CMPE322/326 Assignment 2 - Concrete Architecture

FlightGear

Submitted By: Group 20
Jasper Lim, Laurie Yuzichuk, Jordan Herzstein, Campbell Love, Christopher Seguin Bianchi, Cameron Bennett

Abstract

This report aims to cover the concrete architecture of FlightGear, by analyzing the source code, and comparing discrepancies between the conceptual architecture and inner architecture. Previously, it was established that FlightGear factors a high degree of concurrency and compartmentalization of components. The Inner Architecture is focused primarily on the Flight Dynamics Model (FDM), and the Inner Architecture of the FDM specifically will be of additional focus for this report. At the top level though, FlightGear's major components also consist of the Viewer and GUI, Aircraft, Autopilot, Environment and Scenery, Input and Systems, Network, Sound, Addons and Scripting modules.

Observing the concrete architecture of the FDM, we can find several dynamics models that are used for different aircraft. This includes JSBSIm, YASIm, LaRCsim, and UIUCModel. AIWake aids JSBSim in simulating wake turbulence. ExternalNet and External Pipe Integration let FlightGear communicate with external software and hardware. Special Purpose Integration are additional features specific to certain aircraft or simulation variables.

There are also several header files that define or control properties and other variables in the simulation. Flight.cxx is the most important source code file, with core algorithms and logic located here.

The derivation process for this was motivated by considering our previous work on conceptual architecture, approaching the source code with design goals to consider what aspects were important from the perspective of the developers, and lastly facilitated using SciTools' Understand software to aid in examining the source code and the relations between each component.

Upon reflexion analysis, it was discovered there were some divergences with the conceptual architecture in mind. Some of these discrepancies were relatively minor, such as the partitioning of ATC Simulation into ATC and Radio, whereas there were some surprising discoveries, such as the interactions concerning the Network Manager, which goes to a new 3rdparty module. In fact, much of the data flow goes to the 3rdparty component, likely as a go-between for the Server and Client in the conceptual architecture.

In regards to the reflexion analysis of the FDM, there were some fundamental divergences with our understanding. Src as a whole actually corresponds to what was defined as the FDM Server in the conceptual architecture, whereas Flight Dynamics Calculations corresponds to the FDM subsystem was defined as.

Introduction

FlightGear is a free, open-source flight simulator in development since 1997, supported on multiple OS's including Windows, MacOS, Linux, IRIX, and FreeBSD [1]. FlightGear has been used in academic research, education, training, and for recreational purposes [1].

Previously, we covered the conceptual architecture for FlightGear. It was determined that the implementation uses a combination of High-Level Architecture, Object-Oriented, Repository, and Client-Server architectures. Organized such, FlightGear centers around the Flight Dynamics Model (FDM) Server, which calls upon its various components to handle the bulk of the simulation tasks, and communicates through the network to its connected client machines where input and output are exchanged.

It was concluded that FlightGear's open-source nature basically necessitated and greatly benefitted from a high level of concurrency and compartmentalization of components, improving performance, easing development and maintenance, and facilitated a logical conceptual structure to FlightGear. Now, we can observe the actual, code-level implementations of these concepts and the FlightGear structure.

High-level Concrete Architecture Subsystems

While investigating the concrete software architecture of FlightGear, it's crucial to highlight key components that form the backbone of the system and contribute significantly to its functionality. Among the components that can be found in Figure 1, which is a diagram of the concrete architecture of the whole FlightGear system, the following are important to mention:

FDM (Flight Dynamics Model): This is a central subsystem responsible for simulating aircraft behavior during flight, encompassing aspects of aerodynamics, propulsion, mass distribution, and control systems.

Viewer and GUI (Graphical User Interface): These components handle the graphical representation of the simulation, providing the user interface for interaction and visualization of the simulated environment.

Aircraft: This component represents the various aircraft models available in FlightGear, each with its specific characteristics and behaviors simulated within the FDM.

