Abstract

This report contains our group's derivation of FlightGear 2020.3.19 flight simulator, which is an open source project which is maintained by various developers all over the world. Based on our examination of documentation found on FlightGear's Wiki and other scientific papers on the general construction of flight simulation software, we concluded that FlightGear uses an Implicit-Invocation architecture and consists of the following components:

- Runtime infrastructure - to support federates coordinating their operations,
- I/O system - for communications between the user and software,
- Graphical user interface (GUI) - displaying the actual game on your monitor
- Simulation model - simulating the actions of a virtual aircraft in a simulated environment
- Multi-player system - supporting the game's co-op features, and the
- Audio/sound system - producing the sounds heard throughout playing the game.

This report details the relationships between each of these components, their dependencies, the data flow between them, as well as the evolution of the system over the course of its existence. Additionally, we outline a few major use cases of this system that demonstrate its functionality in several common scenarios.

Introduction & Overview

Flight simulation is a multi-million dollar global industry addressing a wide number of training and leisure applications. The act of simulating flight and training pilots has been around almost as long as flight itself, when simulated flight was in its infancy pilots would use dummy equipment and manually simulated flight to train for the real thing. Nowadays, sophisticated computer systems connected to complicated flying apparatus are used to teach student pilots to fly safely in a simulated environment before potentially putting themselves and others at risk.

FlightGear is just one of many popular virtual flight simulation softwares but stands out for being open sourced and free for use. First released in 1996 and steadily improved upon since[17], it's open source nature has landed itself well to good documentation in its own wiki recording FlightGear's capabilities, features, history and more. After extensive research using this available knowledge, our team has identified the application as being composed of 7 main subsystems: runtime infrastructure (RTI), I/O system, graphics engine, simulation model, multi-player system, and audio/sound system. Using this information, we have further deduced that FlightGear has the Implicit-Invocation style of architecture. Implicit-Invocation can be succinctly described as an architecture involving a loosely coupled collection of components, each of which carries out some operation that may, in the process, enable other operations.

This report will aim to break down FlightGear's high-level architecture, identify and explain its corresponding subsystems, as well as visualize different use cases and how each subsystem interacts within those scenarios. We will also discuss the evolution of this specific system and the improvements made over its developmental journey. Finally, we will discuss how data is used throughout the system, the responsibilities of developers taking part in this project, and conclude with a summary of our findings and lessons learned.

1. Derivation Process

To acquaint ourselves with FlightGear and flight simulators in general, our team started by examining relevant academic sources covering the general development process of industry flight simulators as well as their general design and architecture. These research papers were named "Flight Simulator Architecture and Computer System Design and Research" (Guangda Liu, et al.) and "Architecture Development of Research Flight Simulator Based on COTS," (Zhao Guozhu, et al.) which gave us a good understanding of the reference architectures used in the development of flight simulators and their corresponding benefits and drawbacks.[18] [19]

FlightGear's online wiki was also one of the most important resources available to assist in understanding the software's architecture and functionality. It contained fairly detailed descriptions of the components that make up the software. Although, since this resource as well as the software are publicly maintained, this meant some information was missing in certain places and a lot of it was not kept very up to date. This meant a tremendous amount of interpretation and critical thinking was needed to try and put together the pieces of information found on the wiki to make a cohesive conceptual architecture of the software system.

2. Conceptual Architecture

FlightGear is constructed using the High-Level Architecture convention.[7] This is a general purpose architecture particularly used for distributed computer simulation systems. Additionally, this architecture style is a variant of the Implicit-Invocation software architecture style. This is particularly useful for applications that must be reconfigured quickly and often. Instead of invoking a procedure directly, the system works by allowing components to announce (or broadcast) one or more events which other components in the system can register an interest in by associating a procedure with the event. When a component announces an event, the broadcasting system (connector) itself invokes all the procedures that have been registered for the corresponding event. This means any component can be introduced into a system simply by registering it for the events of that system, not only does this make the code highly reusable, but it also eases system evolution providing easy addition and replacement of components without affecting the other components present in the system.

After analyzing FlightGear's high-level structure, our team has identified 6 main components that create the system's main functionality. These are: runtime infrastructure (RTI), I/O system, GUI, simulation model, multi-player system, and audio/sound system user interface (UI). Dependencies between these modules can be seen in the following diagram (Figure 1).

