CISC 322/326 Assignment 1
Conceptual Architecture of FlightGear

February 18, 2024

Group 3
Team Name: Project Flight Forge

Name & Student Numbers:

Lucas Coster (20223016)

Divaydeep Singh (20143859)

Antonio Sousa-Dias (20289507)

Alan Teng (20312270)

Donovan Bisson (20230334)

Zhongqi Xu (20200067)

Table of Contents

1. Abstract

The report examines the conceptual architecture for the open-source software FlightGear. Based on the examination of several documents including the GitHub source code, books, and scholarly articles the FlightGear architecture was found. It was concluded that FlightGear uses a modularized architecture which contains a main module which provides a connection between the rest of the architecture. Our report dives into the relationships between all these architectures along with details on how data flows through the system. Who oversees upgrading the architecture and an outline of the evolution process the software has seen. The report also provides a few examples of use cases and sequence diagrams to give a strong outline of how the system would communicate and run through processes.

2. Introduction

FlightGear is a free flight simulation software. Initially released in in 1996, FlightGear is a well-equipped flight simulator with many features, including real-time weather updates, third party aircrafts and environments, multiplayer, and scenario simulation. In this paper the conceptual architecture for the software is outlined.

Four main modules of the architecture are identified: aircraft, simulation engine, main build, and terrain model. These each have their own submodules which concurrently update each other in an information flow directed towards the simulation module and finally the main build, to ensure accurate simulation. The support of multiplayer requires a network module to provide data sync between users. The team consulted multiple resources, including FlightGear's own documentation, as well as research paper and books on general flight simulation, in order to build a model of the conceptual architecture proposed in this paper.

3. Derivation Process

To familiarize ourselves with the FlightGear architecture the group each took responsibility for one of the provided resources and then came back together for a meeting where information was shared, and an architecture was designed.

The first two resources were both books and gave a general insight into the organization of simulation software these books provided insight into flight physic calculations and

descriptions of a typical conceptual architecture [4] [5]. The books did not explain how FlightGear itself was built but gave a strong base of information on which the final architecture was built.

The next resource the group studied were the scholarly journal articles which were found through the IEEE XPLORE journal and gave a wide breadth of information about simulation in general containing valuable images such as the one shown below.

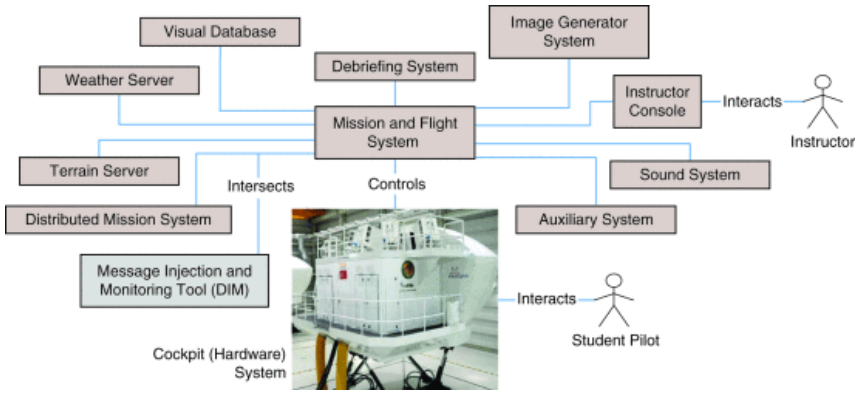

Figure 1: General Simulation Architecture

The three journals were read through and important information such as the module responsibilities and the connection was gathered [6] [7] [8].

The final two resources were the FlightGear wiki and GitHub source code which both gave specific information about the FlightGear software itself and allowed the group to dive into the concrete architecture. With this code base combined with the information learned from the other resources the group was able to predict and create a conceptual architecture for the simulation software.

4. Architecture Overview

FlightGear employs a modular architecture style that functions by routing data through various specialized submodules. This architecture is characterized by its interactive loops whereby each module processes its own data and interacts with other modules to achieve stable flight simulation. The overall software system is believed to utilize a combination of various architectural styles including the object-oriented approach for defining various aspects of a simulation session. Through this architecture, the system can represent various entities within the simulator such as airplanes and airports.

