FlightGear's Conceptual Architecture

Group 4, Chicken AI

CISC 322/326

Instructor: Amir Ebrahimi

February 19, 2024

Abbey Cameron - 19aec6@queensu.ca
Akash Singh - 20as166@queensu.ca
Derek Youngman - 20day1@queensu.ca
Marion Anglin - 20mma2@queensu.ca
Shrinidhi Thatahngudi Sampath Krishnan - 21stsk@queensu.ca
Ximing Yu - 20xy@queensu.ca

Abstract

This paper analyzes the conceptual architecture of FlightGear, a flight simulator. FlightGear is open-source, which is distinct from other flight simulators and promotes the evolution and modifiability of and within FlightGear [1]. It is used for pilot training, research, and as a video game, available on a wide variety of operating systems [2].

FlightGear can be broken down into several subsystems. FlightGear is centered around its property tree, which acts as a message bus [3]. The other 9 principal components work together around the property tree to provide a realistic flying experience. Services provided by subsystems include realistic aircraft aerodynamics, realistic flight control, and advanced weather simulation. This paper describes how each subsystem works and how they interact at a high level. FlightGear's conceptual architecture primarily follows a publish-subscribe pattern. Two use cases are described to highlight this architectural style.

Our team derives FlightGear's conceptual architecture by first understanding the general requirements of a flight simulator, followed by relating those ideas to FlightGear-specific documentation. We also discuss the lessons we learned from this project.

Introduction and Overview

FlightGear is a realistic flight simulator primarily used for pilot training. It provides a practical flying experience through features such as realistic flight aerodynamics, authentic flight controls, and an accurate display of the world, sky, and weather conditions. FlightGear also supports multiplayer, making it appealing as a video game and differing from alternative flight simulators [2]. An additional unique feature of FlightGear is its open-source nature, encouraging contribution from anyone, promoting evolution and modifiability [1]. FlightGear's open-source nature also promotes research into flight simulator design. It's available on a wide variety of platforms [2] and has moderate hardware requirements [4], allowing anyone to access it.

In this paper, FlightGear's conceptual architecture is investigated, without exploring FlightGear's actual implementation. FlightGear is made up of ten principal subsystems: aircraft, flight dynamics, AI air traffic control, environment, rendering, sound and audio, user interface (UI), input/output (I/O), and multiplayer, which are all tied together primarily through the property tree subsystem. The property tree stores most of the system's information, allowing components to publish changes to the tree and subscribe to other changes [3]. Thus, a publish-subscribe style is the primary architectural style of the system. In addition, the multiplayer and flight dynamics components employ client-server and process control styles, respectively.

The architecture is described in this paper by outlining each of the subsystems individually, highlighting their purpose and interaction with other subsystems. The architectural style of the system is explained in depth. This paper will further explore the system's concurrency, data flow, and control flow to explain the system's interactions. Two use cases are described: a pilot taking off an aircraft and a weather simulation to highlight the system's architecture. Diagrams are provided for visual aid. Finally, this report will discuss how the selected architectural styles and subsystems were derived from the architecture, as well as lessons learned from the team during the fabrication of this report.

Derivation Process

To derive the conceptual architecture our team first consulted available books and research papers that outline the general requirements and design of a flight simulator. This seemed like a good place to start as opposed to the FlightGear Wiki as the books and papers provide a clearer overview of general flight simulator requirements. With a general knowledge of flight simulation, our team was then able to efficiently parse through FlightGear's large and somewhat disorganized Wiki to uncover its specific architecture. Our team checked initial understandings against the findings from the Wiki to validate our findings. For instance, each paper or book suggested and/or justified the use of certain architectural styles, such as publish-subscribe, client-server, or object-oriented. Then, to narrow down the set of possible architectural styles, the search within the FlightGear Wiki was focused on finding evidence supporting the utilization of these styles. This did allow for the possibility of confirmation bias, but all information was evaluated critically, acknowledging the possibility that FlightGear was designed differently than other flight simulators.

Through this process, our team eventually arrived at an understanding of FlightGear's subsystems, architecture style, and data flow. Individual findings were discussed together to form a box-and-arrows diagram summarizing the whole system. Two use cases that seemed to best highlight our conceptual architecture were chosen.

Architecture Components

The following subsystems can be found in the conceptual architecture of FlightGear.

