Onerous Android Application

Architecture/Design Documentation

Table of Contents:

1. Introduction
2. Specification
3. Logical View
   1. High-Level Design
   2. Mid-Level Design
   3. Low-Level Detailed Design

1 Introduction

This document describes the architecture of the Onerous App developed by the Modelling and Simulations Group of Cranfield University based at the Defence Academy Shrivenham. The Onerous App will be used to teach and demonstrate students about stochastic discrete event simulation and their use in operational analysis. It models helicopters stationed at the frontlines with engines that need to be regularly maintained.

The purpose of this document is to describe the architecture of the Onerous App clearly and accessibly to meet a few requirements:

1. Reduce time taken for future developers/programmers to understand the code at a level needed to be able to maintain, modify, enhance and perhaps even port the app to another platform such as iOS.
2. Allow students to explore and understand how the simulation is built and how it works.
3. Provide multiple views to support various different specialised interests in accordance with IEEE std. 1471.

What this document will not do is explain the Android or Java framework. Understanding of either should not be necessary in understanding the architecture of the application. However, competence with both Java and Android will be necessary for code maintenance or modification.

It will not detail how the individual methods or functions operate but will summarise their jobs and input/outputs. If further technical detail concerning the implementation of individual methods and functions are required, the commented source code can be found in the project files.

2 Specification

The Onerous Android App based on the original Onerous V2.0 desktop application, which will be referred to as Onerous 2008. Onerous 2008 is a desktop application built using Microsoft Visual Studio 2008 using the Visual Basic language.

The objective was to port Onerous 2008 so that it could run on tablets running the Android OS. The specific model that was available at the SSEL was the Samsung Tab 2 10.1 running android version 4.1.1 (16 Jellybean). All functionalities had to be retained and the app UI had to be updated to one that is geared towards tablet usage with a touchscreen as opposed to Desktop usage with mouse and keyboard.

For further information on the specification and details leading to the creation of the original Onerous 2008, file OA 1229 and the presentation file FMS\_SSE\_16\_7\_Onerous\_Responsibility can be referred to.

System Behaviour overview

This section aims to give a brief summary of how the app should behave and what it functions it should be fulfilling so there is context for the underlying logic and architecture.

The Onerous apps functions can be split up into three core functions:

1. Viewing the current simulation run settings and changing them if desired.
2. Running the simulation with the option to view the events animated and visually represented on an activity cycle diagram, or to turn them off and complete the simulation instantly.
3. The ability to view statistics and graphs relating to each entity and its state e.g. the amount of helicopters that are operational.

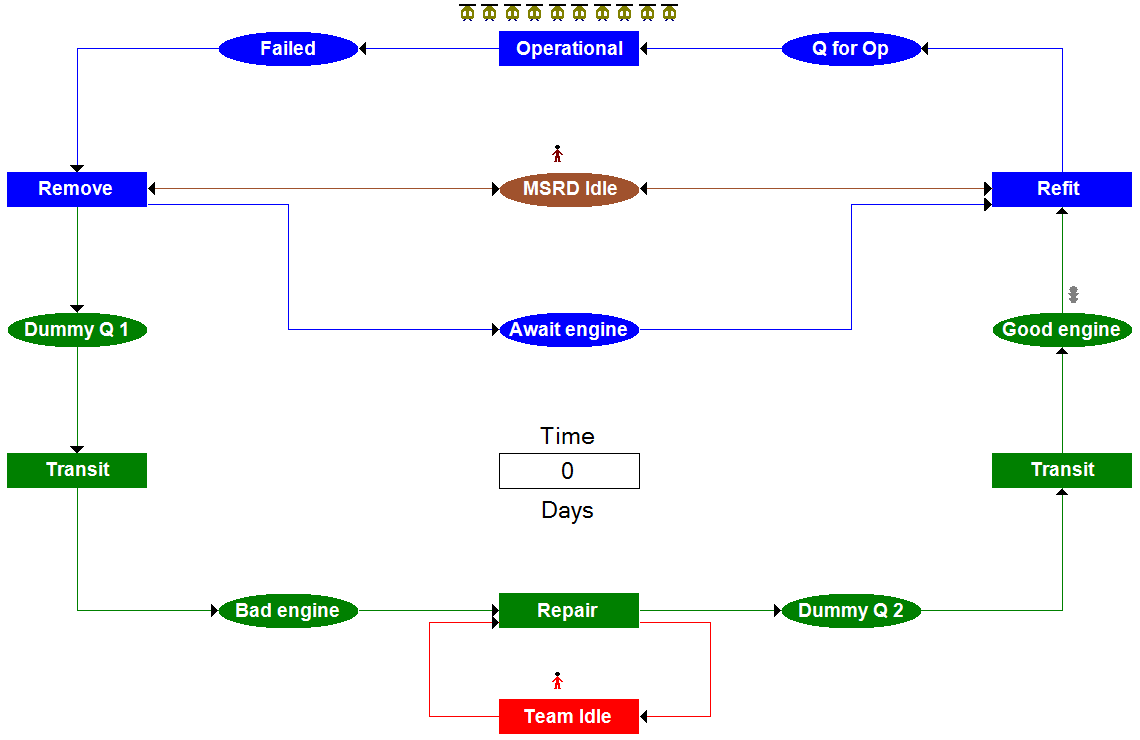


Fig 1. Activity cycle diagram of Onerous 2008

Fig.1 shows the activity cycle diagram that illustrates the model the simulation will be using for the simulation. There are six main activities that occur: helicopters operational (until engine failure occurs), engine removal, transit to manufacturer, engine repair, transit to operation and refit time.

