Assignment 2: FlightGear Concrete Architecture Report

March 25, 2024

Group 1

Matthew Susko

Marcus Dipalma

Mark Kennedy

Darcy Mckinnon

Armaan Wander

Abstract

This report presents a detailed analysis of the architectural framework of FlightGear, a sophisticated flight simulator. It commences with a revision of the conceptual architecture, identifying enhancements and integrations that align with the simulator's operational requirements. The report then progresses to the derivation of the concrete architecture, employing systematic code analysis to map the theoretical framework to the actual system implementation. A critical examination of the interactions between various subsystems within the Main Loop component is undertaken to elucidate the network of dependencies and communication pathways. Subsequent sections provide a reflexive analysis, scrutinizing the congruence between the conceptual and concrete architectures and identifying discrepancies. The rationale behind architectural decisions, particularly those leading to divergences between the envisioned and realized structures, is thoroughly explored. In addition, the report delves into a second-level subsystem, offering an in-depth review of its internal architecture and interaction dynamics. Complementing the technical analysis, two sequence diagrams for non-trivial use cases are included, showcasing the practical application of the architectural components in real-world scenarios. The report concludes with a reflective discussion on the lessons learned throughout the analytical process, encapsulating the insights gained from the comprehensive examination of FlightGear's architectural design. Through this structured approach, the report aims to provide a holistic understanding of the FlightGear simulator's architecture, contributing to the broader knowledge base in the field of software architecture and system design.

1. Introduction

The comprehensive analysis and revision of FlightGear's architectural framework present a meticulous journey through the evolution of its system design. This paper encapsulates the team's endeavor to refine the initial Implicit-Invocation architecture into a more robust Client-Server model, reflecting the dynamic needs of the FlightGear simulator, particularly its multiplayer functionality. Through a methodical examination of the system’s components, their interactions, and dependencies, this report delineates the transition from a conceptual architecture to a concrete, operational model. Enhanced by the integration of new components and the strategic redefinition of existing ones, the study provides an in-depth understanding of the FlightGear simulator's architecture. It underscores the significance of a holistic approach in architectural design, demonstrating the intricate interplay of various elements that contribute to the functionality and performance of the system.

2. Revised Conceptual Architecture

In refining the conceptual architecture initially outlined in our first assignment, our team has implemented significant modifications and enhancements. Our original framework was

characterized by an Implicit-Invocation design, incorporating six primary components: Runtime Infrastructure, I/O System, GUI, Simulation Model, Multiplayer System, and Audio/Sound System. Given the inherently multiplayer aspect of the FlightGear simulator and the requisite central server communication for this functionality, we have transitioned to a Client-Server architectural style.

This shift has also led us to identify and integrate additional components that were either previously mentioned without detailed individual consideration or entirely omitted from our initial analysis. These newly incorporated elements are the Autopilot, Airports, Main Loop, Networking Component, and Server Component.

As a result of these developments, our comprehensive understanding of FlightGear's conceptual architecture now encompasses 16 distinct components: (1) Air Traffic Control (ATC), (2) Environment, (3) Navigation Aid, (4) AI Model, (5) Input Manager, (6) Scenery Manager, (7) Aircraft, (8) Flight Dynamics Model (FDM), (9) Multiplayer, (10) Sound System, (11) Cockpit, (12) GUI, (13) Auto Pilot, (14) Airports, (15) Networking Component, and (16) Server Component.

This expanded and refined architecture offers a more holistic view of the FlightGear simulator, ensuring a robust and integrated simulation experience. Our new conceptual architecture with the additional components is as follows:

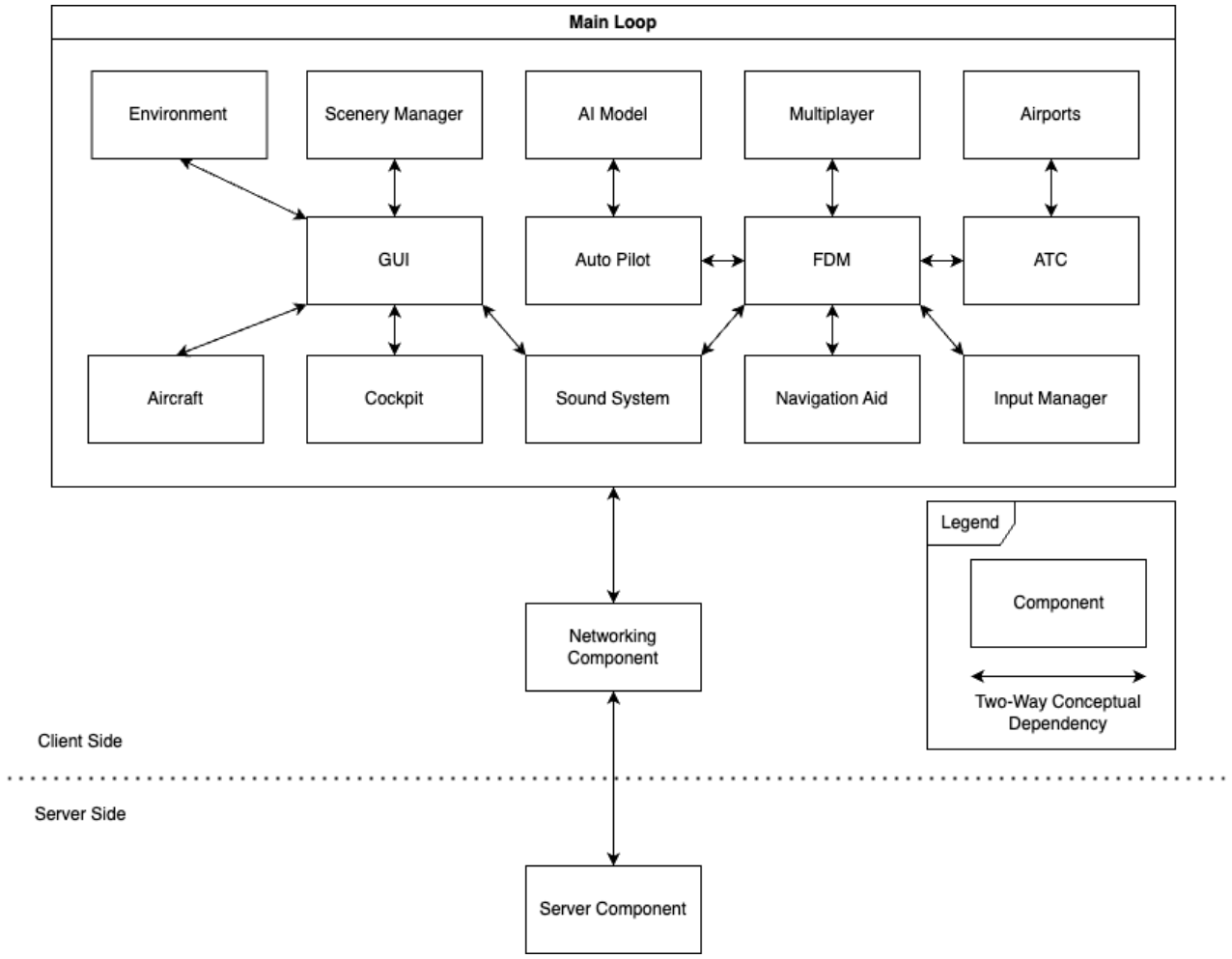

Figure 1. Conceptual dependency graph

In refining our conceptual framework, our team has introduced several enhancements and new elements to our architecture. A pivotal addition is the Main Loop module, which serves as the core of our updated structure. It encapsulates virtually every client-side component, mirroring the dynamics of a game loop—encompassing frame updates, weather modifications, and more. This integration aligns with our vision to encapsulate the simulator's operations within a singular, cohesive loop.

Additionally, we have integrated a Networking Component, establishing a robust conduit for communication between client and server interfaces. This addition marks a significant pivot from our earlier architecture, which, due to a different architectural direction, did not incorporate such a component.

Within the Main Loop, we have also introduced a suite of components previously discussed but not defined as standalone entities in our past reports. These include the Aircraft, Environment, Scenery Manager, AI Model, Cockpit, and Server components. Upon further review of the

FlightGear documentation, we identified additional components that were not previously acknowledged. These are the Airports, Auto Pilot, Navigation Aid, and Main Loop components, which we now recognize as essential elements of our enhanced architectural framework.

The Airports component serves as the central hub for managing information related to all airports within the FlightGear simulator, meticulously tracking each airport's attributes such as availability and activity levels. Meanwhile, the Auto Pilot component is engineered to ensure the safe, autonomous navigation of aircraft, showcasing FlightGear's sophisticated flight simulation capabilities. Navigation Aid components are designed to assist users in following both visual and instrumental navigation guidelines, enriching the FlightGear experience with enhanced realism and guidance.

In our refined architecture, we've strategically established connections between both newly introduced and existing components within the Main Loop. This was undertaken to seamlessly replace the functionality previously provided by the Run Time Infrastructure (RTI) component, a cornerstone in our initial conceptual architecture from Assignment 1. The GUI component, pivotal for user interaction, facilitates a dynamic two-way relationship with the Aircraft, Cockpit, Environment, and Scenery Manager components. This integration ensures all necessary components are rendered and visible to the user, enhancing the immersive experience. Additionally, the GUI forms a crucial link with the sound system, enabling the auditory feedback of GUI-generated sounds.

The Flight Dynamics Model (FDM) component emerges as a foundational element, establishing bi-directional dependencies with the ATC, Auto Pilot, Navigation Aid, Sound System, and Input Manager components. This interconnectivity underscores the FDM's critical role in simulating realistic flight dynamics, influencing various aspects such as sound reproduction, multiplayer interactions, and user input responsiveness.

Moreover, the ATC components forge a bi-directional link with the Airports component, reflecting their intrinsic synergy and mutual dependence for seamless communication and functionality. The Auto Pilot and AI Model components also share a two-way dependency, a testament to their closely aligned functionalities and shared objectives in automating flight paths and behaviors, underscoring the intricate web of interdependencies that define the FlightGear simulator's operational excellence.