The team has identified 4 submodules that work together to provide flight simulation. These submodules are responsible for accepting user inputs, environment conditions,

terrain generation and weather condition. The data is processed and sent to the simulation engine in order to provide a realistic flight simulation.

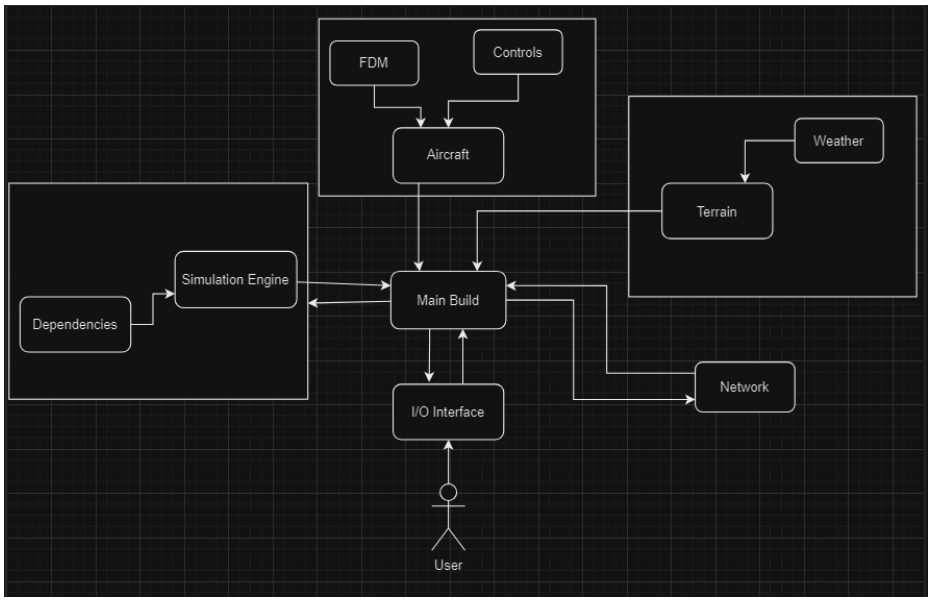

Figure 2: FlightGear Conceptual Architecture

The diagram offers a visual representation of the conceptual architecture for FlightGear. At the core of the system is the main build module which is the central entity coordinating flow of data between various submodules as well as input form the I/O interface. The main build is supported by the simulation engine which uses the data retrieved by the main build in order to render a realistic simulation. The submodule then sends the simulation back to the main build to display the generated simulation. The terrain submodule is used to dynamically obtain scenery data from an external server in order to procedurally generate the terrain including static objects and weather conditions [9]. The aircraft submodule is used to store data necessary to generate visuals for the aircraft as well as performance specifications for various airplanes, and helicopters. This submodule uses the Flight Dynamics Module (FDM) to calculate the aircraft's response to various physical conditions and translate user inputs into corresponding movements within the simulation [1].

5. Subcomponents

5.1. Main Build

Overview: This subsystem is a key component of FlightGear architecture. The main functionality of this system is to act as a simulation host and accept user inputs through the I/O interface, as well as run services to interact with other subsystems, namely the terrain and simulation engine

subsystems. The main build interacts with the simulation engine in order to generate realistic flight simulations [3].

Inputs: The main build takes multiple data inputs in order to create a realistic flight simulation. The inputs are from the user's preferred input method, scenery and weather data from the terrain subsystem and aircraft data. This subsystem utilizes "Terrasync", a service that runs in the background to dynamically generate terrains through the terrain module by utilizing user's current position within the simulation [2].

Outputs: The main build is responsible for processing data obtained from other subsystems and converting it into usable parameters for the simulation engine. The subsystem is also responsible for ensuring coordinating and outputting the results of the simulation to the user such as system conditions, failure scenarios, or external conditions within the simulation.