Property Tree Subsystem

The property tree system controls almost all information held within the FlightGear system. This system acts as the central system of the simulation; representing the internal state of the system, such as the aerodynamics subsystem and the environment subsystem. This subsystem ties together all other subsystems in FlightGear; it allows the mechanics to share data. The property tree also functions as a key-value hash, allowing the program to look up any key (property name) and its associated value while being formatted in a tree-like hierarchy. This approach is highly flexible, which is beneficial when dealing with aircrafts with different types of control inputs and data inputs. An example of this use includes joystick input; once revealed to the property tree, it can be read by the aerodynamic/FDM subsystem. The property tree can also be extended to create a full API by subsystems such as the AI air traffic system and the multiplayer mode. [3]

Aircraft Subsystem

FlightGear is able to model many different kinds of aircrafts and can accurately showcase the controls of the aircraft [4]. The user can pick from a variety of aircrafts, and then the aircraft subsystem models this aircraft and sends that information to the rendering subsystem to visualize it with the UI subsystem [5]. This information includes fully interactive 3D cockpits that showcase that particular aircraft's controls accurately [4]. Additionally, Flightgear can smoothly and accurately model the aircraft's instruments, even modeling the instruments' inaccuracies, such as instrument lagging, as they would occur in the real world [4].

The aircraft subsystem is also involved with the multiplayer subsystem, in that it allows for updates of other aircrafts in a multi-user environment [4]. It models other aircrafts and their controls so that the user can view these from their own aircraft's cockpit.

Aerodynamic/FDM Subsystem

FlightGear features an aerodynamic subsystem. This subsystem is what controls how the user's aircraft flies in the game in relation to the air in the environment. FlightGear allows 3 options for the aerodynamic system: JSBSim, YASim, and UIUC [4]. All three of these models accomplish the same goal, but in different ways: allow the aircraft to fly. JSBSim is a generic model for simulating the motion of the aircraft. YASim is an integrated part of FlightGear and approaches the goal differently. YASim simulates the effect of the airflow on the aircraft. This approach allows the simulation to run based on geometry and the aircraft performance. Finally, UIUC, written by NASA, uses lookup tables to gather components holding the aerodynamic force and moment coefficients for aircrafts. From there, UIUC uses these values to calculate the forces and moments working on the aircraft at a particular moment. The aerodynamic system, regardless of what model is used, performs the flight dynamic calculations. Due to these calculations, the aerodynamic system is performance-critical [4].

AI Air Traffic Control Subsystem

This subsystem allows for interactions between the pilots and the program to create a more realistic experience. It does this in part through air traffic control, which separates aircrafts to prevent collisions and runs the flow of traffic [6]. This is especially beneficial for a multi-user environment, where multiple users and aircrafts need to interact in accurate and safe ways. This is achieved by using AI models that can regulate and direct traffic flow. Another feature of this subsystem is ground traffic simulation, which entails simulating things like car and train traffic which can be seen from the aircraft [6].

The AI air traffic control subsystem interacts with almost all of the other subsystems, such as the sound subsystem and the multiplayer component: it interacts with the sound subsystem by allowing the user to listen to the directives of the Air Traffic Controller; it interacts with the multiplayer component by determining and directing the flow of traffic with multi-user environments. The AI air traffic control subsystem implements a full API by extending the property tree subsystem, this allows AI traffic to be created and controlled [3].

Environment Subsystem

FlightGear and all generic flight simulators feature environment models. This subsystem controls all aspects of the environment, scenery and terrain development, sky, weather controls, and lighting [7]. The scenery is controlled by tiles, which are paged into the simulation via a separate thread in order to reduce the framerate reduction when loading new areas. In addition, this system features accurate time-of-day modeling, meaning that the system has accurate moon, sun, and planet placement for the user's time of day and location. The environment model also features night lighting and lighting in concentrated urban areas accurately resembling cities. This allows for realistic night flying, with the ability to spit cities and roads [4].

The world terrain is based on the most recently released SRTM (Shuttle Radar Topography Mission) terrain data, giving a realistic layout and feel to the environment of FlightGear. The SRTM data used features 3 arc second resolution and includes North America, South America, Europe, Asia, Africa, and Australia. In addition, across the world terrain, the subsystem includes over 20,000 airports as well as runway markings and placements [4].

Sound and Audio Subsystem

The sound subsystem consists of audio scheduling, as well as control of the input and output of sound. This subsystem runs by producing output audio streams based on commands and inputs supplied to the simulation host. For a typical flight simulator, sound components are typically run by exterior software, measuring outputs and audio mixing for the audio streams directed to the computer's onboard speakers. The sound system can handle three kinds of input streams: In-Cab Audio inputs, Ident Codes, and ATIS speech sounds. The latter is used for understanding spontaneous speech patterns [8]. The sound system takes things like microphone input and interprets it into the system when a user is participating in multiplayer, this is controlled through FlightGear's FGCom system [9]. An alternative method of implementing the sound subsystem is using sound cueing implemented by a sound card when using object-oriented architecture. Overall, the sound subsystem controls the output of all sound to the user's system in the simulation, as well as inputs like voice control for multiplayer options.