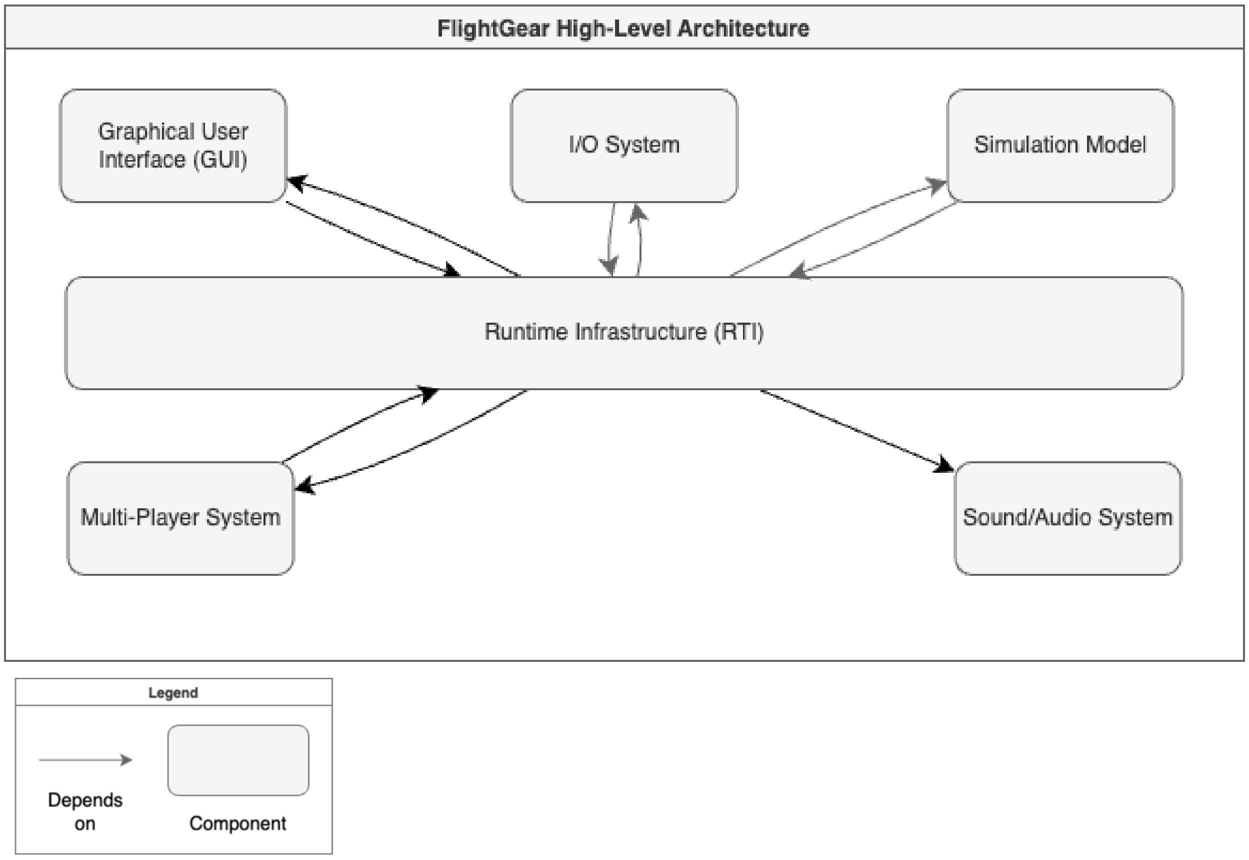
FlightGear High-Level Architecture
Graphical User Interface (GUI)
I/O System
Simulation Model
Runtime Infrastructure (RTI)
Multi-Player System
Sound/Audio System
Legend
Depends on
Component

Figure 1: FlightGear High-Level Architecture

The pivotal component of this architecture is the broadcasting system, which is called the runtime infrastructure. The RTI is primarily responsible for managing and coordinating/synchronizing distributed simulations. The RTI allows for the simulation to be split into many different pieces, providing a robust multi-threaded environment to take advantage of computers with multiple cores.[7] Additionally, this can even allow the simulation to be run on different computers all at the same time.

