Units: Plane Type: N/A

Priority level: N/A

Engines: N/A

Mean time between arrivals: days

Min inspection time: days

Max inspection time: days

Repair probability: N/A

Repair time: days

Daily cost: dollars

Range: Plane type: 1 - max plane types (1-7 for this analysis)

Priority level: ≥ 0

Engines: ≥ 0

Mean time between arrivals: ≥ 0

Min inspection time: ≥ 0 , \leq max inspection time

Max inspection time: \geq min inspection time

Repair probability: ≥ 0 , ≤ 1

Repair time: ≥ 0

Daily cost: ≥ 0

Data Type: Aircraft: Aircraft

ID: int

Priority level: int

Engines: int
Mean time between arrivals: float
Min inspection time: float
Max inspection time: float
Repair probability: float
Repair time: float
Daily cost: float

Attribute: Data

Data Store

Location: Pre-defined aircraft will be stored in a Java-formatted file. The aircraft defined for the selected scenario will be loaded when the QS is loaded from file.

Accuracy: All floats will be rounded to two decimal places.

***Name:* Avg Aircraft Delay**

Description: The average delay in queue for each aircraft type; part of the summary statistics displayed to the output GUI.

Used In: Process Endsim, Process Events

Units: Days

Range: 0 - 365

Data Type: Float

Attribute: Data

Data Store

Location: Temporary memory; derived from Simlib statistics; output to screen at the end of the simulation.

Accuracy: Rounded to two decimal places.

Name: Avg Aircraft Down

Description: The time-average number of aircraft down for each aircraft type; part of the summary statistics displayed to the output GUI.

Used In: Process Endsim, Process Events

Units: Aircraft

Range: ≥ 0

Data Type: Float

Attribute: Data

Data Store

Location: Temporary memory; derived from Simlib statistics; output to screen at the end of the simulation.

Accuracy: Rounded to two decimal places.

Name: Avg Downtime Cost

Description: The total average daily downtime cost for all planes added together; part of the summary statistics displayed to the output GUI.

Used In: Process Endsim, Process Events

Units: Dollars

Range: ≥ 0

Data Type: Float

Attribute: Data

Data Store

Location: Temporary memory; derived from Simlib statistics; output to screen at the end of the simulation.

Accuracy: Rounded to two decimal places.

Name: Avg Queue Length

Description: The time-average number of aircraft in the wait queue(s); part of the summary

statistics displayed to the output GUI.

Used In: Process Endsim, Process Events

Units: Aircraft

Range: ≥ 0

Data Type: Float

Attribute: Data

Data Store

Location: Temporary memory; derived from Simlib statistics; output to screen at the end of the simulation.

Accuracy: Rounded to two decimal places.

***Name:* Bay (i.e., service station)**

Description: An abstract data type that describes a service station. Specifically, it contains the following variables:

ID (unique identifier)

Bay type

Used In: All modules utilize this information, as it is part of the queuing scenario.

Units: N/A

Range: ID: 1 - max bays

Bay type: 1 - max bay types

Data Type: Bay: Bay

ID: int

Bay type: int

Attribute: Data

Data Store

Location: Bays will be created in temporary memory when the queuing scenario is

loaded from file.

Accuracy: N/A

Name: Bay Type

Description: An abstract data type that describes bay details. Bay type contains information that is common to multiple bays. This saves storage space as only one bay type must be defined for all bays of that type. Specifically, it contains the following variables:

ID (unique identifier)

Aircraft accepted

Used In: All modules utilize this information, as it is part of the queuing scenario.

Units: N/A

Range: ID: 1 - max bay types

Aircraft accepted: 1 - max aircraft

Data Type: Bay Type: Bay Type

ID: int

Aircraft accepted: array of ints

Attribute: Data

Data Store

Location: Pre-defined bay types will be stored in a Java-formatted file. The bay types defined for the selected scenario will be loaded when the QS is loaded from file.

Accuracy: N/A

Name: Delay Time

Description: Controls the speed of the simulation. Specifically, it is the amount of time to keep each change in system state up on the progress GUI. Can be set by the “throttle” slider on the progress GUI.

Used In: Process Events main loop.

Units: Milliseconds

Range: 500 to 5000 (exact range TBD)

Data Type: Int

Attribute: Control

Data Store

Location: Temporary memory; global variable; initialized to __ seconds.