UI Subsystem

The UI subsystem refers to the part that the user sees. It is connected to the rendering subsystem as this subsystem determines the frame that the user sees at any given time. When the user interacts with their screen, the input is sent to the I/O subsystem, which then directs the input into the property tree subsystem. The property tree then sends the information to the correct subsystems to make the user input a reality in the Flightgear system. The UI subsystem can implement a full API by extending the property tree subsystem. This allows OpenGL textures to be created and modified, which is what is used to implement HUDs, GUIs, instruments, MFD avionics, and even liveries or scenery textures (VGDS) [3].

I/O Subsystem

The I/O subsystem manages data flow into and out of the property tree subsystem. In addition, it directs data to the appropriate subsystem from the property tree [3].

Rendering Subsystem

The rendering subsystem collects all the information from the other subsystems with the help of the property tree and transforms it into a visual frame that the user can see [5]. This entails updating the frame as the user, other aircrafts, and the scenery move. Additionally, it is able to implement a full API by extending the property tree. This allows for various 3D models, such as OpenGL textures to be created and modified, which is what is used to implement HUDs, GUIs, instruments, MFD avionics, and even liveries or scenery textures (VGDS) [3].

Multiplayer Component

A unique feature included in FlightGear compared to other flight simulators is the use of multiplayer functions. The architecture contains several networking options, which allow FlightGear to communicate with other instances of FlightGear, GPS receivers, and other software like the Atlas mapping utility through the multiplayer protocol [4, 10]. This subsystem connects with the sound subsystem as it supports voice communication with the help of FGCom. FGCom is a voice communication feature that allows users to communicate with other pilots and controllers during an in-game flight [9]. Additionally, the multiplayer system interacts with the AI air traffic control system; this feature is required to see other users' aircrafts in the game. FlightGear's network options allow the synchronization of multi-display instances. The network use of MPmap allows user aircraft locations to be updated in real-time to other players.

Non-Functional Requirements

The non-functional requirements featured in FlightGear's conceptual architecture are portability, modifiability, and most importantly, performance. [8] In addition, with the new inclusion of FlightGear's multi-user system, scalability, and capacity have been added as non-functional requirements to handle potential user influx.

Modifiability is important for FlightGear's system as it supports the addition of new components by users. [4] By requiring the need for modifiability, it allows the smooth integration of these components for users. In addition, FlightGear features various networking and system options, allowing it to run on various operating systems, requiring the need for portability. [4] The most important NFR as mentioned is performance. This is a requirement for various subsystems including I/O, rendering, and the aerodynamic/FDM subsystems. Fast response times are required in these subsystems to create a realistic flight experience; without this, the system would fail to execute as required and would perform slowly, or poorly.

Architectural Styles

FlightGear utilizes a few different architectural styles in its conceptual framework. It uses a publish-subscribe style to ensure real-time responsiveness, a client-server style for its multiplayer functionality, and a process control style to simulate aircraft control systems.

Publish-Subscribe Style

A flight simulator is a real-time system that must guarantee fast response times to provide a realistic flight simulation. For applications involving human input and displays, the latency should not exceed 60 ms, otherwise, discontinuities or lags in response are discernible by a human operator and will affect the response of the operator [11]. Therefore, data transactions are time-sensitive; data that must be transmitted through the system includes elevator and rudder position, aileron displacement, training instruction, and aircraft state [7]. Each producer does not need to know which consumers (possibly many) are interested in their production and vice-versa [7]. Hence, a publish-subscribe architecture is appropriate. A publish-subscribe style provides advantages in terms of performance compared to a client-server style (another reasonable choice for this system) [5], for example, potentially cutting latency in half compared to a client-server pattern [7]. The style also allows for parallelism, a critical component of flight gear, as synchronization is possible immediately upon receiving the latest data [7]. Given the strict response time requirements for the system, the system must be distributed to optimize processing speeds. A publish-subscribe pattern is standard for distributed simulation [12].

Different from other flight simulators, FlightGear is open source which promotes evolution of the system, taking input from anyone who would like to contribute [1]. A publish-subscribe style meets this requirement as it eases evolution since components can be modified without affecting other components in the system.