5.2. Simulation Engine

Overview: The simulation engine subsystem is the primary module responsible for generating visuals for the simulation. It achieves this through converting data obtained from other subsystems including data from the aircraft module and terrain data from the terrain subsystem to generate a cohesive flight simulation. This subsystem converts user inputs into relevant data and monitors its effects on the aircraft within the simulation in order to mirror the complexities of real-word aviation [3].

Inputs: The simulation engine takes in data from the aircraft subsystem such as information from the Flight Dynamics Model detailing the forces acting on the aircraft, such as lift, drag, thrust, and weight. Additionally, the engine interacts with the terrain subsystem in order to receive data regarding weather, wind speed, topographical data and static object information to render the environment. The module also receives and processes user inputs.

Outputs: The simulation engine provides visual rendering of the flight environment, including landscapes, weather effects, aircraft models, and airport infrastructure. Additionally, the engine utilizes data to generate dynamic visual renderings of physical conditions such as aerodynamic forces, aircraft performance, and environmental effects on the aircraft. The simulation engine provides real-time feedback on the status of the

simulation, including performance metrics, error reports, and results of the simulation.

5.3. Terrain

Overview: This subsystem is responsible for generating a detailed representation of terrain features. It is responsible for interacting with the main build in order to dynamically fetch terrain information from the terrain server through “Terrasync”. The terrain server is a dedicated HTTP server which houses a collection of scenery files for “just-in-time” downloads [2]. This requires a steady internet connection in order to ensure seamless downloads . Alternatively, users without a stable internet connection can manually download scenery files from the server.

Inputs: The subsystem requires the user’s location within the simulation along with simulation time. The module uses this data to dynamically generate terrain and weather conditions in order to determine the placement and characteristics of static objects.

Outputs: The module fetches files containing data required by the simulation engine to generate terrain simulation. The files contain data regarding weather, airport and navigation as well as static objects within the scenery.

5.4. Aircraft

The Aircraft subsystem defines how the FlightGear simulation controls, handling the specifications of the virtual “hardware” the user chooses to pilot. Both the aircraft appearance (external model, internal cockpit interfaces) and the Flight Dynamics Model (FDM) are within the domain of this subsystem.

The “appearance” of the Aircraft includes the models, textures, lighting and animations used in visualizing the aircraft in the simulation. It also includes what and how information is communicated to the user from within the cockpit, such as the aircraft’s status, fuel, altitude, et cetera.

The Flight Dynamics Model defines how the Aircraft interacts with the environment via the physics simulation, and how the Aircraft is controlled by the user. This includes any vehicle subsystems, collision, aerodynamics, weight balance, center of mass, trajectory, forces such as

thrust lift and drag, and the mapping of user controls to functions such as engines, wing flaps and landing gear, as well as how those functions operate.

5.5. Network

The Network subsystem is responsible for communication between multiple instances of Flightgear. It allows for LAN/server-based multiplayer (allowing for users to simulate air traffic control, flying in a formation and other processes that cannot be done in singleplayer), Google Maps integration to track where other users are currently flying, as well as handling of multi-monitor environments (where multiple FlightGear instances are synchronised and run together).

6. Evolution

FlightGear took its first steps in 1996 as an open-source project initiated by David Murr. In its infancy, the simulator was rudimentary, offering basic graphics and flight dynamics. Progress was slow during this period due to limited resources and a small development team. However, despite these challenges, FlightGear laid the foundation for what would become a thriving open-source flight simulation community. Early contributors focused on laying the groundwork for future development, establishing the core architecture of the simulator and ensuring compatibility across multiple operating systems [9].

FlightGear experienced significant growth and expansion during the period spanning from 2001 to 2015. Milestones such as the release of version 0.9.10 marked important stages in its development, introducing improved graphics, enhanced flight dynamics, and broader platform support. As the community grew, so did the scope of the project. New features were added, including support for multiplayer mode, allowing users to fly together in shared virtual environments [9]. FlightGear became increasingly popular among aviation enthusiasts, researchers, and educators, with its realism and open-source nature making it a valuable tool for flight training, aerospace research, and educational purposes.