Autopilot: An essential component that manages automated control functions during flight, enhancing the realism of the simulation and providing options for autopilot-enabled aircraft.

Environment and Scenery: These components deal with the simulation of environmental factors such as weather conditions, time of day, and terrain features, contributing to the immersive experience of the simulator.

Input and Systems: Input systems capture user commands and translate them into actions within the simulator, while systems components manage various subsystems and functionalities of the simulation.

Network: Facilitates networking capabilities, allowing for multiplayer interactions and communication between multiple instances of FlightGear.

Sound: Responsible for audio output, providing realistic sound effects related to aircraft operations, environment, and interactions.

Add-ons and Scripting: These components extend the functionality of FlightGear through custom add-ons, scripts, and extensions, allowing for modularity and customization of the simulator.

By understanding how these components interact and collaborate within the architecture, we can generate insights into the functionality of the FlightGear system. In the next sections, we will investigate deeper into the FDM subsystem, focusing on its functionality and contributions to the overall simulation experience.

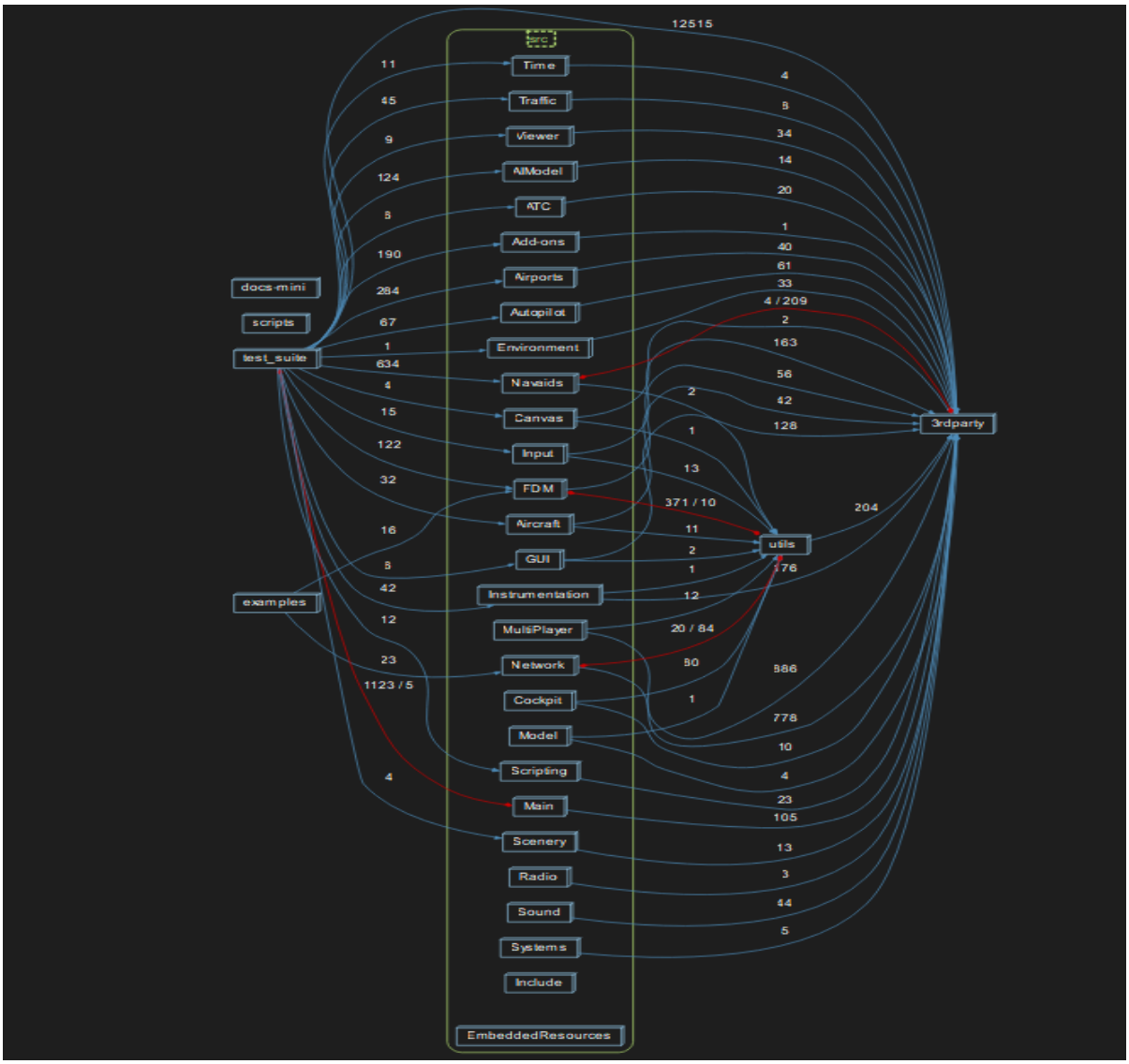

Figure 1- Concrete Architecture diagram.