Accuracy: N/A

***Name:* Event**

Description: Abstract data type containing information describing an event and its attributes. The attributes include:

Event type (Arrive, Depart, or Endsime)

Time (time the event occurred)

Bay ID (uniquely identifies bay)

Aircraft ID (uniquely identifies aircraft)

Aircraft type

Used In: Initialization, Process Events, Arrival, Departure, Insert, Delete, Endsime

Units: Event type: N/A

Time: days

Bay ID: N/A

Aircraft ID: N/A

Aircraft type: N/A

Range: Event type: 1 - 3 (Arrive, Depart, Endsime)

Time: > 0

Bay ID: 1 - # bays (user specified)

Aircraft ID: ≥ 1

Aircraft type: 1 - # aircraft types (1 - 7 for this simulation)

Data Type: Array of 10 floats (most represent integer values).

Attribute: Data

Data Store

Location: Stored in temporary memory; created by Simlib.

Accuracy: N/A

***Name:* List (Simlib variable)**

Description: Simlib list of events. The event list (list 25) sorts scheduled events in increasing time order. Queues and service station contents are also represented by Simlib lists. Each list is identified by an integer ID ranging from 1 to 25.

Used In: Initialization, Process Events (main event loop), Departure, Insert, Delete

Units: N/A

Range: N/A

Data Type: Linked list of Events.

Attribute: Data

Data Store

Location: 25 lists are stored in temporary memory; created by Simlib.

Accuracy: N/A

***Name:* Overall Avg Aircraft Delay**

Description: The overall average delay in queue for all aircraft types; part of the summary statistics displayed to the output GUI.

Used In: Process Endsim, Process Events

Units: Days

Range: 0 - 365

Data Type: Float

Attribute: Data

Data Store

Location: Temporary memory; derived from Simlib statistics; output to screen at the end of the simulation.

Accuracy: Rounded to two decimal places.

***Name:* Pause Flag**

Description: Indicates whether the simulation has been paused or not. When the simulation is paused, the Insert or Delete functions may be called via a button on the progress GUI.

Used In: Process Events main loop.

Units: N/A

Range: true/false

Data Type: Boolean

Attribute: Control

Data Store

Location: Temporary memory; global variable; initialized to true.

Accuracy: N/A

***Name:* Queue**

Description: An abstract data type that describes a queue. Specifically, it contains the following variables:

ID (unique identifier)

Queue type

Used In: All modules utilize this information, as it is part of the queuing scenario.

Units: N/A

Range: ID: 1 - max queues

Queue type: 1 - max queue types

Data Type: Queue: Queue

ID: int

Queue type: int

Attribute: Data

Data Store

Location: Queues will be created in temporary memory when the queuing scenario is loaded from file.

Accuracy: N/A

Name: **Queue Type**

Description: An abstract data type that describes queue details. Queue type contains information that is common to multiple queues. This saves storage space as only one queue type must be defined for all queues of that type. Specifically, it contains the following variables:

ID (unique identifier)

Name

Aircraft accepted

Priority level

Used In: All modules utilize this information, as it is part of the queuing scenario.

Units: N/A

Range: ID: 1 - max queue types

Name: N/A

Aircraft accepted: 0 - max aircraft

Priority level: ≥ 0

Data Type: Queue Type: Queue Type

ID: int

Name: string

Aircraft accepted: array of ints

Priority level: int

Attribute: Data

Data Store

Location: Pre-defined queue types will be stored in a Java-formatted file. The queue types selected for the simulation will be loaded into a temporary variable when the QS is loaded.

Accuracy: N/A

***Name:* Queuing Scenario**

Description: An abstract data type that contains:

Array of queues

Array of service stations

Array of aircraft

Array of queue types

Array of bay types

These variables will be used to describe the simulation.

Used In: All modules of the code utilize the queuing scenario (including the GUI format) as it defines the scenario parameters.

Units: N/A

Range: N/A

Data Type: Queuing Scenario

Attribute: Data

Data Store

Location: Pre-defined queuing scenarios will be stored in a Java-formatted file. The queuing scenario selected for the simulation will be loaded into a temporary QS variable.

Accuracy: N/A.

***Name:* Simulation Clock**

Description: Simlib variable that indicates the current day of the simulation.

Used In: Initialization, Process Events

Units: Days

Range: 0 - 365

Data Type: Float

Attribute: Data (also has a control effect)

Data Store

Location: Temporary memory; created/managed by Simlib.

Accuracy: Rounded to two decimal places.

***Name:* Stream**

Description: A list of random numbers use to calculate probabilistic parameters (one stream per random variable).

Used In: The Initialization routine used to calculate and schedule arrival times and the Departure routine used to calculate and schedule departure times.

Units: N/A

Range: > 0

Data Type: Array of long integers

Attribute: Data

Data Store

Location: Stored in temporary memory; created by Simlib.

Accuracy: N/A

Name: **Transfer Array**

Description: A storage location for one event; used to transfer events between Simlib lists.

Used In: Process Events, Arrive, Depart, Endsim (processing summary statistics).