In more recent years, FlightGear has continued to evolve and adapt to changing technologies and user expectations. Releases such as version 2020.1.1 showcased significant updates to the simulation engine, improved graphics, and enhanced flight dynamics. The simulator gained traction in educational institutions and flight training programs, further solidifying its position as a leading open-source flight simulation platform. Ongoing development efforts focus on refining realism, expanding features, and exploring new technologies such as virtual reality (VR) support and advanced weather simulation [9]. FlightGear remains a vibrant community-driven project, with a dedicated

team of developers and contributors committed to pushing the boundaries of open-source flight simulation.

7. Data Flow

In FlightGear, the program initializes all the data necessary to build the flight simulation. This will include which aircraft the users select, and the aircraft specification or specification generated from making a custom aircraft, the environment that the simulation is rendering in the form of terrain and weather. This will be enough to render the world that the simulation is taking place.

During the simulation, the system's role is to help accommodate with the user's goal and experience. The program receives real-time input from the users via the I/O and if there are multiple users, the network. This will include inputs to the aircraft's controls and if necessary, any changes to the user interface system. It then goes to the main build and gathers all the other data in the system. Other data includes all the aircraft specifications, the terrain system and its modifiers like wind speed, altitude, position of the aircraft, the overall terrain etc.

With all the data, the main build then processes the data and updates the positions of the aircraft. For the rest of the updates such as graphics, terrain change and other data. The system sends the data to the simulation engine to do the rest of the modifications necessary. The simulation engine uses third-party dependencies and libraries to run and modify the data and do the rest of the updates. The data then sent back to the system and to the users so they can see the changes made in the simulation. This process is then repeated in real-time for continuity. To ensure that the simulation runs smoothly, the number of times that the process repeats itself must be at least 30 times to make the system function like it is in real-time.

8. Concurrency

FlightGear was initiated in 1996 by David Murr in the belief that the performance of single-core processors would continue to improve, so for many years the software did not take concurrency into account. The FlightGear software runs with a repeating main loop. The main loop in FlightGear is different from the usual game loops that detect user input and update data. Instead, in FlightGear, many components are constantly working on the backend, and the main loop in FlightGear periodically passes control to the components and allows them to update themselves. Since control is handed off to different components, the performance of a single component has a significant impact on the

performance of other components. At the same time, since the renderer is in the main loop, the frequency the main loop runs is related to the frame rate. Therefore, FlightGear can be described as a single-threaded application.

Since 2005, there have been discussions in the FlightGear community about changing the architecture to support multithreading to improve performance. High Level Architecture (HLA) has been identified by the community as the new architecture that best meets FlightGear's requirements, as shown in the following figure. Each federate is connected to an RTI, which provides information, synchronization, and coordination services. This Topology allows each federate to ignore the status of other federates. Each independent federate can simulate aircraft, provide AI, record data for playback, etc. It can improve the overall performance of the software. But Curt Olson, as one of FlightGear's development leader, has been arguing against the use of multithreading because FlightGear is implemented in C and C++, both of which have poor support for concurrency. This proposal has made no progress since it was introduced in 2009 [6].

FlightGear's current concurrent code exists mainly in the OpenSceneGraph which is a third-party graphic toolkit implemented in C++ and OpenGL, multiplayer subcomponent and a subcomponent called Terrasync. TerraSync allows users to download upcoming terrain in the backend when gaming. It is initially set up to be in the same process as the game but can be set up to run on a different process. The multiplayer component requires a shared environment, communication and real-time interaction with other players to update game data. However, the concurrency of all parts is not related to the conceptual architecture of the simulator itself, so our team will not discuss further.