Inner Architecture of FDM

To better understand the architecture of FlightGear, the group decided to focus on the Flight Dynamics Model subsystem as the group had previously looked at the subsystem in detail in A1.

After looking at the source code via *Understand*, the group decided that the components involved in the conceptual architecture involved A Base flight physics model, an environmental modelling component, and a User Interface component that would deal with what the user sees based on which aircraft they chose. When looking at the Concrete Architecture and how its built, each of the conceptual architecture groups are hit, however the concrete architecture has more subgroups.

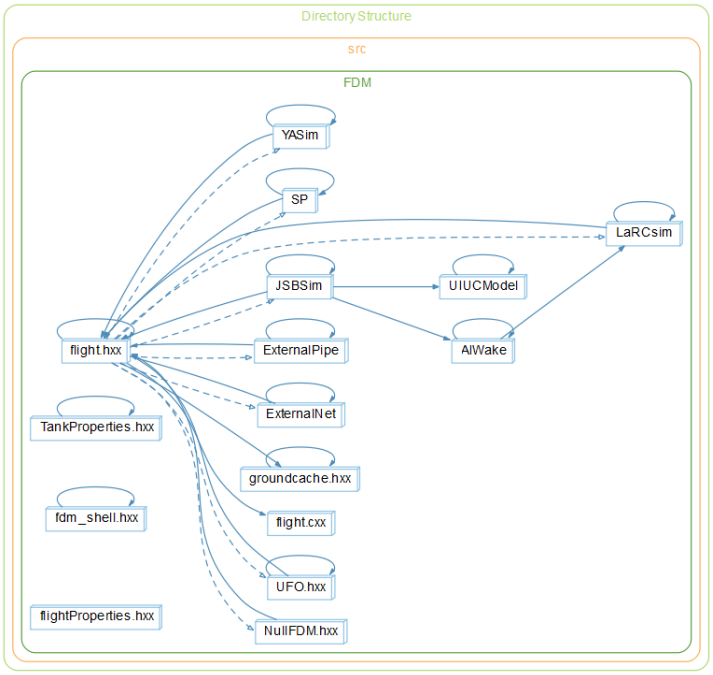

Figure 2: Graphical View of the FDM architecture and all its components.

Here is a quick description of each component and what it does:

JSBSim (The JavaScript Basic Simulation): JSBSim serves as the primary flight dynamics model within FlightGear, simulating aerodynamics, propulsion, and control systems of aircraft. It provides a flexible and customizable framework for accurately modeling aircraft behavior.

YASim (Yet Another Flight Dynamics Model): YASim offers an alternative FDM option within FlightGear, focusing on simplicity and ease of use. While less complex than JSBSim, YASim provides a straightforward approach to aircraft simulation, particularly suitable for beginners and lightweight aircraft.

