Conceptual Architecture of FlightGear

CISC 326/322 Assignment 1 Report

February 20th, 2024

Authors:
Max Lindsay - 19mjl5@queensu.ca
Vincent Proulx - 20vp24@queensu.ca
Mandi Wilby - 19mlw1@queensu.ca
Tyler Swan - 23lpg1@queensu.ca
Isaac Wood - 20ipw@queensu.ca
Vansh Panwar - 19vp8@queensu.ca

Table of Contents

Abstract

This report seeks to dissect the core functionality of the open source flying simulator FlightGear at a conceptual level. FlightGear provides users with the ability to simulate flying a plane step-by-step from pre-departure all the way to landing, with the ability to customize all aspects of a flight from environment settings to aerodynamic calculations.

Throughout the program's 28 years of evolution being supported by a community of volunteer developers striving to make the flying experience as realistic as possible, the codebase for FlightGear has grown to be quite extensive. By testing the software firsthand and studying the developer documentation available on the program's wiki pages, it becomes clear that FlightGear's architecture can be broken into eight key components; the flight dynamics model (FDM), environment, audio and visual, graphical user interface, artificial intelligence, networking, modding and add-ons, and database subsystems. These components have various dependencies on one another, and while the architectural style of the program is somewhat ambiguous due to its open source nature, certain components show distinct styles within their construction.

Introduction

The fact that FlightGear is open source is core to its identity, architecture, and shortcomings. The modularity of FlightGear is directly correlated to it being an open source passion project, since many people from around the world could make changes to its design at any time. The fact that it is a passion project also means that the developers could code what they wanted to code in as much detail and as in depth as their heart desires. In particular, the developers in charge of the FDM component of FlightGear have created a very advanced and complex physics engine which is still very modular, allowing for easy modification. That said, the parts of the project that are lacking are quite apparent. For example, the AI component of FlightGear plays scripted animations of traffic around airports to give the illusion of it being realistic and crowded. However, the planes in these animations are stiff and do not seem to obey the laws of physics the same way the users' planes do. FlightGear's age would also indicate that many of the developers who began the project have most likely moved on from the project after 28 years. The community is still present, only much less in the present day.

In the first section of the report, we will be discussing the High-Level Architecture of flightgear. It does not currently have a well defined overall architecture, However there are currently plans for it to be reconstructed with the 'High level Architecture' (HLA) as its architectural style for the future of FlightGear. In addition, certain components of Flightgear seem to have elements of their own architectural style that were then combined together into the project as a result of it being open source. We will also be discussing the other models which each subcomponent of FlightGear seems to employ. These include, but are not limited to, the Flight Dynamics Model (FDM), Networking, GUI, and Environment.

We will also analyse how FlightGear's current architecture affects the splitting of tasks amongst developers, data flow both internally and externally, concurrency, and how the system evolved to its current state. We will also provide some common use cases of FlightGear for the end user and how we came to these conclusions, along with some lessons learned along the way including limitations with our research in reversing this architecture using only external sources of information.

Architecture Overview

High-Level Architecture

There are multiple possible architectures from what we have found within the components, for instance networking uses a client server model for its multiplayer, and this can be extended to other module communications such as splitting off components like the FDM and parts of the GUI such as instrumentation to client machines and having the host be the main point of use for the user. However this is a rather niche use case and some other architectures would be more applicable to the standard single player use, such as pub-sub due to the global state given by the property tree. This also works with the modular style of components like the FDM, being able to have its relevant data modified externally by other components without needing to know what processing the FDM does. In its current state FG doesn't have a single consistent High Level Architecture though, as the community admits it needs to change to one as seen in [15]. The current plan in FlightGear development is to redesign it with reference to HLA, a reference architecture specifically for distributed simulators. It uses a message bus that handles serialization of messages, events and objects called the RTI, this then handles communication between different components called 'federates' which we expect share similar responsibilities as the components we describe in a later section, these federates can be run in completely different processes, even on different systems, making it fully modular and much more concurrent than the current system.

Components

FDM

The Flight Dynamics Model (FDM) is a core component of FlightGear; it is designed to simulate the physical and aerodynamic properties of the aircraft used [12]. This subsystem is crucial for rendering a realistic flying experience, replicating how different forces and flight conditions affect an aircraft's behavior. The FDM is very modular – allowing users to choose among several models or even integrate an external FDM for purposes like enhanced realism or enjoyment-related purposes, like a SpaceFDM [12]. The FDM interacts with various components, mostly with the environment module to help process the effects of external forces on the airplane.

Central to FlightGear's approach to flight dynamics simulation is its support for multiple FDM engines. This flexibility enables users to select the most suitable model for their needs. The FDMs that are natively supported are JSBSIM and YASIM, which each offer different capabilities and uses[12]. Performance of the FDM is crucial to how plane flight performs within FG.

YASim and JSBSim differ significantly in their approach to simulating flight dynamics. YASim is designed for scenarios where detailed aerodynamic data is unavailable, using aircraft geometry (shape of the aircraft) to approximate flight characteristics [13]. It's user-friendly for those with basic knowledge of aircraft design and flight performance, but achieving realistic behavior requires significant tweaking. On the other hand, JSBSim is a highly detailed, physics-based model that requires comprehensive aerodynamic data (how air flows around the plane). It's suitable for advanced simulations, offering precise control over aircraft behavior through extensive configuration options [14].