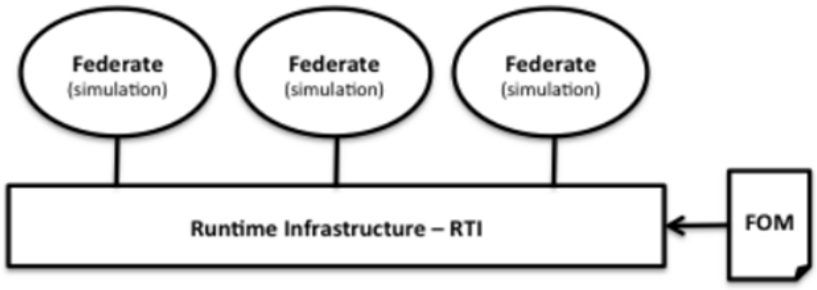

Figure 2-2: The topology of HLA

9. Use Cases/Sequence Diagrams

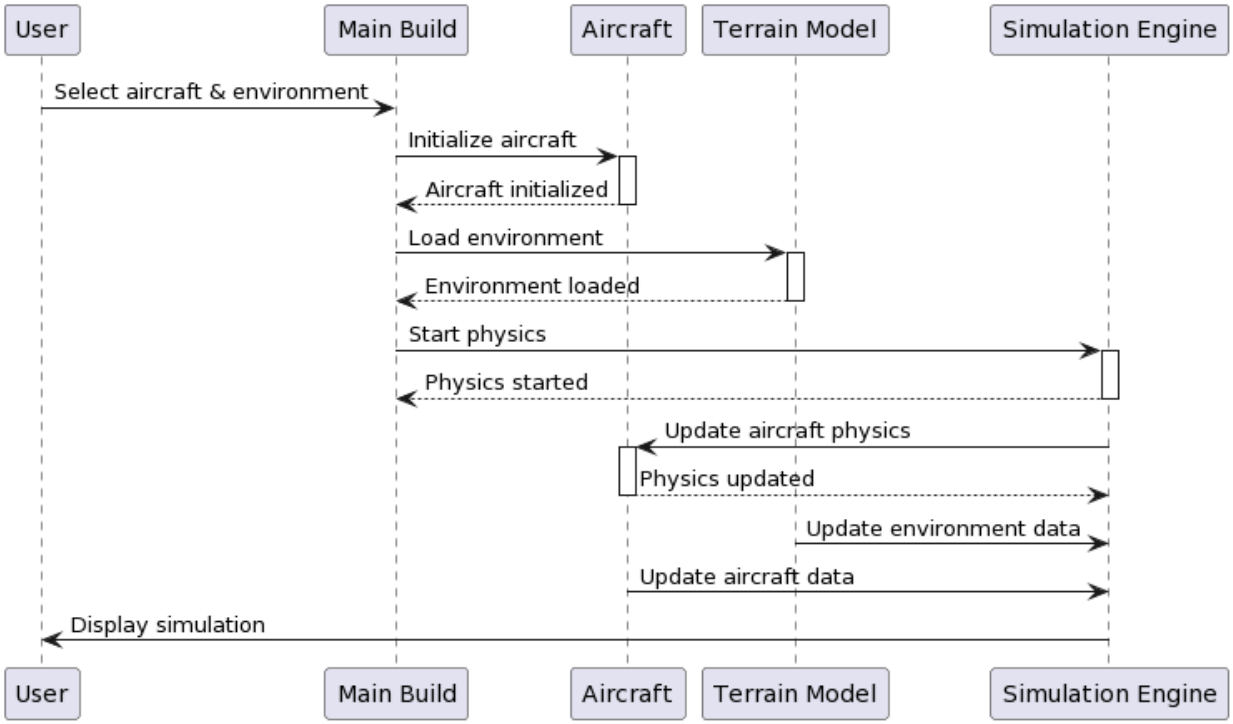

Figure 3) 9.1 Use Case #1: Launching a Simulation

This sequence diagram describes the process of a user launching a new simulation under the proposed architecture. A user will first select the aircraft model and the environment they will be simulating. The main build module sets off a series of requests to the other four modules, initializing the environment for simulation. The simulation engine will update the aircraft physics based on the initial conditions followed by the environment data and the aircraft data updating for simulation, before being displayed to the user.