LaRCsim: Developed by NASA, LaRCsim is a high-fidelity flight dynamics model capable of simulating advanced aircraft configurations and maneuvers. It offers detailed aerodynamic modeling and advanced control system simulation, catering to research and development applications. LaRCSim is currently inactive in the FDM since 2000 and helped develop UIUCModel's code.

UIUCModel (University of Illinois Urbana-Champaign Model): UIUCModel provides another option for simulating aircraft dynamics within FlightGear. Developed by the University of Illinois Urbana-Champaign, this model offers unique capabilities and features tailored to

specific research requirements. UIUC is no longer built on default due to it's lack of ground interaction and will only be accessed if chosen by a user. [2]

AIWake: AIWake is a feature that helps simulate wake turbulence in the JSBSim, using an AI-Controlled aircraft. For better understanding, wake turbulence refers to turbulent airflow behind significantly large aircraft, which can affect handling of trailing aircrafts. [3]

ExternalNet and External Pipe Integration: ExternalNet and External Pipe facilitate communication between FlightGear and external software or hardware components. They enable data exchange with external flight dynamics models, control systems, or simulation environments, enhancing the versatility and extensibility of FlightGear. [2]

SP (Special Purpose) Integration: SP subsystems encompass specialized features or enhancements tailored to specific aircraft or simulation requirements, including custom aerodynamic models, control system algorithms, or environmental effects. Integration with JSBSim or other flight dynamics models ensures compatibility and seamless operation within FlightGear. [2]

TankProperties.hxx, fdm_shell.hxx, groundcache.hxx: Each of these header files define properties and interfaces within aircraft models or the surrounding environment. TankProperties focuses on fuel tanks within aircraft models and how they differ. Fdm_shell defines the interface for external FDM's in FlightGear. Groundcache focuses on terrain caching and management in FlightGear. [3]

Flight.cxx: flight.cxx is the main implementation of the FDM. All the core algorithms and logic responsible for simulating aircraft behaviour during the flight are in this document. All of the previous source files aside from TankProperties, fdm_shell, and groundcache are directly correlated to this file. [3]

flightProperties.hxx: flightProperties.hxx is a header file defining properties and configurations related to the flight simulation. Things like Weather conditions, time of day, aircraft selection, and other parameters are all done in this document. [3]

All these components work together to create FDM. Looking at Figure 1, most of the components connect to flight.cxx, with the only outliers being the header files. JSBSim and YASim work together to create the actual physics of the flight, using JSB as the main source and YASim to help with more experienced/advanced aircrafts. [2]

Derivation Process

The group examined the FlightGear source code and dependencies to learn about the derivation process of its concrete architecture. The source code was provided to us from a google drive zip file, though the most recent versions of the code can be cloned from https://git.code.sf.net/p/flightgear/flightgear, and FlightGear documentation provides all the necessary git repositories to build FlightGear from source [4]. We set up design goals that would align with the intended purpose of the system. Firstly we wanted to consider performance as the system needs to do many operations in parallel such as in the FDM. We also wanted to consider the maintainability of the project, as it relies on many contributors to update the existing source code over time. With these design goals in mind, we examined the source code of FlightGear in the src/ directory using the Understand tool from SciTools to see the dependencies within the source code, as seen in Figure 3.

Figure 3: src Dependencies for FlightGear from flightgear git repository

After the subsystems were established form conceptual architecture, which architecture components match each subsystem in the conceptual architecture would need to be deciphered. This was dome with the help from the src dependencies generated by Understand as seen in Figure 3, in addition to the source code itself, documentation, and file/folder names. Mostly, architectures in Figure 3 corresponding to each subsystem name would be used, as a rudimentary example FDM architecture would match FDM conceptual architecture. The group also would keep in mind the architectural styles as outlined in conceptual architecture: High Level Architecture, Model View Controller Architecture, and Client Server Architecture. The Understand dependencies graph would also help identify any unexpected new elements within the concrete architecture to further compare with the conceptual architecture.