Units: The unit of measure for the variable's data.

Range: N/A

Data Type: Array of 10 floats

Attribute: Data

Data Store

Location: Stored in temporary memory; created by Simlib.

Accuracy: N/A

APPENDIX B: Project Schedule

This appendix summarizes the tasks required to complete the ARMS project. Figure 9 is a Gantt chart which graphically displays the overall project path. Table 1 provides a complete listing of all tasks and their dependencies.

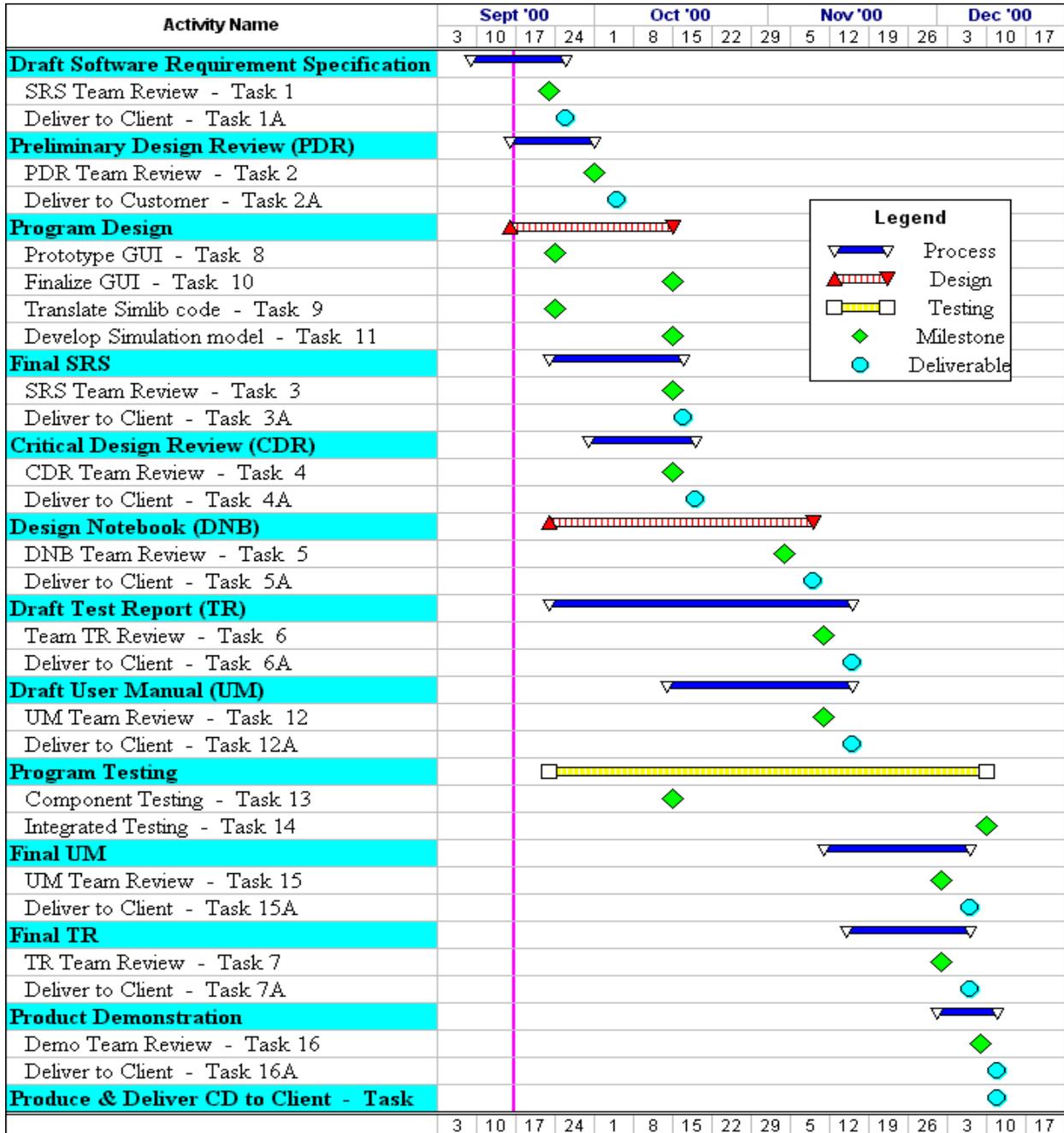


Figure 9. OurColor project schedule

Table 1. OurColor tasks and dependencies

Task #	Description	Dependent on task number	Timeframe	Milestone
1	Draft SRS		09/09 - 09/23	09/23/2000
1A	Incorp Team Review	1	09/23 - 09/25	09/25/2000*
2	PDR		09/16 - 09/30	09/30/2000
2A	Incorp Team Review	2	09/30 - 10/04	10/04/2000*
8	Prototype GUI		09/16 - 09/23	09/23/2000
9	Convert Simlib Code		09/16 - 09/23	09/23/2000
10	Finalize GUI	8	09/23 - 10/14	10/14/2000
11	Develop Sim model	9	09/23 - 10/14	10/14/2000
3	Final SRS	1A, 8, 9	09/25 - 10/14	10/14/2000
3A	Incorp Team Review	3	10/14 - 10/16	10/16/2000*
4	CDR	2, 8, 9	09/30 - 10/14	10/14/2000
4A	Incorp Team Review	4	10/14 - 10/18	10/18/2000*
5	DNB	1	09/23 - 11/04	11/04/2000
5A	Incorp Team Review	5	11/04 - 11/08	11/08/2000*
6	Draft TR	1	09/23 - 11/11	11/11/2000
6A	Incorp Team Review	6	11/11 - 11/15	11/15/2000*
7	Final TR	6A	11/15 - 12/02	12/02/2000
7A	Incorp Team Review	7	12/02 - 12/06	12/06/2000*
12	Draft UM	4, 10, 11	10/14 - 11/11	11/11/2000
12A	Incorp Team Review	12	11/11 - 11/15	11/15/2000*
13	Component Testing	8, 9	09/23 - 10/14	10/14/2000