Both FDMs not only simulate external forces such as lift, drag, and gravity but they also consider the aircraft's response to control inputs and environmental conditions [12]. This includes the effects of wind, turbulence, and thermals, as well as the interaction with the runway surface during takeoff and landing phases.

Moreover, FlightGear's open architecture and the active development community encourage continuous improvement and the addition of new features to the FDM. Developers and enthusiasts can contribute models, improve existing ones, or develop entirely new FDM engines tailored to specific requirements, as per it being open source.

Environment

The environment component is responsible for creating a virtual world that reflects the real-world for flight simulation. It aims to immerse users in different conditions through the use of various environmental aspects that change and respond to internal and external influences. Those aspects include terrain, weather, climate, atmosphere and day/night lighting.

Terrain renders the physical features of the virtual world such as cities and mountains using real-world data. Weather is split into two systems, basic and advanced [43]. The basic weather system is the default everywhere while advanced is a better match to the real world. Climate updates the weather and seasons via the Köppen-Geiger climate map for different regions of the world. Atmosphere models the air at different heights. The day and night lighting in FlightGear is simulated with the aspect of ephemeris which renders the stars and moon in the sky in their proper location, resulting in lighting that reflects the time of day.

This component can be tested by comparing it with known sources of environmental data through changing the component conditions and observing environmental performance. As the environment also has to interact with the aircraft, testing the in-game interactions between them should be done to check if it produces proper results.

Some performance critical areas include the terrain rendering, aircraft and environment interactions, weather simulations and that user interface inputs result in the correct changes to the environment. As the point of FlightGear is to simulate the experience of aircraft flight, the game cannot work if these areas fail. The aircraft cannot take-off if there is no terrain to start on or other landmarks for orientation. The aircraft cannot fly correctly if it cannot interact with its surroundings. The weather significantly affects the world and the movement within it and its lack would result in a simulation without the realism of the real-world. Also, a user should be able to customize their environment to their tastes, otherwise much of the draw of the game would be removed without the ability to be creative nor the ability to tailor simulation to aid in flight training for a range of environments.

As seen in the previous section, this component mainly interacts with FDM as it needs information from the environment to figure out how the aircraft reacts to it.

A/V System

The rendering subsystem in FlightGear is the component responsible for all the visual components in the game with the help of multiple subcomponents. These subcomponents include Effects, Atmospheric light scattering (ALS), Canvas, and Shaders.

The Compositor encapsulates the rendering pipeline that composites the various rendering frameworks to create FlightGear’s visuals with its parameters exposed to the Property Tree.[29] The compositor became the default rendering framework in 2020.3. Each of the following rendering frameworks are handled through The Compositor’s pipeline.

The Effects subcomponent is a rendering framework that supports and encompasses and is responsible for the other rendering frameworks. Effects are containers for techniques which create the textures, lighting, and all other visual effects for the terrain and models. For each material type in materials.xml, an effect .eff file is created to parametrize each individual effect. Parameters for terrain effects are indicated through the ambient, diffuse, specular, emissive, shininess, and transparent fields of the material.xml file for which it is used. Model effects have a few more parameters than terrain effects since there is more variation possible from the model loaders than from the terrain system, but the model parameters include all of the terrain parameters save for the transparent parameter.[30] The Effects framework also supports custom effects which are used to change any visual attribute of a material to the liking of the user. A core motivation for effects is to support OpenGL shaders.

Shaders are a big part of the visual effects in FlightGear. They provide the means to calculate rendering effects on graphics hardware, creating the textures of the terrain users fly over (including everything from skylines and traffic to trees) and the skydome without stressing the CPU.[31][32] Shaders are implemented with GLSL (OpenGL Shading Language or "GLslang") which is the official OpenGL shading language to write shaders; based on the C programming language. Shaders can be programmed to replace parts of the fixed OpenGL function pipeline which indicates they are modular in nature, as is much of FlightGear.[33] The procedural generation of the terrain is also handled by Shaders as well as more complex noise functions.[34]

ALS is the rendering framework that supplies the shader effects for physically accurate lighting and fogging throughout the whole scene. These effects aim to replicate Rayleigh scattering and Mie scattering for light coming from the sun. It also positions the sun and moon to mirror their real world real time position based on the user’s clock and location to calculate the correct lighting as well as the correct phase of the moon. Haze, fog, clouds, horizon color, and color shifts in the terrain are also handled by ALS.[35]

The canvas subsystem, implemented entirely on the Property Tree in FlightGear, manages images that are rendered to a texture and updated dynamically at runtime by modifying a sub-tree in the property tree that represents the texture.[36] An individual canvas is used for each of the monitors and instruments seen in the HUD and GUI.

There is no dedicated audio subsystem in FlightGear. There are, however, features such as FGCom, sound effects, and a text-to-speech function for some radio dialog. FGCom 3.0 is the voice communication feature that allows communication between pilots and airspace controllers during a flight in multiplayer. It was designed with real radio communication in mind. It is available as feature integrated within FlightGear and as an external software for special cases such as using an older version of FlightGear or for more flexible yet complicated setup.[37]

Sound effects in FlightGear are composed of sound sample files along with an associated configuration PropertyList XML files. These configurations include orientation and position of the sound source, volume, pitch, and condition for which the sound effect is played.[38]

Text-to-speech in FlightGear is very rudimentary. It is used for advisory messages, limit warnings and failure notices while flying rather than “real” communications with mission control, but there has been a demand for more general callouts to be generated using text-to-speech in 2017.[39][40]

GUI

The GUI component is responsible for enhancing the experience of the user. It aims to present the user with an interface that is intuitive to use by being simple to navigate and understand. If the user cannot affect it, it is not part of the GUI. The subcomponents of the GUI include a menu bar, dropdown menus, dialogs and user actions [27], though a majority of its subcomponents are for its interactions with other components.