FlightGear's integral property tree facilitates a publish-subscribe pattern [3]. The property tree organizes all of the system's attributes hierarchically for access from FlightGear's subsystems [3]. Subsystems can publish changes to the property tree and can configure a listener that reports changes in the attributes they're interested in [3, 14]. That is, subsystems can publish and subscribe.

Client-Server Style

The multiplayer subsystem of FlightGear employs a client-server architectural style. To use multiplayer, users (clients) must configure and connect to a multiplayer server [14]. The multiplayer works with the rest of the subsystems of FlightGear to receive updates from players and propagate those updates to other players in their multiplayer environment [14]. The multiplayer servers also facilitate the ability for players to talk to each other via text and FGCom voice chat [15]. When a server receives any update from a client, the updates are sent to other clients who may or may not be connected to the same server [14].

Unfortunately, the use of the multiplayer component often hinders performance. Users often experience frame rate drops [15]. There is room for improvement in the multiplayer server protocol to address performance issues [16].

Process Control Style

The process control architecture style is used to mimic the control system of a real aircraft. In a real aircraft, there are sometimes automatic control systems that ensure the aircraft stays on its intended trajectory. For example, control systems of an aircraft must maintain airspeed, rate of descent, altitude, and heading [11]. A flight simulator aims to provide a realistic simulation of an aircraft, so these control systems should be mimicked by the flight simulator. This is best modeled by the process control architecture style. The control variables would be such as airspeed, rate of descent, altitude, and heading. A sensor would detect any relevant changes in the flight conditions which would trigger any necessary adjustments to the control variables. This style can be implemented within the overarching style to ensure the correct maintenance of these critical process variables.

The JSBSim system allows for the configuration of an automatic flight control system [17]. The user can configure reference values for a large number of control variables, such as heading. With the help of the user's configuration, FlightGear keeps track of your control variables and appropriately adjusts them to keep them sufficiently close to your reference values [18].

Box-and-Arrows Diagram for the Conceptual Architecture

Figure 1 presents the Box-and-Arrows diagram for the conceptual architecture.

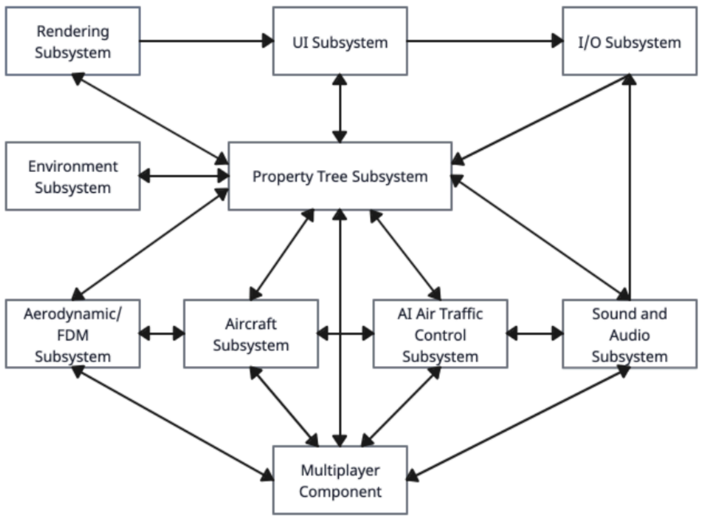

Figure 1: Box-and-arrows diagram for the conceptual architecture

Figure 2 presents a box-and-arrows diagram showing a possible high-level configuration of clients connected to multiplayer servers.

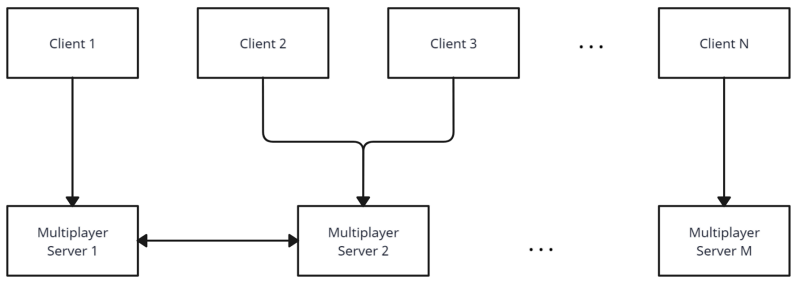

Figure 2: Box-and-arrows diagram for multiplayer client-server configuration

Concurrency

The real-time nature of the system requires many concurrent operations. The publish-subscribe architectural style can support concurrency well because the publishers are decoupled from the subscribers, allowing them to operate independently. Publishers can send messages without needing to know who the subscriber is, which enables asynchronous communication and parallel processing. Subscribers can receive and process messages concurrently as they are produced.