The other 5 components are called Federates in the High-Level Architecture convention, and make up the rest of the system.[7] These components send and receive messages from the RTI in order to trigger and be triggered by certain events. These components are what allow for the actual functionality of the flight simulator to be performed. Instead of having the software run under one monolithic simulation, this allows the system to be split up into different Federates, each having access to the full virtual process address space provided by the host OS instead of being forced to share it with other subsystems. The Federates are designed using the Federation Object Model (FOM) which helps define the objects and their updates which get published by the RTI.[7]

Subsystems (Modules)

2.1 Runtime Infrastructure

The Runtime Infrastructure component is the cornerstone of the FlightGear architecture. It acts as a middleware which is required in implementing the High Level Architecture convention. This component offers a set of services which are fundamental in supporting Federates' exchange of data during runtime execution as well as to coordinate their individual operations. FlightGear specifically uses OpenRTI which is an open source IEEE 1516 standard RTI (written by Mathias Fröhlich).[7] This system also took advantage of an interface library (also written by Mathias Fröhlich) which sits on top of the RTI and hugely simplifies interfacing with the RTI and Federates.[7]

2.2 I/O System

The I/O System component is responsible for handling communication between the users hardware and the software, translating user actions into valid inputs in the game. FlightGear allows users to use their own input devices such as joysticks, yokes, and rudder pedals to control aircraft within the simulator.[14] Each type of input device and its actions are recognized through the application using an XML file which helps describe what axis and buttons to be used to control which functions in FlightGear.

These associations between simulator functions and buttons are called "bindings." FlightGear has a large number of preset binding files made for most standard input devices but also allows users to create their own if their input device is not immediately recognized by the software.[14] The I/O system takes inputs given by valid input devices and sends messages through the RTI to carry out the appropriate actions to be taken within the Simulation Model and other Federates.

2.3 GUI

The GUI component is responsible for displaying the actual simulation and what is functionally possible for the user to do within the simulator. This can be partitioned into two separate components, the graphics engine and the user interface. The graphics engine is primarily responsible for mimicking the world's surroundings while in the flight simulator. The user interface (UI) is responsible for receiving input from the user through the visible graphical interface to trigger certain actions within the simulator.

Graphics Engine: The graphics engine is responsible for rendering the current state of the simulator, giving a pseudo-realistic representation of what flying a plane might actually look like. This involves rendering multiple different parts of the scenery in the following order: (1) Skydome, (2) Terrain and opaque objects, (3) Clouds, (4) Translucent objects, (5) lights, (6) Cockpit.[12] The rendering system used is primarily “one pass”, where the entire visible scenery is rendered in one single pass.[12] By default, The Graphics Engine is set up to use at least three cameras: Near camera, Far camera, and GUI camera.[12] The first two of which are used to render things like the scenery and the last one to render all visible 2D things, such as menus.

FlightGear takes advantage of a high performance 3D graphics toolkit called OpenSceneGraph during its rendering.[12] This is a library written entirely in Standard C++ and OpenGL and is well established as the “world leading scene graph technology, used widely in the vis-sim, space, scientific, oil-gas, games and virtual reality industries.” [20] OpenSceneGraph provides some important benefits to the rendering load experienced when running a heavy-weight system such as this. Benefits include models and scenery being able to load in separate threads, multi-monitor view, and improved precipitation modeling.

UI: The User Interface (UI) is primarily responsible for allowing the user to be able to customize certain simulator configuration options and interact with the simulation as a whole. These options can be accessed through the use of menus, dialogs, and keybindings to further customize and augment the functionality of the simulator.[12] The UI is strictly limited to elements which the user is able to interact with. For example, one is able to adjust the knob on the altimeter, but the barometer dial cannot be interacted with.[12] Users can interact with the UI using a number of different methods, these include: mouse location, mouse clicks, specifically mapped keystrokes, or a physical device which allows external physical actions to trigger events.

The GUI primarily receives messages from the RTI to update what needs to get rendered on screen given the changes in the Simulation model, I/O System, UI, and Multi-Player Systems. It additionally sends messages to the Simulation model to change the way it performs, depending on which options are changed within the UI.

2.4 Simulation Model

The Simulation Model component is responsible for handling the actions of the aircraft within the simulated environment. This includes; the Flight Dynamics Model (FDM), simulated weather, and the AI system which handles things like air traffic control, air traffic and ground traffic simulation.