14	Integrated Testing	10,11, 13	10/14 - 12/02	12/02/2000
15	Final UM	12A	11/11 - 12/02	12/02/2000
15A	Incorp Team Review	15	12/02 - 12/06	12/06/2000*
16	Develop Product Demo	14	12/02 - 12/09	12/09/2000
16A	Incorp Team Review	16	12/09 - 12/11	12/11/2000*
17	Produce CD	2A, 3A, 4A, 5A, 7A, 14, 15A, 16	12/09 - 12/11	12/11/2000*

* indicates external milestone

APPENDIX C: Requirements Traceability Matrix

Table 2. Requirements Traceability Matrix.

Req. ID System Level	Req. ID Sub-System Level	DFD Identifiers	Module Name(s)	Verification Method	Test
A01.00		GUI		D	
	A01.01	GUI		D	
	A01.02	GUI		D	
	A01.03	GUI		D	
	A01.04	GUI, 1.2	Evaluate # of Bays	D	
	A01.05	GUI, 1.3	Eval. Random Seed	D	
	A01.06	GUI		D	
	A01.07	GUI		D	
	A01.08	GUI		D	
	A01.09	GUI		A, D	
	A01.10	GUI		D	
	A01.11	GUI, 3.3	Update GUI	I, D	
	A01.12	GUI, 3.3	Update GUI	I, D	
	A01.13	GUI, 3.3	Update GUI	D	
	A01.14	GUI, 3.3	Update GUI	I, D	
	A01.15	GUI		D	
	A01.15.1	GUI, 3.2.4	Process Endsim	D	
	A01.15.2	GUI, 3.3	Update GUI	D	
	A01.15.3	GUI, 3.3	Update GUI	D	
	A01.15.4	GUI, 3.3	Update GUI	D	
	A01.15.5	GUI, 3.3	Update GUI	D	
	A01.15.6	GUI		D	
	A01.16	GUI, 2.3	Init. Pause Flag	D	
	A01.20	GUI, 3.2.4	Process Endsim	D	
	A01.21	GUI, 3.2.4	Process Endsim	I, D	
	A01.22	GUI, 3.2.4	Process Endsim	I, D	
	A01.23	GUI, 3.2.4	Process Endsim	I, D	
	A01.24	GUI, 3.2.4	Process Endsim	I, D	
	A01.25	GUI, 3.2.4	Process Endsim	I, D	
	A01.26	GUI		D	
	A01.30	GUI		I	
A02.00		GUI		D	
	A02.01	GUI		D	
A03.00		2.1	Initialize Simlib	I	
	A03.01	3.2	Proc. Intern. Events	I	
B01.00		2.	Init. Simulation	A, I	
	B01.01	2.1	Initialize Simlib	I	
	B01.02	2.1	Initialize Simlib	I	