Concurrency plays an important role in the multiplayer feature, where multiple users connect to a shared simulation environment to fly together in real-time. Each connected pilot should operate in its own thread or process, allowing for parallel processing of incoming data and rendering updates. Concurrent processing is needed to manage user communication, synchronize aircraft positions, process various inputs, and update environment variables. Simultaneous processing can ensure a smooth and responsive multiplayer experience, as users interact with each other and share the same virtual environment.

Another instance where concurrency is essential is in decoupling the AI traffic system from other processes. By separating the AI traffic system into its own standalone component running in a separate thread or process, the system can better manage resources and improve runtime performance [19]. For example, in FlightGear's multiplayer mode, independent AI allows all players to see identical AI-controlled aircraft movements and behaviors simultaneously. Without decoupling the AI component, players may see visuals with delay or lag. Concurrency enhances the sense of immersion and realism of the simulation.

Furthermore, it is necessary that the rendering should be isolated into its own thread, running concurrently with other simulation tasks. The system needs to handle a large number of inputs affecting visuals, from aircraft's changing location to scenery updates, so the rendering should respond to these user inputs simultaneously and be able to deliver high frames. Additionally, while the simulation task and the processing of user inputs can operate at a consistent and elevated rate, the frame rate generated by the renderer can vary according to the scene's complexity. This flexibility allows FlightGear to adapt dynamically to changing environmental conditions and user interactions without sacrificing overall performance.

However, high levels of concurrency may pose a negative impact on testability. Multithreading and parallel processing introduce complexity to the system, especially for interactions between subsystems [20]. Any problems occurring in multithreaded code can be hard to track down and debug later on, affecting the overall system testability.

Global Control Flow and Data

Anyone who is developing a flight model for a simulator needs a dataset to develop a model for the simulator to operate on. Flight test data requires instrumentation of the aircraft to measure important factors such as acceleration, rates, velocities, altitude, etc. Since it is expensive to collect this model data, all the information should be covered and collected when flying the prototype aircraft. These datasets can only be processed by developers due to confidential information released in the public domain. If not available in the public domain, most of them are proprietary, since the manufacturers spend tens of millions of dollars [11].

If there is not enough funding to collect the model data, then alternatives are available to collect it, but it is not guaranteed that it will be as accurate or complete as the original manufacturer data. One possible way is to refer to technical papers or textbooks where the authors are given access to proprietary information. NASA archives are another place where aircraft data would be available, but there is a risk that it might be very old data [11].

A solution to address the above problems would be to get the data from papers that derive the data using the basic geometry of the aircraft, with equations, This is the basis of the USAF DATCOM program. Flightgear particularly uses data derived from DATCOM [11].

Data flow is primarily transferred through SATA (Serial Advanced Technology Attachment). They are hard drive interfaces that include registers to access the status of a device, and setting up transfers to and from a device, in Flight Gear’s case, the UI subsystem [11].

In FlightGear, the property tree is considered the central nervous system, where almost all inter-subsystem communication is done. Non-property data structures include flight plan structures such as GPS, Navaid/Airport/Runway structures which come from NavDataCache, ground-cache/scenery intersection code, and AI objects [13].

External Interfaces

FlightGear maintains several external interfaces that can be used to interact with or retrieve information from the system. FlightGear utilizes a combination of various open-source libraries, frameworks, and components to create a comprehensive flight simulation experience.

GUIs play a vital part in allowing users the ability to interact with their environment. For instance, "FlightGear Launch Control (FGRun)" [21] provides a graphical frontend, allowing users to make full use of the simulator. Additionally, the "Airport Diagram Generator" [22] is capable of generating airport diagrams using FlightGear data, and the GUI tool “TerraGear” allows developers to create custom scenery for their rendering projects.

Furthermore, various external interfaces transmit files from the user system in order to make the gameplay experience effective. FlightGear integrates FlightGear Communications [9] "FGCom" for voice chat during flights.

Installed as a file, FGCom facilitates communication among pilots in real-time. It leverages the user's input devices to transmit sound data. User information and sound data are externally transmitted to a web server and communicated live through their networks.

Additionally, FlightGear's "TerraGear" [24] is a collection of tools and rendering libraries that convert internal data files into 3D representations as real-time graphic renders. TerraGear imports various 3D datasets, such as coastlines and city outlines, and generates environments such as airports and runways.