The menu bar displays various features including settings, aircraft, location, environment and add-ons while the dropdown menus specify the available options. The user may click on these to customize their simulation and application experience. There are many dialogs including ones for files, state controller and scenery which pop-up to present the relevant information when needed. The I/O takes the user's input devices and passes the information to various gameplay calculating subsystems in FlightGear and then outputs the impact the input had on the game based on returned information.

This component can be tested by checking if all GUI buttons work and that inputs are correctly validated. Users can navigate and attempt to find relevant information to check how intuitive and easy to use it is. The GUI should also be tested on a range of systems as all users should have similar experiences.

Some performance critical sections include the layout, graphical rendering and dynamic information updates. Layout should be clean and well put together, visuals should be clear and correct and the simulation should accurately reflect what a user chooses for customizations.

The GUI interacts with a multitude of components including add-ons, aircraft, airports, environment, cockpit, instrumentation, network etc. Essentially, anything that the user needs to be able to affect or observe likely interacts with the GUI. If the user can affect it, then that component acquires information from the GUI, otherwise the component likely is the one to send the GUI information to display.

AIModel

In the world of flight simulators, AI is a heavily utilized tool. It is used for a vast number of purposes such as enhancing realism, intelligent data analysis, adaptive learning and autonomous flight. However, in the case of FlightGear, the use of AI is severely limited and out-of-date. The AI systems within FlightGear are used for the purposes of air traffic control, traffic generation, ground traffic simulation and scenarios. These tasks are accomplished through a scripted approach rather than a neural network approach. This makes the AI within FlightGear semi-intelligent since it is non-predictive and more so responsive.[23][25]

The first use of the AI is AI traffic, or what it is commonly known as, interactive traffic. The purpose of this is to enhance realism by adding AI controlled traffic at airports. This system works off of four elements, AI aircraft models, AI traffic schedule which is stored as XML files, ground nets which help the plane go from gates to runways and vice versa, and finally the runway use configurations which determine what runways are to be used by AI traffic.[22] In terms of the ATC, AI implementation is very limited, to the extent of simulating basic tower communications, providing instructions and clearances to pilots based on their flight plans and the airport traffic. Ground traffic vehicles such as cars, trucks, trains, boats and ships are also controlled through AI. Lastly, there are scripted scenarios that use AI. For example, participating in aerial refueling with a tanker aircraft or dodging missiles in a combat scenario. These scenarios are pre-scripted AI that randomly occur or can be manually called for. There is major room for improvement in terms of AI for FlightGear, however, progress has come to a halt due to a developer's personal reasons.[21]

In terms of evolution, FlightGear does plan on switching to a higher level architecture for the software to improve modularity. One of the main goals of this development is improvement to the AI traffic system. Currently, the AI traffic subsystem works within the main loop for FG, the goal is to have an external component that does this instead. This would free up computing resources and allow for easier synchronization of AI traffic. Currently, Nasal scripting is used by many users to simulate their own AI and allow for flexible and customizable AI behavior via interactions with the simulated environment. The evolution and testing of the AI are very closely related, currently much testing is not needed since most AI is user generated via Nasal scripting, but in the future, if FG does implement a HLA, they would need to do unit testing and integration testing with several complex scenarios to make sure all the AI systems work together efficiently.[15]

The AI system also closely interacts with several other systems within the simulation. It interacts with the FDM to get current information for the AI traffic planes, using this information for controls. There is also interaction with the environment model for a similar reason, the AI traffic uses this info to influence its decisions and controls. With the overall use of AI, there are two major requirements at hand. One, it must be safe and reliable. The AI traffic must be reliably working to perfect realism and not ruin the simulation experience. Lastly, it must have good performance. The AI decision making and control execution should have no overhead delay to make sure the simulation runs as smoothly as possible. [21]

Networking

The networking subsystem in FlightGear is the component responsible for facilitating various functionalities through standard networking protocols. Its scope encompasses features such as multiplayer support, networked controllers, and integration with external modules for flight dynamics, GPS, mapping, and I/O operations.[1]

Multiplayer functionality enables multiple users to interact within the FlightGear environment simultaneously, acting as a pilot in a plane or on the ground as an air traffic controller. This feature is achieved through a custom-built protocol[2] based on UDP and XDR[3][4], ensuring efficient and reliable communication. In the main use case, players connect to a central server to interact with remote participants, while direct peer-to-peer connections can be established between local instances of FlightGear for local multiplayer sessions or synchronizing multiple instances of FlightGear to control one plane.[5] Additionally, within multiplayer the networking subsystem supports FGCom, an embedded voice chat service. FGCom utilizes a direct UDP connection[6], foregoing the use of XDR, to enable real-time voice communication between players.

Using the custom built Native protocol, you can also use external FDM's to replace the built in ones in FlightGear[7] making it possible to run the FDM completely separately from the rest of FlightGear, allowing the user to run the FDM component on a different computer.