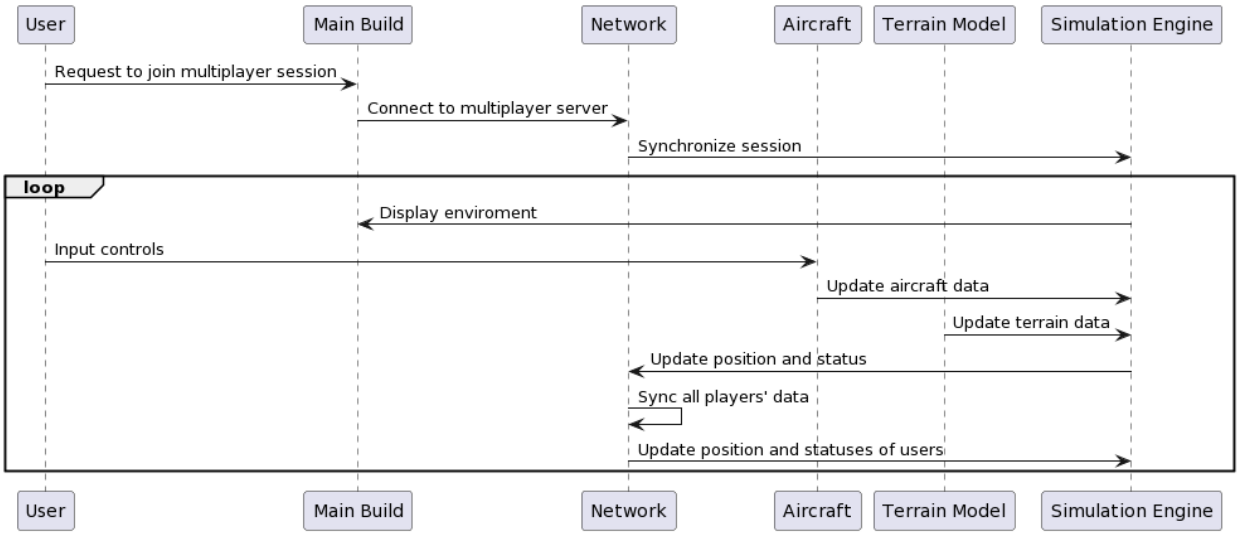

Figure 4) 9.2 Use Case #2: Multiplayer Session

FlightGear supports multiplayer sessions for flight simulations. This sequence diagram describes a multiplayer session under the proposed architecture. The user sends a request to join the multiplayer session and is then connected to the network. The network synchronizes all the initial data from each user and sends the data to the simulation engine. As the user provides inputs to control the aircraft, the updated aircraft data and terrain data is sent to the simulation engine, which updates the network to sync all the user's data. Finally, the data of the other users is provided to the simulation engine and the environment is updated accordingly. This occurs in a loop until the user exits the session.

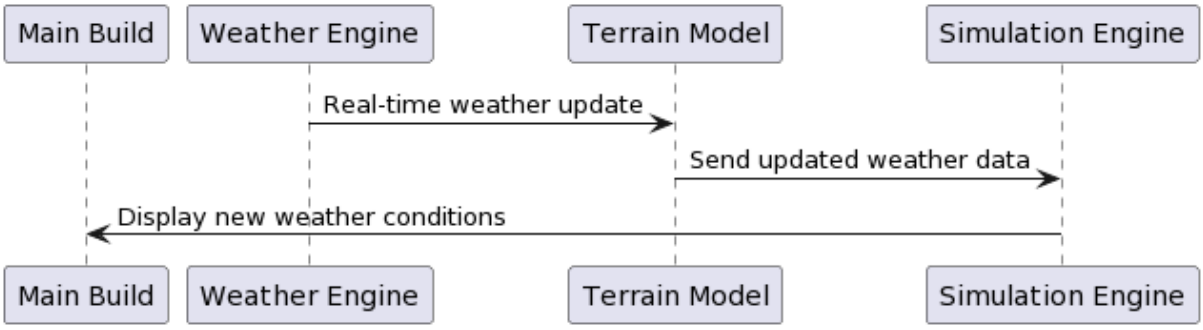

Figure 5) 9.3 Use Case #3: Dynamic Weather Conditions

This sequence diagram describes the process of the dynamic weather conditions being updated within the proposed architecture. FlightGear simulates weather conditions through weather engines that support real time weather fetch. The weather engine will provide real time updates for the terrain model, which will send the updated data to the

simulation engine. The simulation engine uses the provided data to display the weather conditions for the main build.