Flight Dynamics Model (FDM): The flight dynamics model is the utilized set of math equations used to calculate the different physical forces acting on a simulated aircraft (e.g., thrust, drag, etc.).[10] This means every aircraft used in FlightGear needs its own bespoke FDM to accurately represent it within the simulator’s environment. FlightGear currently uses two different integrated FDMs: JSBSim and YASim. JSBSim supports simulation of airplanes, rockets, helicopters, and some lighter than air aircraft within the simulator.[10] It was originally developed specifically for FlightGear and has been the default FDM used since 2000.[10] YASim was introduced to FlightGear in 2002 and uses different calculation methods to additionally support airplanes and helicopters.[10]

Weather: Weather is simulated in FlightGear using one of two weather engines: Basic Weather (BW) and Advanced Weather (AW).[11] These engines accurately simulate real weather fetch, predefined weather scenarios, 3D clouds as well as many other weather related interactions. Basic Weather is very simple and straightforward to customize, but knows nothing about the effect of terrain on weather and applies the same weather conditions for the current position and for every part of the world.[11] Advanced weather is a bit more complicated, modifying all weather variables as a whole, keeping things closer to reality. It allows for the terrain to interact with the weather, allowing for clouds and wind to climb up mountains, and generate thermals consistent with the ground and clouds.[11] Additionally, AW can be set up to simulate a realistic weather distribution throughout the world.[11]

AI System: FlightGear operates a variety of semi-intelligent systems within the application to simulate things such as air traffic control (ATC), air traffic, and ground traffic interactions.[13] ATC is mainly used to prevent collisions of aircraft around the world and organize the flow of air traffic. FlightGear's implementation of AI ATC attempts to simulate these tower communications with pilots in the simulation.[13] AI air traffic is used to reflect the actual daily movements of aircraft between airports, giving FlightGear a much more realistic overall experience. Additionally, ground traffic such as cars, trucks, trains, and ships are approximated within the simulation to mimic their realistic movements and interactions seen in real life.[13]

This Simulation Model component sends messages using RTI to receive updates from components like the I/O and Multi-Player systems as well as publishing messages through the RTI to the rest of the 4 components.

2.5 Multiplayer System

The Multiplayer component of FlightGear is responsible for allowing players to see other pilots while playing the game. The Multiplayer component allows for additional gameplay features like flying in formation with other players, midair refuelling from tankers flown by other players and allows for whole new gameplay systems like player run air traffic control.[15] To connect to a multiplayer server, there are multiple ways, with the simplest being via the built-in launcher. All one needs to do is enter a corresponding "callsign", select a server from the menu, press connect and go online.[15] It's also possible for players to set up a local multiplayer instance between two simultaneous FlightGear sessions using a multiplayer server by mapping each I/O port of one instance to the corresponding port(s) or the other instance.[15]

The Multi-Player System Component receives incoming messages from the I/O and Simulation Model components to update the state of the simulation for other players playing at the same time. Conversely, it receives updates from the online multiplayer sessions, and sends update messages through the RTI to client side components of the simulation to keep it up to date with multiplayer actions.

2.6 Audio/Sound System

The Audio/Sound System component of FlightGear is responsible for relaying actions taken within the simulator to the user through their corresponding sound queues. Sound effects can be triggered conditionally, and the sound's pitch and volume can depend on properties in the simulation's property tree.[9] Sounds can be triggered by anything from effects from aircraft, to avionic sounds like navigation radio receivers. Additionally, FlightGear uses OpenAL, a cross-platform 3D audio API, for 3D positional audio cues to further the simulation's realistic immersion for users.[9] The Audio/Sound System component receives messages through the RTI from components such as the Simulation Model and Multi-Player System to create the appropriate sound cues from the gameplay being performed.

3. System Evolution

This section summarizes the changes made to the software with each new version from the initial code release to version 2020.3.18. Throughout the years, FlightGear has undergone a versioning format scheme, starting with a simple "#.#" system where the first number indicates larger changes to the whole system and the second number indicates smaller, less impactful changes. The versioning was later changed to add the year of release to the beginning, as well as resetting the versions back to 1.1, making the first version of this new system "2016.1.1". It seems this versioning system has been largely abandoned, as the most recent version, released in March 2023, still starts with 2020.

3.1 Initial Code Release (Not Versioned)

The first versions of the game were developed by Eric Korpela while he was working on his thesis. These earlier versions of the game were much simpler with no buildings, just poorly textured flat land with some mountains. One of its few complexities during this time, however, was the sky having accurate sun and moon positions as well as accurate lighting conditions. After finishing his thesis, development on this version largely halted due to Eric no longer working as the game's main developer. After this, development of a OpenGL version of the game started.[1]

3.2 FlightGear 1.0

A multiplayer text and voice chat system was implemented using Asterisk, which is an open source framework for building communication applications. There were also improvements to the FDMs, specifically YASmin, to allow for water planes, improved gliding, and a major overhaul of helicopters. There were also big improvements to the AI systems, with an emphasis on taxiing and ground operations of AI aircraft, as well as some smaller changes to ground networks at airports and boats.[2]