Key: A = Analysis, D = Demonstration, I = Inspection, K = Analogy

Table 2 continued. Requirements Traceability Matrix

Req. ID System Level	Req. ID Sub-System Level	DFD Identifiers	Module Name(s)	Verification Method	Test
	B01.03	2.1	Initialize Simlib	I	
	B01.04	2.1	Initialize Simlib	A, I	
	B01.05	2.2	Init. Delay Time	A, D	
	B01.06	2.1	Initialize Simlib	A, D	
	B01.07	2.1	Initialize Simlib	A, D	
B02.00		2.	Init. Simulation	I	
B03.00		3.2.4	Process Endsime	A, D	
B04.00		3.2.2	Process Arrival	I	
	B04.01	3.2.2	Process Arrival	I	
	B04.02	3.2.2	Process Arrival	I	
	B04.03	3.2.2	Process Arrival	A, I, D	
	B04.04	3.2.2	Process Arrival	A, D	
	B04.05	3.2.2	Process Arrival	D	
	B04.06	3.2.2	Process Arrival	D	
	B04.07	3.2.2	Process Arrival	D	
B05.00		3.2.3	Process Departure	I	
	B05.01	3.2.3	Process Departure	I	
	B05.02	3.2.3	Process Departure	A, D	
	B05.02.1	3.2.3	Process Departure	I, D	
	B05.02.2	3.2.3	Process Departure	I	
	B05.02.3	3.2.3	Process Departure	I	
	B05.02.4	3.2.3	Process Departure	I, D	
	B05.02.5	3.2.3	Process Departure	I	
	B05.02.6	3.2.3	Process Departure	I, D	
	B05.02.7	3.2.3	Process Departure	I, D	
	B05.02.8	3.2.3	Process Departure	A, D	
	B05.03	3.2.3	Process Departure	A, D	
	B05.04	3.2.3	Process Departure	I, D	
B06.00		3.2.2, 3.3.3	Arrival/Departure	I, D	
	B06.01	3.2.2	Process Arrival	D	
	B06.01.1	3.2.2	Process Arrival	I, D	
	B06.01.2	3.2.3	Process Departure	I, D	
	B06.01.3	1.	Evaluate Input	D	
	B06.02	3.2.2	Process Arrival	I, D	
	B06.02.1	3.2.2	Process Arrival	I, D	
	B06.02.2	3.2.2	Process Arrival	I, D	
	B06.02.3	3.2.2	Process Arrival	I, D	
	B06.02.4	3.2.3	Process Departure	I, D	
	B06.02.5	3.2.3	Process Departure	I, D	
	B06.02.6	1.	Evaluate Input	D	
	B06.03	3.2.2	Process Arrival	D	

Key: A = Analysis, D = Demonstration, I = Inspection, K = Analogy

Table 2 continued. Requirements Traceability Matrix

Req. ID System Level	Req. ID Sub-System Level	DFD Identifiers	Module Name(s)	Verification Method	Test
	B06.03.1	3.2.2	Process Arrival	I, D	
	B06.03.2	3.2.2	Process Arrival	I, D	
	B06.03.2	3.2.2	Process Arrival	I, D	
	B06.03.4	3.2.3	Process Departure	D	
	B06.03.5	3.2.3	Process Departure	D	
	B06.03.6	1.	Evaluate Input	D	
	B06.03.7	3.2.3	Process Departure	I, D	
	B06.04	1.	Evaluate Input	D	
B07.00		2.1	Initialize Simlib	I, D	
B08.00		2.1	Initialize Simlib	A, I, D	
B09.00		3.1	Proc. Ext. Events	D	
	B09.01	3.1.2	Process Insert	D	
	B09.02	3.1.1, 3.1.2	Proc. Insert/Delete	D	
	B09.03	3.1.1, 3.1.2	Proc. Insert/Delete	D	
B10.00		3.1, GUI	Proc. Ext. Events	I, D	
	B10.01	3.1.1, GUI	Process Delete	D	
	B10.02	3.1.1, GUI	Process Delete	D	
	B10.03	3.1.1, GUI	Process Delete	D	
B11.00		3.2.4	Process Endsim	A, I, D	
	B11.01	3.2.4, GUI	Process Endsim	D	
	B11.02	3.2.4	Process Endsim	I	
	B11.03	3.2.4	Process Endsim	A, D	
	B11.04	3.2.4	Process Endsim	A, D	
	B11.05	3.2.4	Process Endsim	A, D	
	B11.06	3.2.4	Process Endsim	A, D	
	B11.07	3.2.4	Process Endsim	A, D	
	B11.08	3.2.4	Process Endsim	A, D	
	B11.09	3.2.4	Process Endsim	A, D	
	B11.10	3.2.4	Process Endsim	A, D	
C01.00		3.2	Proc. Intern. Events	D	
C02.00		3.2	Proc. Intern. Events	D	
D01.00		2.1	Initialize Simlib	I	
	D01.01	2.1	Initialize Simlib	I	
	D01.02	2.1	Initialize Simlib	I	
	D01.03	2.1	Initialize Simlib	I	
	D01.04	2.1	Initialize Simlib	I	
D02.00		1., 2., 3., GUI		I	
	D02.01	1., 2., 3., GUI		I	
D03.00		2.1	Initialize Simlib	I	
D04.00		3.1.2	Process Insert	I	