The other components such as the remote controller[8] and external modules other than FDM rely only on UDP[9][10] and provide alternative I/O or extend the GUI submodules further via additional instrumentation.

Modding/Add-ons

This is the subsystem describing add-ons and scripting for the game, allowing users to be able to add additional content into FlightGear such as new airplanes, scenery, I/O functionality, additions to the HUD, and new game modes. FlightGear does scripting using Nasal and XML.[41]

Nasal is an interpreted object oriented scripting language developed specifically for FlightGear, made in C++ with syntax influenced by javascript, perl and python[42], and while used for some internal features such as wildfires and CDUs, It's primary purpose is to develop user generated add-ons.

Add-ons are downloaded and installed via placing them in a folder and pointing FlightGear to the folder in the settings, this then allows them to be enabled in the settings and loaded for use in Flight.

Database and Property Tree

There are several databases used throughout FlightGear. Several components interact with these databases for different reasons. The first example is the NavDB. NavDB, commonly known as Navdata Cache, stores information related to navigational aid for pilots. This database is accessed by the AI Traffic system for navigation and communication purposes.[17]

The next example is the Scenery Database which closely interacts with the environment system because it stores all the 3D world objects that populate the simulated environment. Things such as terrain elevation and features , buildings, airports, runways and textures and colors for realistic visuals are stored here.[16]

Lastly, there are the Aircraft and Livery databases that exist outside the software, however, they are often used for downloading and installing new aircrafts for the simulator. This database closely interacts with the FDM to share aircraft details.[18][19]

Most of these databases support scalability, however, the livery database has stopped growth and is limited to what is available. Performance with the databases is crucial because data flow must be quick for a proper flow of simulation.[19]

The property tree is known as the central nervous system. It acts as a hierarchical, key-value data structure that serves as an interface for accessing and manipulating internal variables within modules and subsystems. Subsystems access and modify variables in other subsystems through the property tree, facilitating data exchange and coordination.[17]

Non-functional Requirements

- Performance - The GUI requires responsive buttons that return inquiry results efficiently. The A/V system requires that shaders do not stress the CPU. FlightGear aims for 10 PPS and ping below 50 ms in multiplayer networking, but works around this by accounting for the desynchronisation of Clients clocks and updating the server selectively based on this[11]. the FDM, AI, A/V, & Environment systems have to work in real time, thus a new frame should be output every 16ms for 60FPS.
- Accuracy - Since FG is a flight simulator, all of its components must be as realistic as possible. This requires the components to function with extreme accuracy, for example. The A/V system must output the most accurate visuals and audio, whereas the FDM must accurately process the data it is being given for a smooth flight experience.
- Maintainability/Modularity - Since FG is an open source project with hundreds of contributors, it is key that the modules are made with maintainability and modularity in mind. This ensures future devs do not have a hard time making progress.

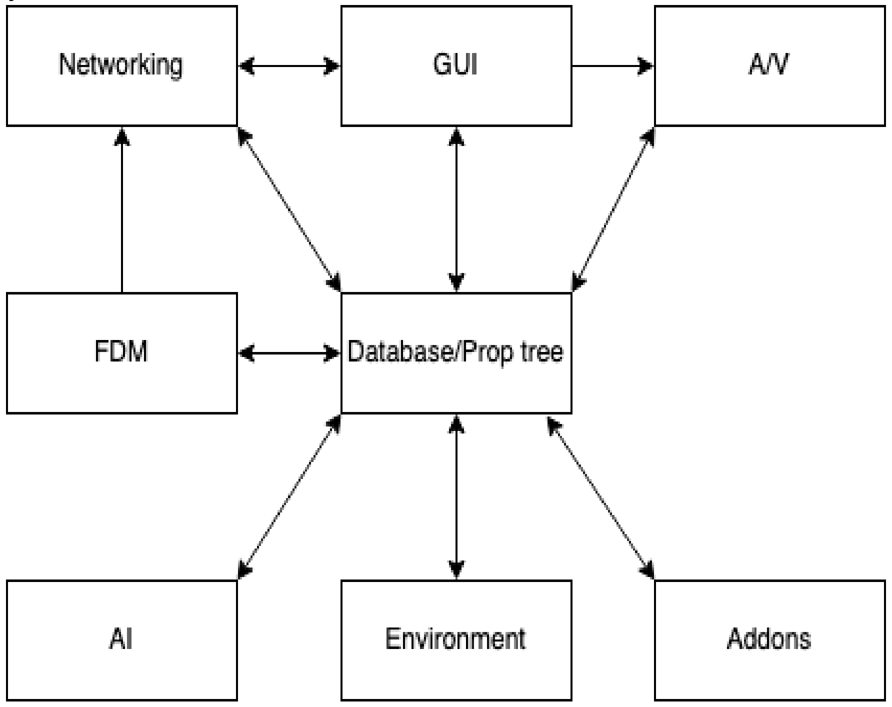
- Scalability - The FDM should be able to accommodate a wide range of aircrafts, from big to small, without a significant impact on computational performance or accuracy.