Investigations of the Discrepancies

A. Reflexion Analysis of High-Level

A comparison was made using Figures 1 and 3, which can be seen earlier in the report and to the right respectively. Most components within the FDM Server, such as Rendering and Scenery Update, still do not connect between each other and therefore there are no divergences in that regard. However, these subsystems have been broken down into more precise components.

Additionally, one key difference between the conceptual and concrete architecture is that instead of relaying data flow through a Network Manager, each subsystem within the server directly with 3rdparty, which relays everything to the user.

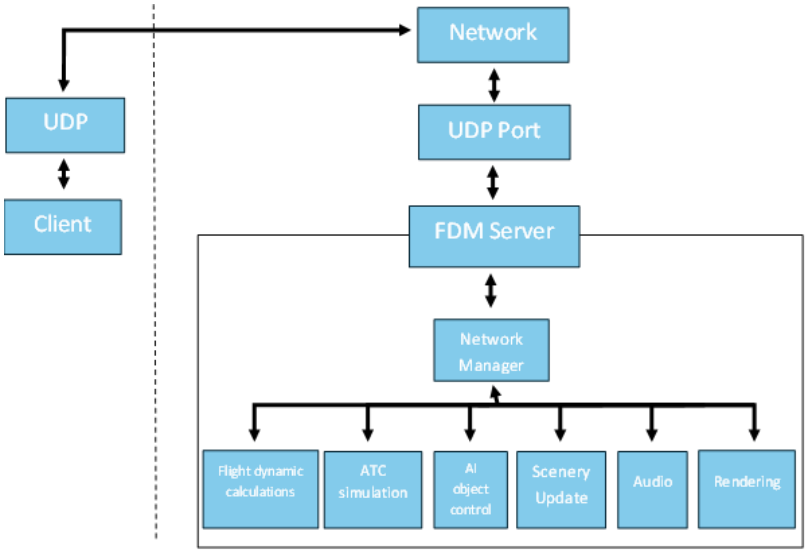

Figure 3: Conceptual Architecture of FlightGear from a previous report

Absences/Divergences with Network and UDP Port

Network and UDP Port correspond to 3rdparty. However, instead of having all the subsystem data relayed via a single component, like the Network Manager for instance, each subsystem directly connects to it. This results in many divergences between all the various subsystems.

Absences/Divergences with FDM Server

FDM Server from the conceptual architecture correlates directly to src with no divergences besides with 3rdparty as mentioned above.

Absences/Divergences with Network Manager

Network Manager maps to Network, Input and Systems, however does not communicate between the server and the network. Instead, all the data from the subsystems of src flow directly to 3rdparty, while the Network and Input components nested in src do the same.

Absences/Divergences with Flight Dynamics Calculations

Flight Dynamics Calculations directly correlate to FDM and Model with no other divergences besides with 3rdparty as mentioned above.

Absences/Divergences with ATC Simulation

ATC Simulation is broken down into two subsystems, ATC and Radio, with no other divergences besides with 3rdparty as mentioned above.

Absences/Divergences with AI Object Control

AI Object Control is broken done to AIModel and Traffic, as well as Autopilot. The only divergence is with Network Manager/3rdparty as mentioned previously.

Absences/Divergences with Scenery Update

Correlates to Scenery, Environment, Time, and Airports. The only divergence is with Network Manager/3rdparty as mentioned previously.

Absences/Divergences with Audio

The Audio component from the conceptual architecture directly correlates to Sound with no other divergences besides with Network Manager/3rdparty as mentioned above.

Absences/Divergences with Rendering

Rendering is split into multiple components that each handle updating various visual aspects as the user interacts with the aircraft's controls and in flight. This includes the Cockpit, GUI, Viewer, Navaids, and Aircraft components. The only divergence within these subsystems is with how data is transferred directly to 3rdparty as mentioned previously.