Key: A = Analysis, D = Demonstration, I = Inspection, K = Analogy

Table 2 continued. Requirements Traceability Matrix

Req. ID System Level	Req. ID Sub-System Level	DFD Identifiers	Module Name(s)	Verification Method	Test
D05.00		3.1.1	Process Delete	I	
D06.00		2.1	Initialize Simlib	I	
	D06.01	2.1	Initialize Simlib	I	
	D06.02	2.1	Initialize Simlib	I	
D07.00		1., 2., 3., GUI		D	

Key: **A** = Analysis, **D** = Demonstration, **I** = Inspection, **K** = Analogy

APPENDIX D: Identified Test Cases

This appendix describes the principal test cases that have been identified for the ARMS software. They are based on the transactions identified above in Section 3.4. and target the main functionality of the ARMS system. In particular, they will verify that each queuing scenario (one - three) performs correctly, that the insert and delete functions perform correctly, and that the error checking of input parameters is handled correctly. This list, however, represents only a small subset of the actual tests that will be performed. Additional test cases and testing methods will be explained in the Draft Test Report.

Test Case 1:

Purpose: Verifying the description and logic associated with QS1.

Method: User selects QS1, enters a valid number of service stations, and presses start.
ARMS runs to completion using the logic defined by QS1.

Response: Results are displayed to output GUI.

Test Case 2:

Purpose: Verifying the description and logic associated with QS2.

Method: User selects QS2, enters a valid number of service stations, and presses start.
ARMS runs to completion using the logic defined by QS2.

Response: Results are displayed to output GUI.

Test Case 3:

Event: Verifying the description and logic associated with QS3.

Method: User selects QS3, enters valid numbers of service stations, and presses start.
ARMS runs to completion using the logic defined by QS3.

Response: Results are displayed to output GUI.

Test Case 4:

Purpose: Verifying the error checking of input parameters.

Method: User selects QS3, enters *invalid* numbers of service stations, and presses start.

Response: Error message is displayed to GUI. ARMS prompts user for re-entry until valid parameters are entered.

Test Case 5:

Purpose: Verifying the Delete function.

Method: User presses Pause button on progress GUI and highlights an aircraft from a queue (or an event from the event list or an aircraft from a service station -- all three will be tested). User presses button to delete and then presses Resume.

Response: ARMS deletes that item from the selected list and then runs to completion. Progress GUI shows that the item has been removed from the list.

Test Case 6:

Purpose: Verifying the Insert function.

Method: User presses Pause button on progress GUI, presses button to insert an event to the event list, and presses Resume.

Response: ARMS prompts the user for a description of the event, inserts the event to the event list, and then runs to completion. Progress GUI shows that the event has been inserted into the event list.

APPENDIX E: Simlib Flow Diagram

This flow diagram, which represents the next-event time-advance approach used by Simlib, was taken from Law and Kelton, page 10 [6].