Concurrency

Some concurrency does exist within FlightGear such as code run via Nasal scripting, as that typically relies on worker threads to do the work while leaving the main FlightGear loop mostly untouched. However adding concurrency to other components runs into many issues in its current architecture, the existence of the property tree, and a single source of global state leads to issues in multiprocessing from race conditions. This means that if large parts of the environment and FDM etc were in separate processes, the simultaneous changes would lead to large issues in accuracy.[28]

System Diagram and Data Flow

The property tree as the representation of the global state has full read/write access in every other module. A/V takes canvas data from the property tree as well as self modifying images dynamically on the tree, The GUI requires data directly from the A/V system to output video and sound to the user, And it has to take measurements for instrumentation from the property tree as well as push user input into it. Networked I/O requires 2 way access to the GUI. It also reads and writes multiplayer user data to/from the property tree. The FDM uses Networking if using an external FDM, has to read/write physics data to/from the property tree. AI reads its tasks/path from the property tree and writes its location etc to it. The environment has to read current external changes from the property tree and write changes in calculations to it. Lastly add ons has to be able to write its canvas data among other things to the data tree and the addons manager has to be able to read the currently available addons from the property tree.



Evolution of System

FlightGear’s architecture is designed to be open, free and modular, allowing for continuous evolution and integration of new features and technologies. This structure allows for ongoing development and incorporation of innovations across various aspects of the simulator, including the FDM, aircraft models, and environmental simulation, like snow [15]. The project's commitment to open standards and community-driven development ensures that FlightGear can readily embrace new advancements in simulation technology, improve realism, and meet the evolving needs of its user base. This approach not only enhances the simulator's capabilities but also fosters a vibrant ecosystem for research, education, and hobbyist activities.

FlightGear’s evolution is helped by its adoption of HLA (High Level Architecture) [15]. This architecture enables the separation of the simulation into Federates that communicate with a Run-Time Infrastructure (RTI) [15].This approach also allows for the system to have better multithreading capabilities, and facilitates prioritizing different tasks. Most importantly, HLA allows for users to build their own components for FlightGear with a multitude of different programming languages [15].The shift towards HLA indicates FlightGear's commitment to scaling, and, accommodating evolving technologies and community contributions.

Division of Responsibilities Among Developers

FlightGear is an open source project originating in 1996. It has maintained its roots and continues to be a project worked on by many enthusiasts. While this democratized approach fuels fresh ideas and diverse perspectives, challenges arise. Managing communication, ensuring code cohesion, and navigating individual copyrights require constant attention. Sometimes the personal passion of a developer also leads to long rants and unnecessary complication of even the simplest tasks. Developers work in teams or independently working on modules or even things like manuals and the wiki. Each module comes with a copyright identifying the developers and their contact info. Developers choose to work on any part of the project they are passionate about, sometimes it is trivial as lighting in a building to work on improving AI traffic. The community has also developed a large list of add-ons/mods using Nasal that are a core component of the softwares current day popularity and usage [25].

External Interfaces

Networking has a lot of information transmitting both in and out of it, with multiplayer giving real time updates on other players locations via the custom-built multiplayer protocol, in addition to the networked I/O such as remote controls and voice chat.

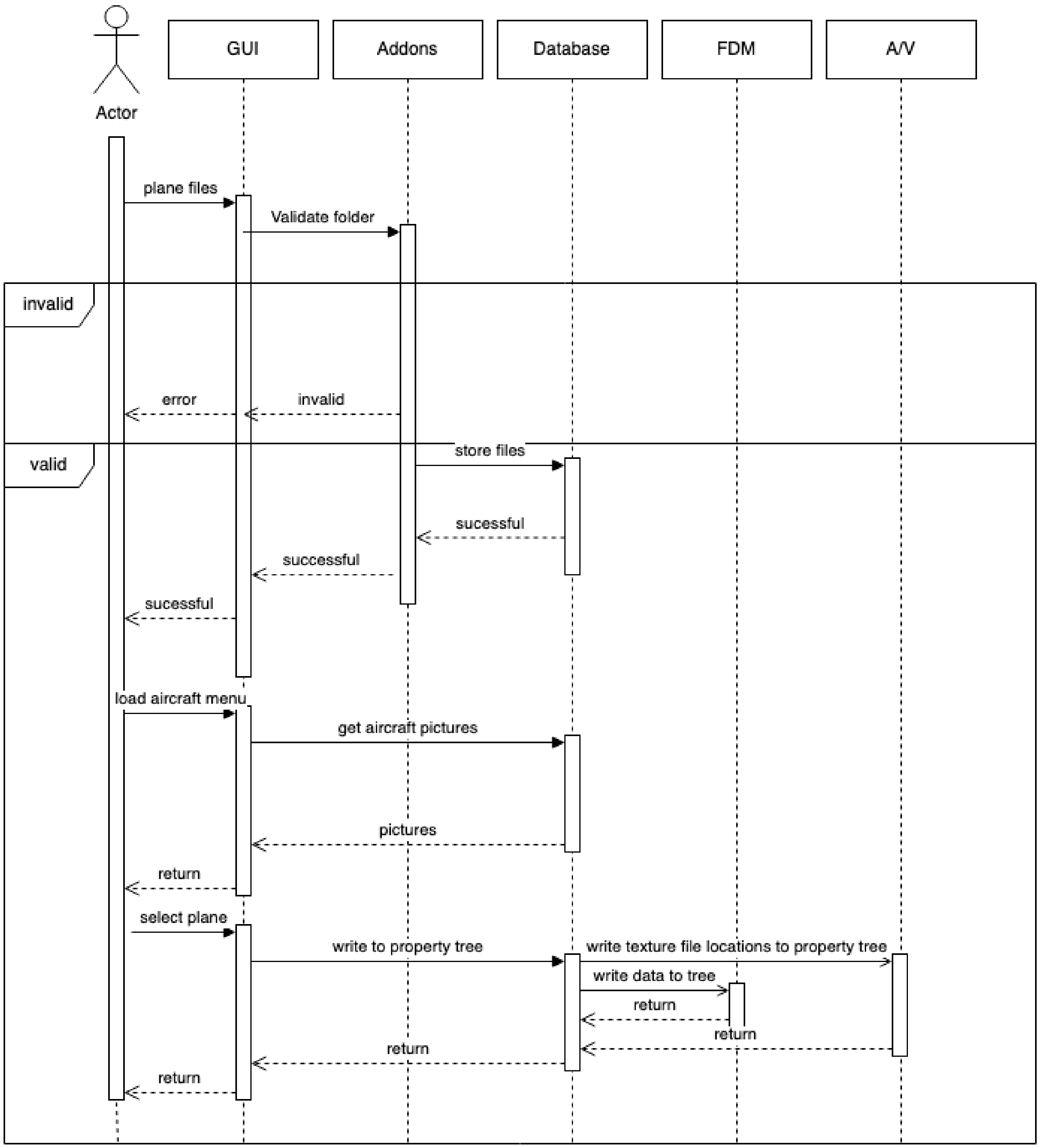
The addons module transmits information into the system via its Nasal and XML files, along with other possible files for textures in the case of plane models and custom scenery.