Use Cases

Use Case #1: Weather Simulation

Perhaps one of the more critical use cases for FlightGear is the ability to simulate realistic and, at times, even live weather conditions for each flight. The following sequence diagram essentially outlines the component interactions from the time the user uses the "Setup Flight" option to see a list of available configurations and options until the simulation begins playing on the screen and the user starts to fly the plane.

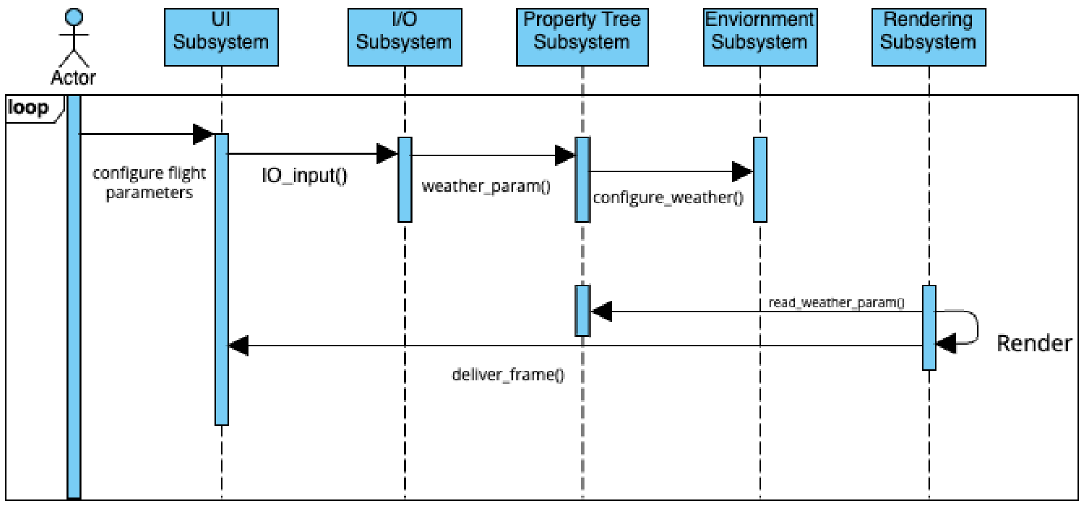

Figure 3: Sequence Diagram for Use Case #1

Weather simulation in FlightGear allows for the representation of atmospheric conditions, crucial for realistic flight experiences. Before takeoff, the pilot begins simulation setup, selects the desired plane, and determines their preferred weather conditions using the UI Subsystem. The UI Subsystem passes input to the I/O subsystem. The aircraft subsystem and the environment subsystem as subscribers read the aircraft and weather parameters from the property tree subsystem and the rendering subsystem also reads variables for rendering.

The visuals are rendered, the property tree subsystem reads the weather parameters, and the UI subsystem receives the rendered frames in its interface. The weather is continuously generated and updated until the user is satisfied and finishes the simulation using a loop.

Use Case #2: Pilot takes off the aircraft

One of the common scenarios in a flight simulator is the pilot taking the aircraft off the ground and into the air. The pilot enters the simulator, selects the desired aircraft model, chooses the departure airport, and sets up various flight parameters such as weather scenario and fuel load through the UI subsystem. All input variables are processed by the I/O subsystem and then published to the property tree subsystem for other subsystems to access. This initial configuration process continues until the user is satisfied with their choices.

After settling the flight configurations, the user presses the "Fly!" button to start the simulator. The aircraft subsystem and the environment subsystem as subscribers read the aircraft and weather parameters from the property tree subsystem and the rendering subsystem also reads variables for rendering.

The user can then start the engine and proceed with their takeoff operations. When the engine is started, the updated engine status is published to the property tree subsystem. The sound subsystem reads the updates from the property tree subsystem and makes the appropriate sound.

When the user is ready for takeoff, they can send operations through the instrument panel or joystick to initiate the takeoff. During this process, from navigating to the runway to accelerating for liftoff, the constantly changing aircraft location and applied force are updated in the property tree. The interested subsystems read the variable changes from the property tree subsystem and take corresponding actions. The environment subsystem listens to any scenery changes and updates the parameters. The Aerodynamic subsystem subscribes to force inputs, does calculations on them, and updates. The sound subsystem plays audio according to the engine status changes and airflow information. The rendering subsystem reads rendering-related parameters, produces the frame, and delivers it to the UI. The user continuously provides inputs via the UI subsystem, and other subscriber components keep responding to input changes. This process continues until the aircraft is taken to the sky and takeoff is successful. *Figure 4* presents the sequence diagram for the described use case.