B. Reflexion Analysis of FDM

In our original conceptual architecture, we originally thought that FDM included and controlled components such as ATC simulation, scenery updates, and rendering.

However after further inspection of the concrete architecture using Understand, FDM Server in our conceptual architecture maps to src in FlightGear's concrete architecture, and Flight Dynamics Calculations is actually what the FDM subsystem is.

Sequence Diagrams

Below is a sample sequence diagram for the simple use case of starting an aircraft. It diverges from the sequence diagram created for the conceptual architecture for reasons described previously in the Reflexion Analysis.

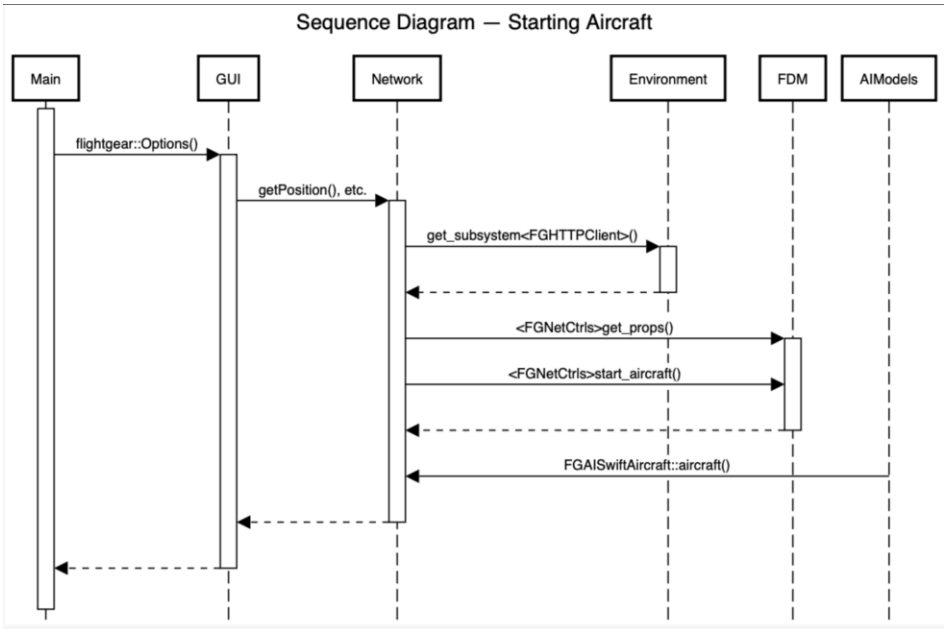

Figure 4: Sequence diagram for starting an aircraft.

In addition, a sequence diagram for an aircraft in flight was created. This again differs from the diagram created for the conceptual architecture as before.