The A/V system transmits both video and sound from the system to the user via the GUI I/O device drivers, while the GUI system may transmit additional information for devices such as haptic feedback depending on the current state on the FDM. user input will also be transmitted into the system via the GUI I/O devices. FlightGear also supports the option to log aspects of flights, such as the rudders and elevators, and output a file containing details after the fact.

Use Cases

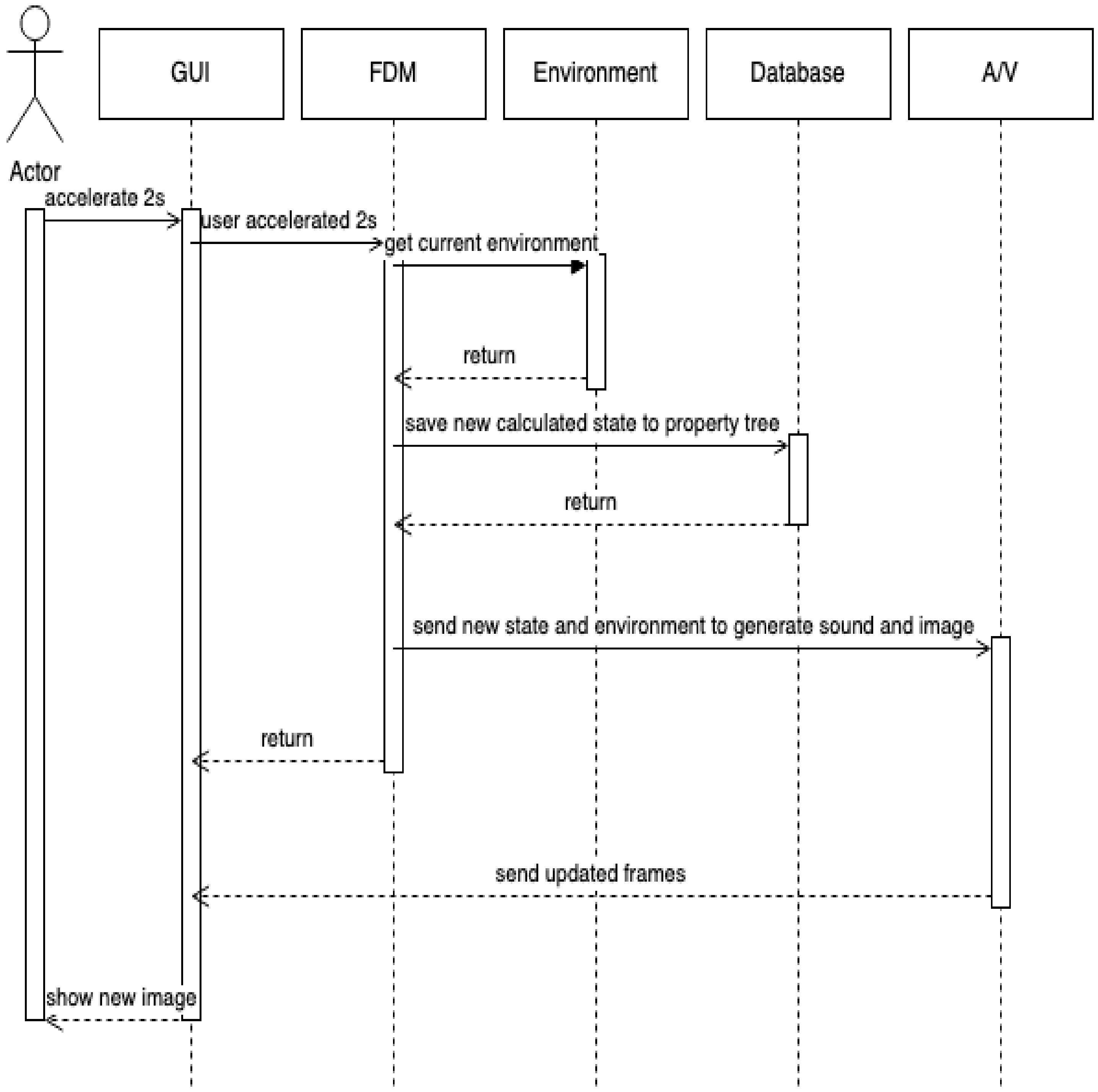
Installing a new plane and preparing for flight