**3 Logical View**

The logical view describes the components of the app and how they work together to provide the overall functionality of the application. The logical view will begin at a high level view and work its way down to low level. High level will explain and show how the major components work together at an architectural level and will then progress towards more detail by showing smaller components and then eventually to individual methods and functions, but any further detail will require a look at the source code and the comments included with it.

**3.1 High-Level**

The three core functions of the Onerous 2008 application has been split up into three different UI screens for the android application. They have been chosen to be logically intuitive for the user and we can use these as the major components that build up the high level architecture view.

**Memory**

View current run settings.

Edit run settings.

Restore default settings.

**Variables**

Show statistics of each activity.

Show graph of each activity.

**Results**

Run simulation.

View animation/graphics.

Toggle graphics on/off.

**Simulation**

*Read*

*Write*

*Read*

*Write*

*Read*

***Fig.2 Diagram showing major components of Onerous Android App 2017***

The function of each component is shown in the diagram and what kind of interaction the component has with the memory. The read/write interactions are expanded on further below:

* Variables
  + Read: Reads the current simulation settings stored in the memory to display to the user and accesses it every time the settings are edited to update the display for the user.
  + Write: Writes the users inputted changed settings into the memory. This overwrites the previous settings.
* Simulation
  + Read: Reads the simulation run settings and runs an instance of the simulation accordingly. Reads run variables used to keep track of entities and other values, such as runTime*,* of the specific instance of the simulation that is running.
  + Write: Writes any changes to the run variables as the instance of the simulation progresses. Stores data that will be used to generate statistics and graphs as the simulation progresses. Calculates various statistics and datasets at the end of the run and stores it in the memory.
* Results
  + Read: Reads the data generated and stored during the simulation run that to generate a table of statistics and a graph of each node of the activity cycle diagram.
  1. **Mid-level View**

Variables

1. onCreateView()
   1. Inflate variables tab layout file (layout\_fragment.xml)
   2. Initialise buttons onClick functions.
   3. Setup spinners and what happens when each item is selected: focus (nextFocusDown and nextFocusForward) behaviour, textView text and editText/TextView visibility.
2. updateVarBtn onClick
   1. Calls updateVars()
      1. Changes the simulation run settings saved in the memory to values entered by the user.
      2. Calls refreshCurrentVarValues()
         1. Refreshes the display showing the user the current simulation run settings with the newly updated settings.
3. restoreDefaultBtn onClick
   1. Sets the simulation run settings saved in the memory with the default values.
   2. Calls updateVars()
      1. Refreshes the display showing the user the current run settings with the newly updated default values.

Simulation

initVariables()

initList()

resetAllStats()

resetAllGraphData()

Set DaysPassed to simTime

endFlag = true?

getNextEvent()

if simTime > runTime

then call next event function e.g.

Events.engineFails()

else set endFlag to true

end if

If (graphicsOn = true)

Set DaysPassed to simTime

simTime > runTime?

Set simTime to runTime

Set DaysPassed to simTime

finaliseStats()

Return

Yes

No

Yes

No

**Class references**

**ActivityDistribution**

The ActivityDistribution class is a user defined type that holds values that will be used to calculate how long each ‘activity’ (e.g. engine removal, engine refit, bad engine transit etc.) takes.

**EventData**

The EventData class is a user defined type that is used to hold two bits of data: the time of an ‘event’ and the ‘kind’ of event.

**Events**

The class Events contains the methods used for performing the actions required when a certain scheduled event occurs.

**MainActivity**

The MainActivity class is the very first class that launches when the app is launched. It is responsible for initialising the UI view and setting up the tabbed fragments that will hold the majority of the UI elements.

**PlaceholderFragment**

Tabbed placeholder fragment that can hold the place of a tab in a tabbed activity.

**ResultsData**

The ResultsData is a user defined type that is used to hold data and statistics when the simulation runs.

**ResultsFragment**

The ResultsFragment class initialises the UI elements of the ‘Results’ tab, and contains the methods required for populating the results table and the graph with the appropriate data.

**SimAnimations**

The SimAnimations class contains all the methods for performing the various animations when visually displaying the simulation running.

**SimMethods**

The SimMethods class contains much of the backbone for actually simulating the scenario.

**SimulationFragment**

The SimulationFragment class initialises the UI elements of the “Simulation” tab.

**Stats**

The Stats class contains the methods used when generating the simulation run stats.

**Variables**

The Variables class contains all the global variables that the application needs to access from multiple other classes. Many of these global variables are also the variables that control the simulation run settings.

**VariablesFragment**

The VariablesFragment class initialises the UI elements of the “Variables” tab which allows the user to change many of the simulation settings.