3.3 FlightGear 1.9.0

The entirety of FlightGear was switched from PLIB (Portable Game Library) to OSG (OpenSceneGraph). These libraries are responsible for many systems in the game, such as GUI widgets, audio drivers, Joystick support, networking, etc.[2]

3.4 FlightGear 2.0.0

There was a major overhaul of the sound systems of the game allowing for the Doppler effect, distance attenuation (things get quieter as it moves farther away), etc. the visual effects were also overhauled though not to the same extent as the sound systems, new 3D clouds were added, better lighting, dynamic water, etc.[2]

3.5 FlightGear 3.0.0

A new voice chat system was added called FGCom within the simulator. A new rendering system to allow for real-time shadows and lighting. Built-in on-the-fly terrain loading to allow for loading of terrain that is close to the player, instead of needing to load the whole map before playing. An advanced weather simulation was added to create more realistic weather, such as weather fronts, cloud formations based on mountain ridges.[3]

3.5 FlightGear v2016.1 "San Francisco"

With the v2016.1 update, the versioning scheme was changed from #.# to "year.# *update codename*". Also, a whole new launcher system was added to allow for the ability to download certain aircraft, allowing for a reduction of overall package size. The rendering system saw some improvements to allow for a more realistic cockpit, in-cockpit shadows, dynamic raindrops on the cockpit window, glass reflections, etc.[4]

3.6 FlightGear v2017.2 "Boston"

The YASim FDM received heavy development. The multiplayer system got some improvements to allow for a reduction in bandwidth.[5]

3.7 FlightGear v2020.3 "Keflavik"

Improvements to the JSBSim and YASim FDMs to allow for easier development, better debugging, etc.[6]

3.8 FlightGear Versions from v2020.3 to Present

Since the v2020.3 update, there has only been small bug fixes as the developers are in the process of overhauling a lot of the major systems such as moving to OpenGL core profile, a new world scenery system, a system to automate the population of the world map based on OpenStreetMap, a new rendering system, etc.[6]

4. Developer Responsibilities

FlightGear is openly available on the version control platform Github for developers around the world to modify and contribute to. Developers are classified by the FlightGear community as “people who build FlightGear from source code and make modifications to the source code.” [16] These types of developers can only provide patches to the current source code, which then needs to be reviewed, improved, then committed by core developers. A core developer is someone who has master commit rights to the FlightGear Github repository, giving them the final say on any code that will potentially be integrated with the main codebase.[16]

For any kind of developer contributing to the FlightGear project, a core understanding of the base package is essential. The base package is home to the core data of the simulator. This includes the scenery, aircraft, sounds, images/texture, configuration files, and scripts[16]. The majority of this data does not require or involve any programming from developers, the only exception being certain scripts and shaders. People who develop solely on the base package are non-programming developers, working solely on modelling elements such as scenery and aircraft (e.g., scenery developers, aircraft developers, etc.).[16] Additionally, configuration files can provide an interface to modify the simulator's behaviour without much prior programming experience.

This multifaceted approach to open-source development has allowed people of all skill levels and backgrounds to contribute to FlightGear over its extensive history. Furthermore, it has allowed development to continue in a relatively consistent and controlled manner, creating a strong foundation for new improvements in the years to come.

5. Data Flow

The data flow in FlightGear's simulation architecture is primarily managed through its High-Level Architecture (HLA) framework, which facilitates communication between the Runtime Infrastructure which coordinates as well as handles data exchange, and the components (Federates) which process data. The process starts with user input through hardware such as joysticks, yokes, and pedals. FlightGear's I/O system accepts these inputs and converts them into understandable commands for the simulation. These translated commands are then sent to the Runtime Infrastructure (RTI), where they are processed into various messages and events. The

RTI first forwards these messages to the Simulation Model (and the Multiplayer System if multiplayer is enabled), which uses them to update the state of the aircraft and its environment. The Flight Dynamics Model (FDM), is a component within the Simulation Model that calculates the aircraft's response to physical forces such as; thrust, drag, and gravity, all based on the current state of the plane. Within the Simulation model there are also the weather and AI components. For the AI component within the Simulation model, it uses the AI traffic files and ground traffic files (in XML format) to compute the current state of ATC, air traffic, etc. The current (/continuing?) state of the weather is also calculated.