Actor adds a new plane folder to GUI, which then notifies the addon module to manage and check the plane. If false, return error to the user, otherwise pass along to the database. User then goes to the plane menu via GUI. GUI pings database for plane files to show picture. User selects new plane, GUI sends signal to database to load plane data to the property tree for A/V and FDM to access



Flying a plane

The pilot uses the I/O devices to provide input to the FDM Via GUI, then the FDM uses the control commands and external forces to calculate the new state of the aircraft (position, velocity, attitude). In order to do this the FDM queries the Environment module for data like wind conditions or terrain elevation. After this, the Database updates with the new aircraft state. The FDM then sends the updated aircraft state and environmental information to the A/V System to generate realistic visuals. The A/V System finally sends data to GUI which returns the new frame to the user.



Data Dictionary and Naming Conventions

1. PPS - Packets per Second
2. XDR - External Data Representation
3. FDM - Flight Dynamics Model
4. HUD - Heads Up Display
5. HLA - High-Level Architecture
6. RTI - Run time infrastructure
7. FPS - Frames Per Second
8. A/V - Audio Visual
9. ALS - Atmospheric Light Scattering
10. UDP - User Datagram Protocol
11. FPS - Frames per second
12. CDU - Control Display Unit
13. Neural Network - It is a type of machine learning process that uses interconnected nodes or neurons in a layered structure that resembles the human brain
14. Ping - The time it takes for a small data set to be transmitted from your device to a server on the Internet and back to your device again

Conclusions

In conclusion, FlightGear is a complex, open-source, lightweight flight simulator made up of several components that interact to create a realistic simulation. Our findings show that each component employs its own architectural style, such as the client-server model used in networking. However, the main architecture of FlightGear was undergoing a transition away from inefficient data structures, which limited its ability to leverage multiprocessing capabilities. The introduction of High-Level Architecture (HLA) marked a positive step in this direction, as it promotes modularity and simplifies development. This evolution is crucial for FlightGear to utilize modern computing resources, including new languages and tools. Implementing HLA would further enhance modularity, making development easier and more organized. Currently, most development is driven by passionate individuals with a deep interest in specific aspects of FlightGear. This passion can make organization challenging. However, HLA would incentivize better ideas by simplifying their implementation.

Lessons Learned

Referencing provided resources was definitely something our group wishes we did more of to start the assignment off with. Without a basic understanding of a flight sim, it is really hard to look/fish for the information that is crucial for explaining the architecture.

One thing that we wish we knew ahead of time was the current state of FlightGear does not necessarily follow a HLA. It is trying to switch to one, however, currently it functions off of several modules sharing a single property tree.

One last thing we wished to do better was delegation of tasks, we started fairly late, about half-way into the life cycle of the assignment. The way we delegated tasks was also unpolished, causing people to be dependent on the work of others before they could make any progress. These issues caused several delays which could have been avoided and made the assignment much easier.