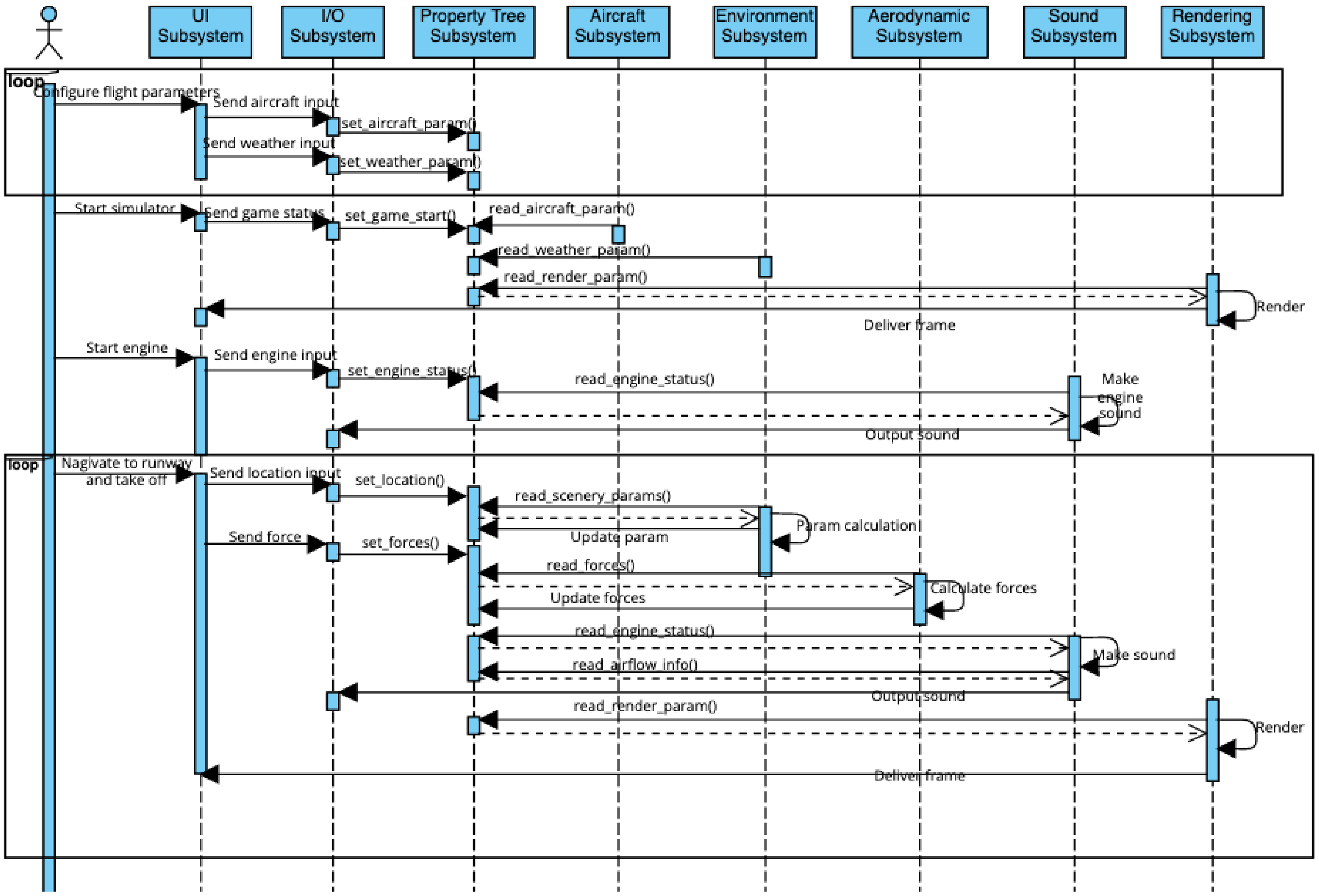

Figure 4: Sequence Diagram for Use Case #2

Conclusion

The conceptual architectures for FlightGear feature a publish-subscribe style, process-control style, as well as client-server for the multiplayer subsystem. The conceptual architecture features the property tree as the main component; controlling the inflow and outflow of input/output data, to and from respective subsystems. In addition, the conceptual architecture features ten subsystems, including the property tree, to run the entire FlightGear system. These subsystems range from performance-critical aspects like the property tree and the FDM system to smaller, more detailed subsystems like the environment subsystem that controls the terrain and sky the user sees in game.