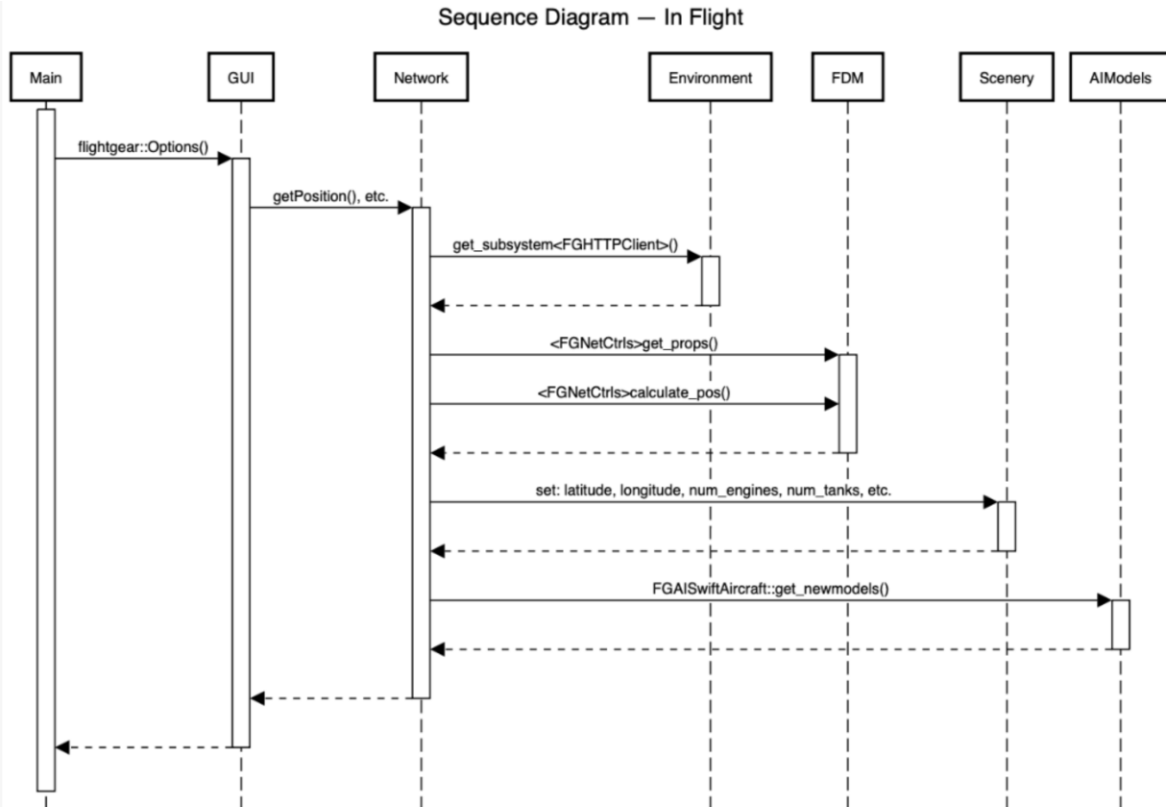

Figure 5: Sequence diagram for aircraft currently in flight.

Conclusion

The group conducted an analysis of the concrete architecture for FlightGear throught the use of SciTools Understand; the main focus of the analysis was on the FDM subsystem. Through mapping the dependencies of the 'src' directory as well as the FDM sub-directory using Understand, convergences and divergences between the conceptual and concrete architecture were found. Using the FlightGear repository source code, a dependency graph was produced which helped recognize gaps as well as similarities between the conceptual and concrete architecture. Reflexion analysis would help identify the most significant gaps.

Lessons Learned

Utilizing the Understand tool, the FlightGear src dependencies and FDM subsystem dependencies were visualized. The lines on the Understand tool were somewhat difficult to understand as they did not show the exact dependencies. Instead, there was a separate window with the list of dependencies which could be loaded for each program, component, or architecture, which took some time to learn. Additionally, it was difficult to visualize some elements in a flexible way. For example, visualizing the whole repository would result in far too many elements, and being able to easily reorient them would have made it easier to create clear diagrams.

Glossary

AI: Artificial Intelligence

ATC: Air Traffic Control

DMA: Direct Memory Access

FDM: Flight Dynamics Mode

Reflexion Analysis: Comparison of concrete architecture to proposed conceptual architecture

Bibliography

[1] "FlightGear About," FlightGear, [Online]. Available: https://www.flightgear.org/about/. [Accessed 19 02 2024].

[2] "Flight Dynamics Model - FlightGear wiki," wiki.flightgear.org, [Online]. Available: https://wiki.flightgear.org/Flight_Dynamics_Model. [Accessed 16 March 2024].

[3] "FlightGear forum," OpenSource, [Online]. Available: https://forum.flightgear.org/viewtopic.php?t=15586. [Accessed 16 March 2024].

[4] "Howto:Get_Local_Copies_of_Flightgear_Source_Code," [Online]. Available: https://wiki.flightgear.org/Howto:Get_Local_Copies_of_Flightgear_Source_Code. [Accessed 25 March 2024].