References

[1] “Features”. FlightGear.org. https://www.flightgear.org/about/features/ (accessed 13/2/2024).
[2] "multiplayer protocol". wiki.flightgear.org. https://wiki.flightgear.org/Multiplayer_protocol (accessed 10/2/2024).
[3] Srinivasan, R., "XDR: External Data Representation Standard", RFC 1832, DOI 10.17487/RFC1832, August 1995, https://www.rfc-editor.org/info/rfc1832 (accessed 13/2/2024).
[4] Eisler, M., Ed., "XDR: External Data Representation Standard", STD 67, RFC 4506, DOI 10.17487/RFC4506, May 2006, https://www.rfc-editor.org/info/rfc4506 (accessed 13/2/2024).
[5] “Howto:Multi-computing FlightGear”. wiki.flightgear.org. https://wiki.flightgear.org/Howto:Multi-computing_FlightGear (accessed 13/2/2024).
[6] ”FGCom 3.0”. wiki.flightgear.org. https://wiki.flightgear.org/FGCom_3.0 (accessed 13/2/2024).
[7] https://wiki.flightgear.org/Property_Tree/Sockets#Using_Proprietary/External_FDMs (accessed 13/2/2024).
[8] "FG777 controller". wiki.flightgear.org https://wiki.flightgear.org/FG777Controller (accessed 15/2/2024).
[9] “Atlas”. wiki.flightgear.org. https://wiki.flightgear.org/Atlas (accessed 13/2/2024).
[10] “FlightGear glass cockpit”. wiki.flightgear.org. https://wiki.flightgear.org/FlightGear_glass_cockpit (accessed 13/2/2024).
[11] “Real time manual” wiki.flightgear.org https://wiki.flightgear.org/Real_Time_manual
[12] “Flight Dynamics Model.” Flight Dynamics Model - FlightGear Wiki, https://wiki.flightgear.org/Flight_Dynamics_Model Accessed 18 Feb. 2024.
[13] “Yasim,” YASim - FlightGear wiki, https://wiki.flightgear.org/YASim (accessed Feb. 18, 2024).
[14] “JSBSim,” JSBSim - FlightGear wiki, https://wiki.flightgear.org/JSBSim (accessed Feb. 18, 2024).
[15] “High-level architecture,” High-Level Architecture - FlightGear wiki, https://wiki.flightgear.org/High-Level_Architecture (accessed Feb. 18, 2024).
[16] “FlightGear scenery database,” FlightGear Scenery Database - FlightGear wiki, https://wiki.flightgear.org/FlightGear_Scenery_Database (accessed Feb. 18, 2024).
[17] “Navdata Cache,” Navdata cache - FlightGear wiki, https://wiki.flightgear.org/Navdata_cache (accessed Feb. 18, 2024).
[18] “Aircraft,” Aircraft - FlightGear wiki, https://wiki.flightgear.org/Aircraft (accessed Feb. 18, 2024).
[19] “FlightGear Livery Database,” FlightGear livery database - FlightGear wiki, https://wiki.flightgear.org/FlightGear_livery_database (accessed Feb. 18, 2024).
[20] “Artificial Intelligence,” Artificial intelligence - FlightGear wiki, https://wiki.flightgear.org/Artificial_intelligence (accessed Feb. 18, 2024).
[21] “Ai Systems,” AI Systems - FlightGear wiki, https://wiki.flightgear.org/AI_Systems (accessed Feb. 18, 2024).
[22] “Ai traffic,” AI Traffic - FlightGear wiki, https://wiki.flightgear.org/AI_Traffic (accessed Feb. 18, 2024).
[23] “Status of AI in FlightGear,” Status of AI in FlightGear - FlightGear wiki, https://wiki.flightgear.org/Status_of_AI_in_FlightGear (accessed Feb. 18, 2024).
[24] “Portal:developer,” Portal:Developer - FlightGear wiki, https://wiki.flightgear.org/Portal:Developer (accessed Feb. 18, 2024).
[25] P. A. T. Center, “How artificial intelligence is revolutionizing pilot training?,” LinkedIn, https://www.linkedin.com/pulse/how-artificial-intelligence-revolutionizing-pilot-training/ (accessed Feb. 18, 2024).

[26] “FlightGear high-level architecture support,” FlightGear high-level architecture support - FlightGear wiki,
https://wiki.flightgear.org/FlightGear_high-level_architecture_support#AI_Traffic (accessed Feb. 18, 2024).
[27] “FlightGear UI overview,” FlightGear UI overview - FlightGear wiki,
https://wiki.flightgear.org/Flightgear_UI_overview (accessed Feb 18, 2024).
[28] Andrew J. Ross. Email. Feb 2002,
https://www.mail-archive.com/flightgear-devel@flightgear.org/msg03069.html
[29] “Compositor,” Compositor - FlightGear wiki, https://wiki.flightgear.org/Compositor (accessed Feb. 19, 2024).
[30] “Effect framework,” Effect framework - FlightGear wiki,
https://wiki.flightgear.org/Effect_framework (accessed Feb. 19, 2024).
[31] “Shader,” Shader - FlightGear wiki, https://wiki.flightgear.org/Shader (accessed Feb. 19, 2024).
[32] “Atmospheric light scattering,” Atmospheric light scattering - FlightGear wiki,
https://wiki.flightgear.org/Atmospheric_light_scattering (accessed Feb. 19, 2024).
[33] “HOWTO:Shader Programming in flightgear,” Howto:Shader programming in FlightGear - FlightGear wiki, https://wiki.flightgear.org/Howto:Shader_programming_in_FlightGear (accessed Feb. 19, 2024).
[34] “Procedural texturing,” Procedural texturing - FlightGear wiki,
https://wiki.flightgear.org/Procedural_texturing (accessed Feb. 19, 2024).
[35] “Atmospheric light scattering FAQ,” Atmospheric light scattering FAQ - FlightGear wiki,
https://wiki.flightgear.org/Atmospheric_light_scattering_FAQ (accessed Feb. 19, 2024).
[36] “Canvas,” Canvas - FlightGear wiki, https://wiki.flightgear.org/Canvas (accessed Feb. 19, 2024).
[37] “FGCOM 3.0,” FGCom 3.0 - FlightGear wiki, https://wiki.flightgear.org/FGCom_3.0 (accessed Feb. 19, 2024).
[38] “Howto:add sound effects to aircraft,” Howto:Add sound effects to aircraft - FlightGear wiki,
https://wiki.flightgear.org/Howto:Add_sound_effects_to_aircraft (accessed Feb. 19, 2024).
[39] “Howto:customizing flite TTS voices,” Howto:Customizing Flite TTS Voices - FlightGear wiki, https://wiki.flightgear.org/Howto:Customizing_Flite_TTS_Voices#cite_note-7 (accessed Feb. 19, 2024).
[40] “Space Shuttle,” Flightgear Forum • View Topic - Space Shuttle,
https://forum.flightgear.org/viewtopic.php?p=321125#p321125 (accessed Feb. 19, 2024).
[41] “Addon” FlightGear wiki, https://wiki.flightgear.org/Addon (accessed Feb. 19, 2024).
[42] “what is nasal” FlightGear wiki, https://wiki.flightgear.org/What_is_Nasal (accessed Feb. 19, 2024).
[43] “Weather,” FlightGear wiki, https://wiki.flightgear.org/Weather (accessed Feb. 19, 2024).