Lessons Learned

The team found that there was a lot of information on FlightGear online and in the various resources provided. So much so that it was quite confusing and partly even overwhelming at times. Additionally, some of the topics we were instructed to discuss in our report had not been covered extensively in our course content, and as such approaching them was difficult because we needed to obtain lots of fundamental information before we were able to discuss the topics in the amount of detail required in the report. Moreover, FlightGear, as well as many other flight simulators, features complicated architecture and limited resources explaining the architecture. We had underestimated this complexity.

Moving forward (i.e. in uncovering the concrete architecture) we would like to get a bigger head start on uncovering information and organize it more carefully to avoid stressfully piecing everything together at the last minute.

Naming Conventions

- SRTM - Shuttle Radar Topography Mission
- UI - User Interface
- SATA - Serial Advanced Technology Attachment
- HUD - Head Up Display
- GUI - Graphical User Interface
- MFD - Multi-Function Device
- API - Application Programming Interface
- VGDS - Visual Docking Guidance System
- USAF DATACOM: United States Air Force Stability and Control Digital DATCOM (Data Compendium)

Data Dictionary

- FGCom: FlightGear voice communication feature
- MPMap: FlightGear utility that shows aircrafts flying in the FlightGear simulator. It also provides access to navigation data.
- OpenGL: A cross-language and cross-platform interface used for rendering 2D and 3D graphics.
- DATACOM: provide a systematic summary of methods for estimating basic stability and control derivatives.

References

[1] FlightGear - About: https://www.flightgear.org/about/
[2] FlightGear - Wikipedia: https://en.wikipedia.org/wiki/FlightGear
[3] FlightGear Wiki Property Tree: https://wiki.flightgear.org/Property_tree
[4] FlightGear Flight Simulator - Features: https://www.flightgear.org/about/features/
[5] A New Architecture for FlightGear Flight Simulator:
https://wiki.flightgear.org/w/images/1/1e/New_FG_architecture.pdf
[6] FlightGear Wiki Artificial Intelligence: https://wiki.flightgear.org/Artificial_intelligence
[7] Architecture Development of Research Flight Simulator Based on COTS:
https://ieeexplore.ieee.org/document/5364558
[8] Flight Simulator Architecture and Computer System Design and Research
[9] FlightGear Wiki FGCom 3.0: https://wiki.flightgear.org/FGCom_3.0
[10] FlightGear Wiki Multiplayer Protocol: https://wiki.flightgear.org/Multiplayer_protocol
[11] Flight Simulation Software, David Allteron:
https://learning.oreilly.com/library/view/flight-simulation-software/9781119737674/c01.xhtml
[12] Wikipedia - High Level Architecture:
https://en.wikipedia.org/wiki/High_Level_Architecture
[13] FlightGear Wiki - Property Tree Explained:
https://wiki.flightgear.org/Property_Tree/Explained
[14] FlightGear Wiki - How to set up multiplayer:
https://wiki.flightgear.org/Howto:Set_up_a_multiplayer_server
[15] FlightGear Wiki - How To: Multiplayer: https://wiki.flightgear.org/Howto:Multiplayer
[16] FlightGear Wiki - Multiplayer Server:
https://wiki.flightgear.org/FlightGear_Multiplayer_Server
[17] FlightGear Wiki - Introduction to Flight Control Systems:
https://wiki.flightgear.org/Introduction_to_flight_control_systems
[18] FlightGear Wiki - How To: Design and Autopilot:
https://wiki.flightgear.org/Howto:Design_an_autopilot
[19] FlightGear Wiki - Decoupling the AI Traffic System: FlightGear Wiki Decoupling the AI Traffic System
[20] FlightGear Wiki - Multi-Threading: FlightGear Wiki Multi-Threading in FlightGear
[21] FlightGear Wiki - Launch Control:https://wiki.flightgear.org/FlightGear_Launch_Control
[22] FlightGear Wiki - Airport Diagram Generator:
https://wiki.flightgear.org/Airport_Diagram_Generator
[23]FlightGear Wiki - Atlas: https://wiki.flightgear.org/Atlas
[24] FlightGear Wiki - TerraGearL: https://wiki.flightgear.org/TerraGear#GUI_Tool
[25] Automated Testing of Simulation Software in the Aviation Industry: An Experience Report:
https://ieeexplore.ieee.org/document/8356168
[26] Principles of Flight Simulation, David Allerton:
https://onlinelibrary.wiley.com/doi/book/10.1002/9780470685662