The updated state of the simulation is sent from the Simulation Model to the RTI, which then distributes this information to the Graphical User Interface (GUI) and the Audio/Sound System. The GUI then uses this data to render the next frame that will be displayed, while the audio/sound system fetches and plays the sounds that have been triggered. When Multiplayer is enabled, the sounds and other users also need to be displayed. Data entailing the details of other players, such as their aircraft's position, orientation, actions, etc…is communicated to the RTI which receives these updates about each player's simulation state and broadcasts the appropriate messages. Typically, when the data for multiplayer arrives it goes to the GUI, which simulates how the other user's plane looks to you and the audio/sound component for any sounds the other user may have triggered. This allows for a synchronized multiplayer experience where players can interact with each other's aircraft and the shared simulation environment in real-time. More on the specifics about how data flows throughout the architecture is covered in section 7. Sequence diagrams.

6. Concurrency

Concurrency is present in high-level architecture (HLA) for distributed simulation systems, primarily in the FlightGear simulation model. The HLA of the FlightGear simulation is split into different federates, which are separate components, each serving a unique purpose. These federates interact with each other through a run-time infrastructure, which handles the communication and synchronization between them. Each federate runs its own thread and processes, giving them all access to the entire virtual process address space provided by the OS instead of sharing it with other subsystems. Additionally, each federate running on its thread and process allows them to run concurrently. The ability to execute concurrently allows for three advantages over a monolithic simulation. Primarily, the technique means that the different parts of the simulation can be processed simultaneously, which takes advantage of multi-core processors and allows for the ability to run on different computers. Furthermore, it allows FlightGear to separate parts of the simulator, such as AI, the FDM, Nasal Scripting and Renderer, from each other along with less time-critical subsystems, such as weather, so that FlightGear can get consistent frame rates[7] Finally, the federate technique provides an excellent framework to allow anyone to program new components that can interact with the FlightGear simulation in languages other than C/C++, such as Ada, Python, and Java. These new components can run on their own threads, making them easier to debug and troubleshoot without modifying or rebuilding FlightGear. Ultimately, this concurrent execution of federates is one of the key advantages of HLA, as it allows for more efficient use of computational resources and can lead to improved performance and responsiveness of the simulation.