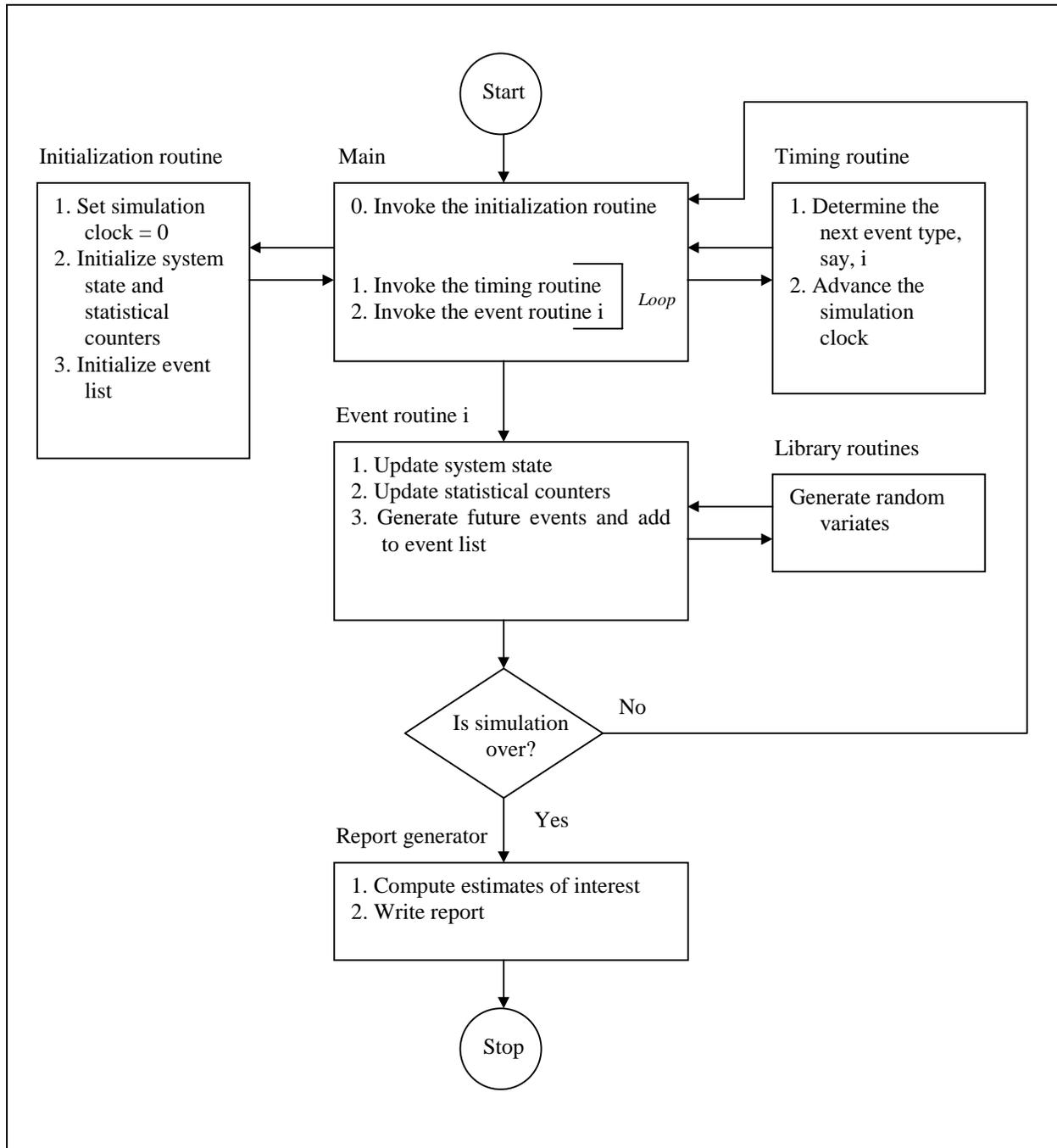


Figure 10. Flow of control for the next-event time-advance approach.

APPENDIX F: Simlib API

The following is a listing of the original Simlib API functions that will be used in the ARMS code [6].

```
/* This is simlib.h. */

/* Include files. */

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "simlibdefs.h"

/* Declare simlib global variables. */

extern int    *list_rank, *list_size, next_event_type, maxatr, maxlist;
extern float  *transfer, sim_time, prob_distrib[26];
extern struct master {
    float *value;
    struct master *pr;
    struct master *sr;
} **head, **tail;

/* Declare simlib functions. */

extern void  init_simlib(void);
extern void  list_file(int option, int list);
extern void  list_remove(int option, int list);
extern void  timing(void);
extern void  event_schedule(float time_of_event, int type_of_event);
extern int   event_cancel(int event_type);
extern float sampst(float value, int varibl);
extern float timest(float value, int varibl);
extern float filest(int list);
extern void  out_sampst(FILE *unit, int lowvar, int highvar);
extern void  out_timest(FILE *unit, int lowvar, int highvar);
extern void  out_filest(FILE *unit, int lowlist, int highlist);
extern float expon(float mean, int stream);
extern int   random_integer(float prob_distrib[], int stream);
extern float uniform(float a, float b, int stream);
extern float erlang(int m, float mean, int stream);
extern float lcgrand(int stream);
extern void  lcgrandst(long zset, int stream);
extern long  lcgrandgt(int stream);
```

APPENDIX G: GUI Prototypes

This appendix presents the current prototypes for the ARMS input, progress, and output screens. These prototypes are shown below in Figures 11 - 13, respectively.

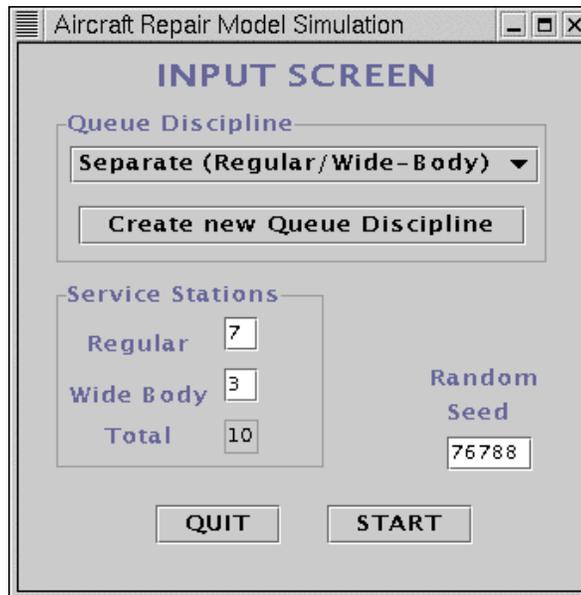


Figure 11. Input screen prototype.

