Assignment 1 Report

18439731 - JACOB JONAS
SOFTWARE ARCHITECTURE AND EXTENSIBLE DESIGN - COMP3003

Multithreading Explanation:

Classes responsible for starting threads:

- App
- AirportScheduler
- Airport
- Plane

The App class begins an AirportScheduler thread for each airport. This then in turn starts the Airport thread it will communicate with. The Airport thread starts the "saed_flight_request" process and when a flight request is produced, it gets placed into a BlockingQueue for requests.

The AirportScheduler thread will take items out of this blocking queue as well as a Plane from a separate BlockingQueue and submit a Plane thread into a thread pool for execution.

The Plane thread is responsible for moving the plane icon to its destination and once it has, it tells the destination airport it has landed by calling a method on the airport which creates a Runnable task that starts the "saed_plane_service" process. This Runnable is then submitted to a thread pool for execution. Once that task has finished, the Plane is placed into the BlockingQueue as it is now ready to serve another flight request.

Thread Communication:

The main threads that communicate are the Airport and Airport Scheduler threads using BlockingQueues. The Airport thread acts as the producer as it generates the flight requests for the AirportScheduler thread to then act upon, making it the consumer. There are two BlockingQueues in the Airport thread:

- requests
- planes

"requests" is where the generated flight requests get placed ready for the AirportScheduler thread to take and give to a Plane. "planes" is where the planes that are at that airport are stored. For every flight request taken by the AirportScheduler thread, a Plane is also taken to perform that flight request.

The Plane thread also communicates with the Airport thread as it will notify the relevant airport when it has landed so that the Airport can submit a task to a thread pool to execute the service.

Shared Resources:

Apart from the BlockingQueues, the other important shared resources are the variables that hold the stats to be displayed in the GUI. These being the total flights completed, current flights in progress, and current services in progress.

Each of these variables has an associated mutex object which any operation involving the variable is synchronised against. Because each variable is only synchronised against its own mutex and not multiple, there is no chance of a deadlock as it does not need to wait of the release of another resource before performing any operation.

Ending Threads:

Once the end button is pressed, the App class interrupts all of the AirportScheduler threads it started at the beginning, as well as shutting down the two thread pools.

Once an AirportScheduler thread is interrupted, it subsequently calls interrupt on the Airport thread it started. This propagation means all associated threads are interrupted.

The two thread pools are responsible for all flights and all services so those threads are interrupted as well.

This is all the threads that are present in the simulation, meaning it has successfully ended.

Modified Version:

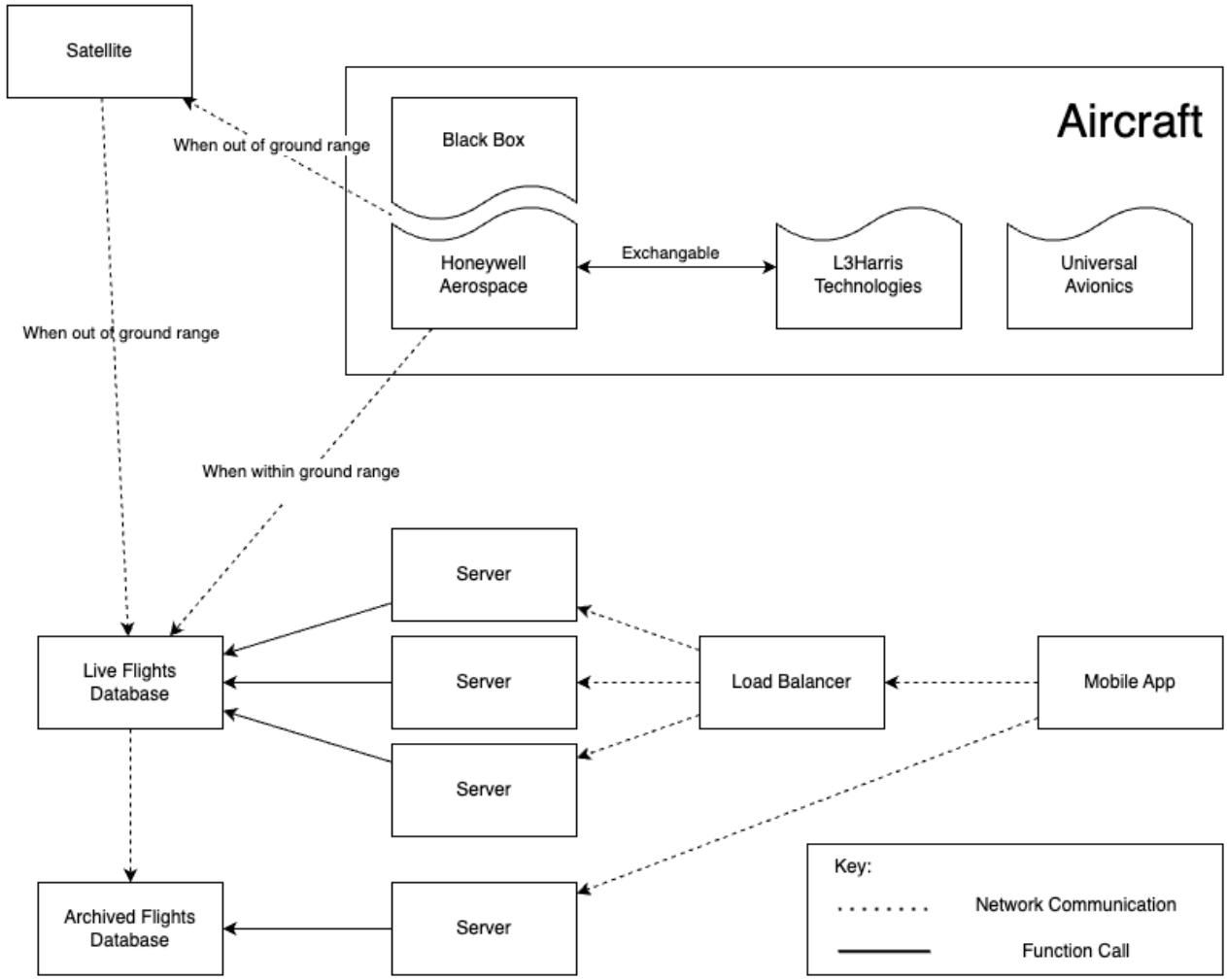
Challenges:

- If the data is coming straight from the aircraft itself, then because there are different black box manufacturers, the data may not be homogenised and as such needs specific handling per manufacturer
- As it will be a world wide flight tracking system, there would need to be multiple servers that airplanes send the data to that then need to be synchronised so that when a plane moves out of range of one and switches to another, there is minimal, if any, data loss.
  - Planes also travel across vast areas where there is no land within communication range, meaning satellites would be needed.
- There is potential for data loss in transmitting data down from an aircraft to a server then across a network to users' mobile devices.
- Load balancing is also a factor to consider as many, many people may want to view flight data concurrently.

Non-Functional-Requirements:

- Visualised flight data should be no more than 1 minute behind the actual flight position because it needs to be an accurate, real time representation of the flight and anyone viewing the data can make well informed decisions based on it, whether it be a user viewing a family member's flight or someone monitoring flights to report any anomalies.
- It will need to handle a large amount of concurrent users because the user base is very large and they should all be able to view real-time data.
- The reliability of the system needs to be extremely high, as large amounts of missing data would not be ideal for a real-time tracking system.
- It will need to integrate with all black box manufacturers as we want to be able to track all flights not just the ones from big companies.
- It would need to be in multiple languages because it is intended to be made available all around the world.
- The connection from the aircraft to the ground would need to be very secure as to prevent malicious actors from intercepting and providing false data to hide a hijacking, or sending commands to the aircraft itself.

Diagram:



Starting with the aircraft, because, black box manufacturing isn't 100% homogenised, there will need to be some specific modules designed to integrate with each different manufacturer. These modules will format and stream the data to a database. The issue with having modules on the aircraft itself, is that deployment to each aircraft is something that needs to be thought about, as well as it's integration with current systems with minimal interference.

This database holds the information for the live flights which will be served to the mobile app through requests that are distributed using a load balancer. This is to ensure that there is enough capacity to serve all users who may be requesting real time flight data.

There will multiple instances of the bottom half of the diagram all around the world to serve that geographic location. These separate databases will also need to be synchronised so that when Aircraft changed from one to another, there is no data loss.

Sometimes there might be a ground server within range of the aircraft, in which satellite will be required to continue data transference.

There are two separate databases, one for live flights and one for archived flights. This is because we want to spread the load where we can and also wanting to view archived data isn't as time important as real-time data so that can be handled by a separate server. Once a flight has been completed, its data will be transferred from the live flights DB to the archived flights DB.