7. Sequence Diagrams

7.1 Use case #1: User GUI Control

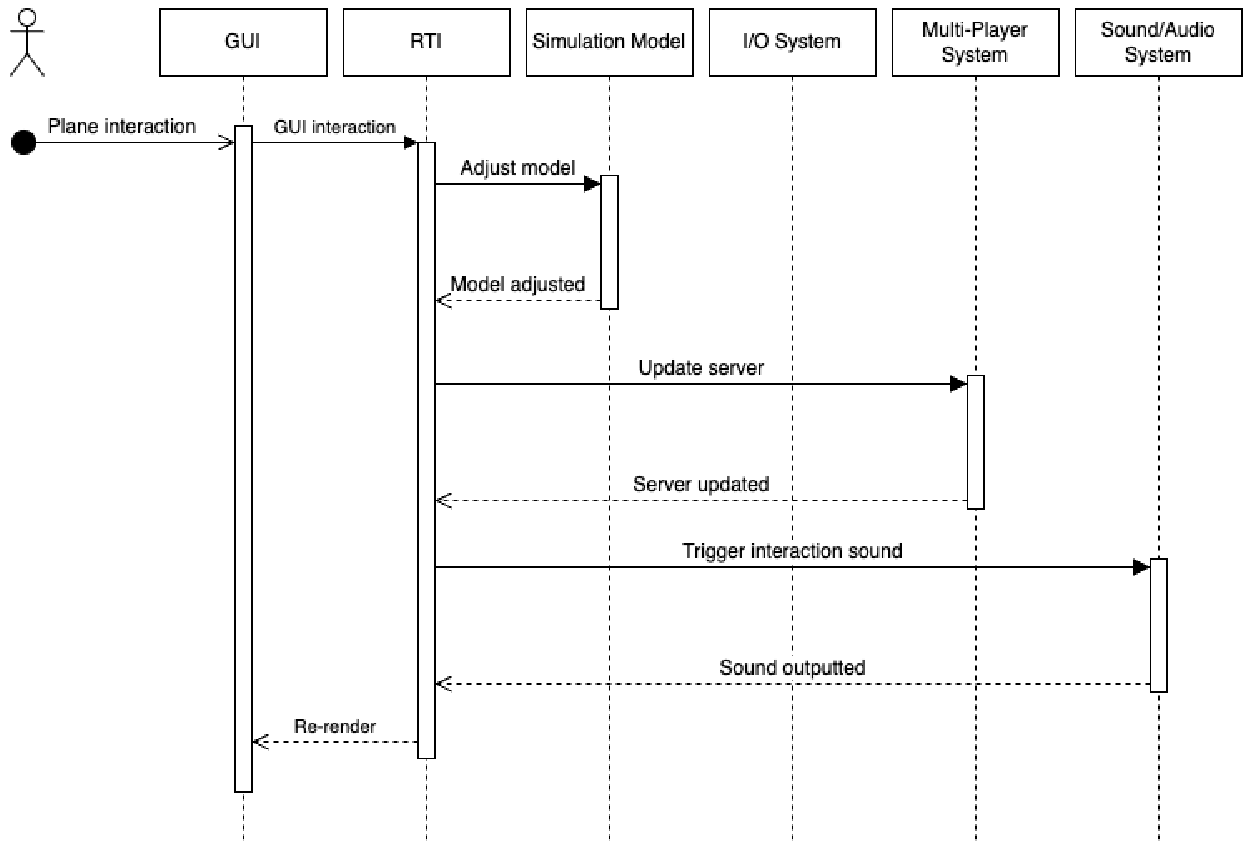

Figure 2: FlightGear User GUI sequence diagram

The most common interaction use case in FlightGear is when a user interfaces in some way with the GUI of the simulation. This interaction can take place in a variety of ways, such as through flipping a switch on the plane's control panel or changing the plane's direction using the steering controls. This use case starts off with the user (actor) causing some action using the GUI, which then triggers a message to be sent to the RTI to update the affected components. First, The RTI sends a message to update the simulation model given the changes to the plane's behaviour due to the GUI change. Second, the Simulation Model sends a response message back to the RTI triggering any updates to the simulator's state that might need to be communicated to any online servers the simulator is currently connected to. Thirdly, the Multi-Player System sends a response message back to the RTI to trigger any audio queues that need to be heard by the user due to the interaction with the GUI. Finally, the Sound System sends a message to the

RTI announcing that the sound has been triggered, allowing the RTI to send a message to re-render the GUI given the updated state.