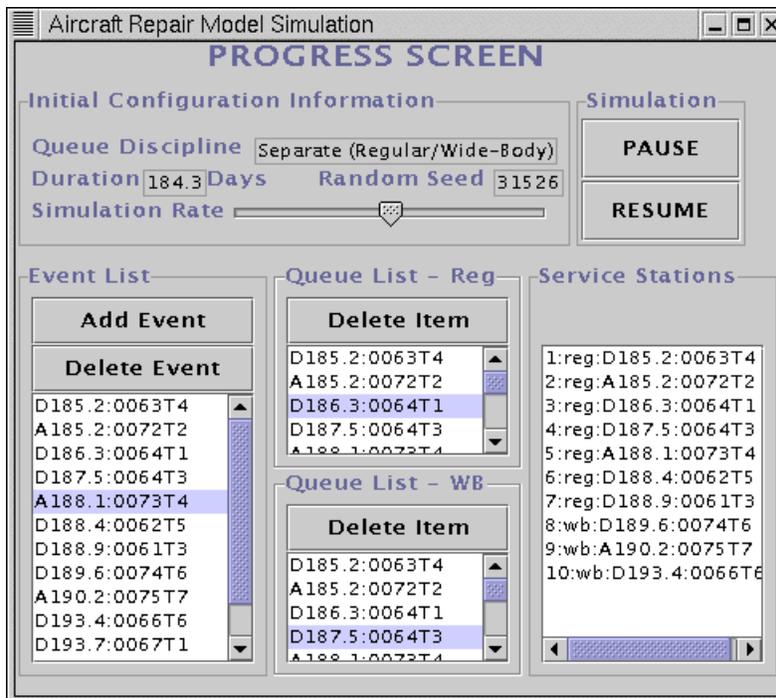


Figure 12. Progress screen prototype.



Figure 13. Report screen prototype

APPENDIX H: User-Provided Aircraft Parameters

This appendix describes the seven distinct aircraft types that will be used in the ARMS simulation. Specifically, the aircraft arrival and departure events shall be modeled according to the aircraft-specific attributes listed in Table 3 below.

Table 3. Aircraft Parameter Table and corresponding value definitions.

i	Aircraft Type	Wide Body	Number of Engines	Arrival Interval	Inspection Time		Repair Probability	Repair Time	Daily Cost
					Minimum	Maximum			
1	B707	No	4	8.1	0.7	2.1	30%	2.1	2.1
2	B727	No	3	2.9	0.9	1.8	26%	1.8	1.7
3	B737	No	2	3.6	0.8	1.6	18%	1.6	1.0
4	B747	Yes	4	8.4	1.9	2.8	12%	3.1	3.9
5	DC8	No	4	10.9	0.7	2.2	36%	2.2	1.4
6	DC9	No	2	6.7	0.9	1.7	14%	1.7	1.1
7	DC10	Yes	3	3.0	1.6	2.0	21%	2.8	3.7

Column	Description
I	An enumeration reference for the different sets of configuration data (<i>aircraft type</i>).
Aircraft Type	Descriptive name that has a unique association with i .
Wide Body	The aircraft’s designation as a wide-body aircraft type (<i>yes</i>) or a regular aircraft type (<i>no</i>).
Number of Engines	The total number of engines this aircraft type has.
Arrival Interval	The mean number of days between successive arrival events (<i>used in conjunction with the exponential distribution to predict arrival times</i>).
Inspection Time, Minimum	The minimum amount of days it takes for this aircraft to have one engine inspected (<i>used in conjunction with a uniform distribution to predict inspection times</i>). This time will be halved for all non-initial engine inspections.
Inspection Time, Maximum	The maximum amount of days it takes for this aircraft to have one engine inspected (<i>used in conjunction with a uniform distribution to predict inspection times</i>). This time will be halved for all non-initial engine inspections.

Repair ProbabilityThe individual probability that any single engine for this aircraft will require a repair of some kind. This probability will be halved for all non-initial engine inspections.

Repair Time.....The number of days that will be needed to complete a repair.

Daily Cost.....The cost per day for this aircraft to remain down. The aircraft is considered to be down the entire time it is in a queue or a service station. (*in \$10,000 US Dollars*).