10. Implications For Divisions of Responsibility

FlightGear is open-source software by design, and the development architecture is structured to support individual contribution where appropriate. The source code is publicly available via the project GitHub repository for ease of access and takes a modular approach to aircraft design structure, affording as much diversity in aircraft design as possible. Contributors can work independently or collaboratively to create their own aircraft designs using the aircraft super-structure, with the freedom to add new processes to best accommodate the designer's vision. Scenery contributions follow a similar formula, with the public TerraGear toolbox available to produce custom scenery for use in simulations.

The project supports official databases for custom aircraft and custom scenery, which store the assets and data for user-created aircraft and terrain modules for users to download and use in their FlightGear simulations. However, third-party aircraft and terrain databases are also supported by the FlightGear engine.

While primarily designed for developer freedom and accessibility, the foundational FlightGear simulation code (SimGear) is more regulated to maintain project integrity. Only a few core community members are allowed write access to this wing of the project, as veteran developers volunteer to manage the development of different components. Any developer can propose changes to such components, but their changes must be validated by this moderation team before they can be added to the official project code.

11. Lessons Learned

FlightGear is a project that showed an enormously slow amount of progress when the team did not have a community. When it gathered a decent community size, multiple community-based suggestions as well as occasional work voluntarily done by interested developers accelerated the development of the project into what it is. The shift from single CPU to multi-threading showed how the project can increase its throughput tremendously. However, to utilize the resources available, a coherent architecture is necessary. The repository style gives the architecture readability to allow the community change.

12. Conclusion

FlightGear has been in development for almost 30 years since the project was released in 1996. FlightGear employs a modular architectural style with a variety of specialized submodules that route data to achieve functionality, including four modules. These submodules are responsible for accepting user inputs, environment conditions, terrain generation and weather conditions. By analyzing the software source code and use cases, we gained an in-depth understanding of the software's architecture and how a flight simulator works. This led to the creation of the Sequence Diagram and Box Diagram above. By reading the wiki and community forums, we learn about the history of the simulator, discuss the concurrency of the simulator, and the division of responsibilities between the various developers. We hope to discuss the functionality of each subcomponent in depth in future reports and look forward to the future development of this simulator.

References

[1] "Flight Dynamics Model," 16 Feb 2024. [Online]. Available: https://wiki.flightgear.org/Flight_Dynamics_Model.

[2] "TerraSync," 16 Feb. 2024. [Online]. Available: https://wiki.flightgear.org/TerraSync.

[3] "SimGear," 16 Feb. 2024. [Online]. Available: https://wiki.flightgear.org/SimGear.

[4] D. Allerton, Flight Simulation Software, New York: Wiley, 2022.

[5] D. Allerton, Principles of Flight Simulation, New York: Wiley, 2009.

[6] Q. H. J. H. Guangda Liu, "Architecture Development of Research Flight Simulator Based on COTS," *IEEE XPLORE,* pp. 1-4, 2009.

[7] S. T. O. S. M. T. K. H. H. F. E. A. O. B. Vahid Garousi, "Automated Testing of Simulation Software in the Aviation Industry: An Experience Report," *IEEE XPLORE,* pp. 63-75, 2018.

[8] H. Z. Zhao Guozhu, "Flight Simulator Architecture and Computer System Design and Research," *IEEE XPLORE,* pp. 35-39, 2020.

[9] "FlightGear Wiki," 19th October 2023. [Online]. Available: https://wiki.flightgear.org/Main_Page.