7.2 Use case #2: User I/O Control

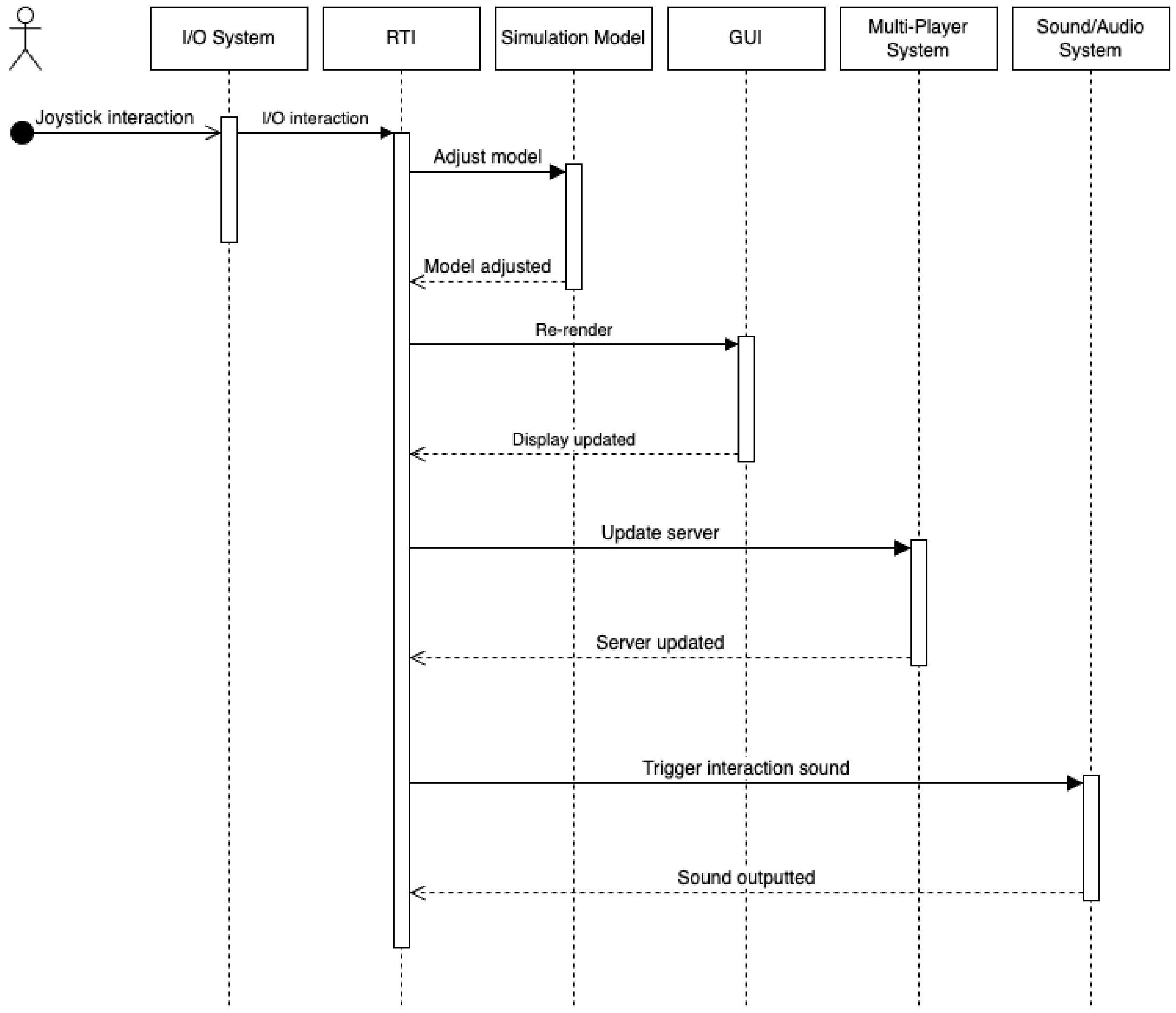

Figure 3: FlightGear User I/O sequence diagram

The second most common interaction use case in FlightGear is the interaction between the simulator and an external input device such as a joystick. This use case is triggered by any valid input using an external input device. This causes the I/O system to send a message to the RTI, notifying it that some I/O interaction has been made. The RTI then sends out a series of messages to update the affected components. First, a message is sent to the simulation model to make any adjustments caused by the corresponding external input. Second, the simulation model sends a message back to the RTI triggering the GUI to be re-rendered using the updated simulation model. Third, the GUI then sends a message to the Multi-Player System with the updated state of the simulation to be communicated to any connected simulations. Finally, the

Multi-Player System notifies the RTI to output any sounds corresponding to the interaction made using the external input device.

8. Conclusion/ Lessons Learned

In conclusion, our analysis of the FlightGear 2020.3.19 flight simulator has provided a comprehensive understanding of the design of its architecture and subsystems, rooted in the Implicit-Invocation style. Through our investigation, we determined that FlightGear consists of six key components: runtime infrastructure (RTI), I/O system, graphical user interface (GUI), simulation model, multi-player system, and audio/sound system. These components work in concert, leveraging a loosely coupled collection of modules, to enable a dynamic and adaptable simulation environment. The utilization of the High-Level Architecture convention facilitates the system's flexibility and scalability, allowing for efficient real-time interaction and simulation. Our study not only highlighted the system's intricate design and functionality but also emphasized the collaborative effort required to maintain and evolve this open-source project. The process of dissecting FlightGear's architecture has enriched our team's understanding of distributed simulation systems and underscored the importance of community-driven development in advancing technological innovation.

References

1. https://wiki.flightgear.org/FlightGear_history#Versions_0.7%E2%80%930.9_(2001%E2%80%932003)
2. https://web.archive.org/web/20180410061153/http://www.flightgear.org/version.html
3. https://wiki.flightgear.org/Changelog_3.0
4. https://wiki.flightgear.org/Changelog_2016.1
5. https://wiki.flightgear.org/Changelog_2017.2
6. https://wiki.flightgear.org/Changelog_2020.3
7. https://wiki.flightgear.org/High-Level_Architecture#Federates
8. https://wiki.flightgear.org/Flightgear_UI_overview
9. https://wiki.flightgear.org/Howto:Add_sound_effects_to_aircraft
10. https://wiki.flightgear.org/Flight_Dynamics_Model
11. https://wiki.flightgear.org/Weather
12. https://wiki.flightgear.org/Rendering_system_improvements
13. https://wiki.flightgear.org/Artificial_intelligence
14. https://wiki.flightgear.org/Input_device
15. https://wiki.flightgear.org/Howto:Multiplayer
16. https://wiki.flightgear.org/Howto:Understand_the_FlightGear_development_process
17. https://wiki.flightgear.org/FlightGear_history
18. https://ieeexplore.ieee.org/document/5364558
19. https://ieeexplore.ieee.org/document/9336527
20. https://www.openscenegraph.com/