The conception of these new connections was informed by an in-depth exploration of the FlightGear Wiki [1], along with valuable insights drawn from analogous flight simulator systems [2] [3].

3. Derivation Process

To derive the concrete architecture of the FlightGear system, we started with an analysis of the source code using the Understand tool [4] – a tool that allows you to perform code analysis, view dependency graphs, etc. to help aid in analysis and understanding of a project's subsystems. We started by mapping the FlightGear systems to our new conceptual architecture formulated above. Using the dependency graphing feature in Understand, we created a directed graph of the dependencies that the systems have on each other as shown in figure 2 below. Here, the blue lines represent one-way dependencies, while the red lines represent two-way dependencies.

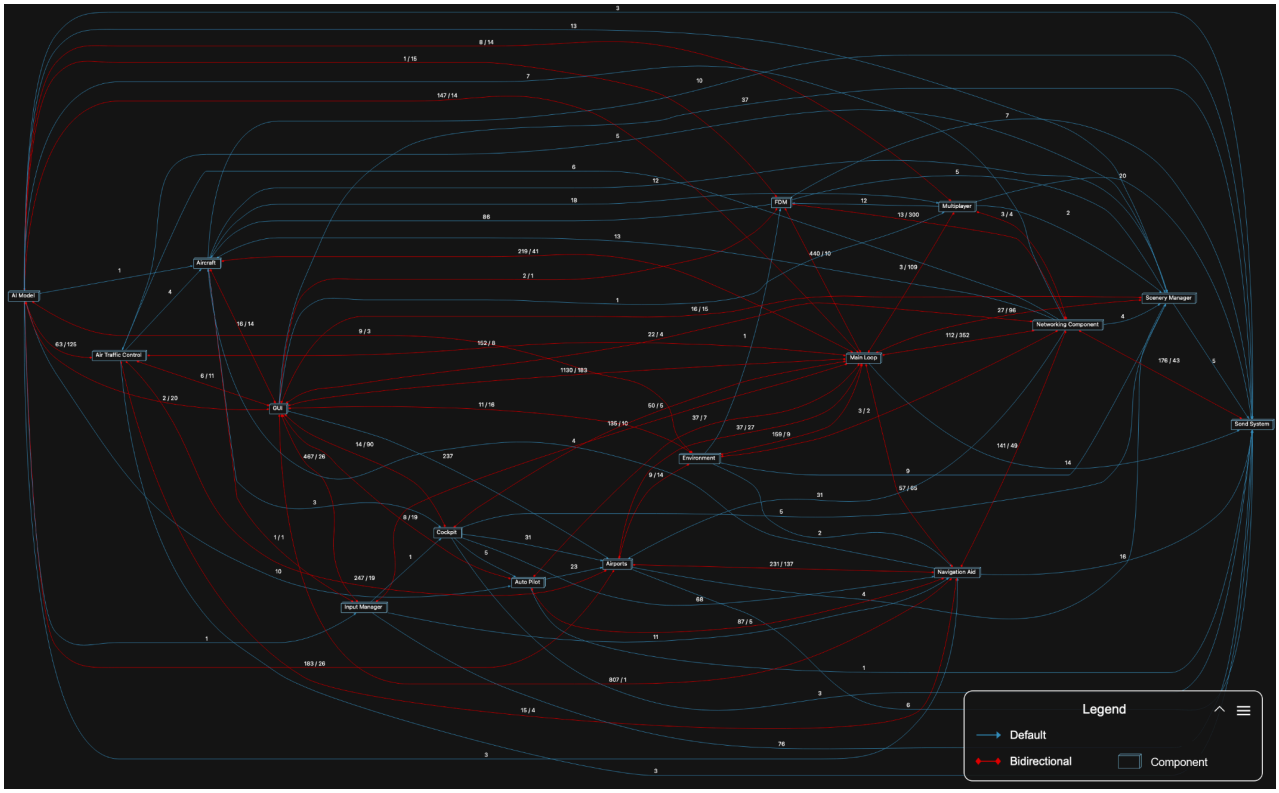

Figure 2. Concrete dependency graph from the Understand tool

The utilization of the Understand tool has significantly enhanced our analytical capabilities, enabling us to meticulously categorize the source code of FlightGear into distinct segments corresponding to each of its components. This strategic organization allowed us to identify and map out the intricate network of dependencies, such as function calls, that interlink these components. By doing so, we've been able to construct a detailed dependency graph that serves as a visual representation of these relationships.

This comprehensive approach has not only deepened our understanding of the interdependencies among the components but has also uncovered previously unrecognized connections. Such insights are invaluable, as they illuminate the complex architecture of

FlightGear, providing us with a clearer perspective on the system's operational dynamics and potential areas for optimization.

4. Final Concrete Architecture

Based on our team's derivation process above, which adds multiple interdependencies between components within the Main Loop component, our final concrete architecture is as follows:

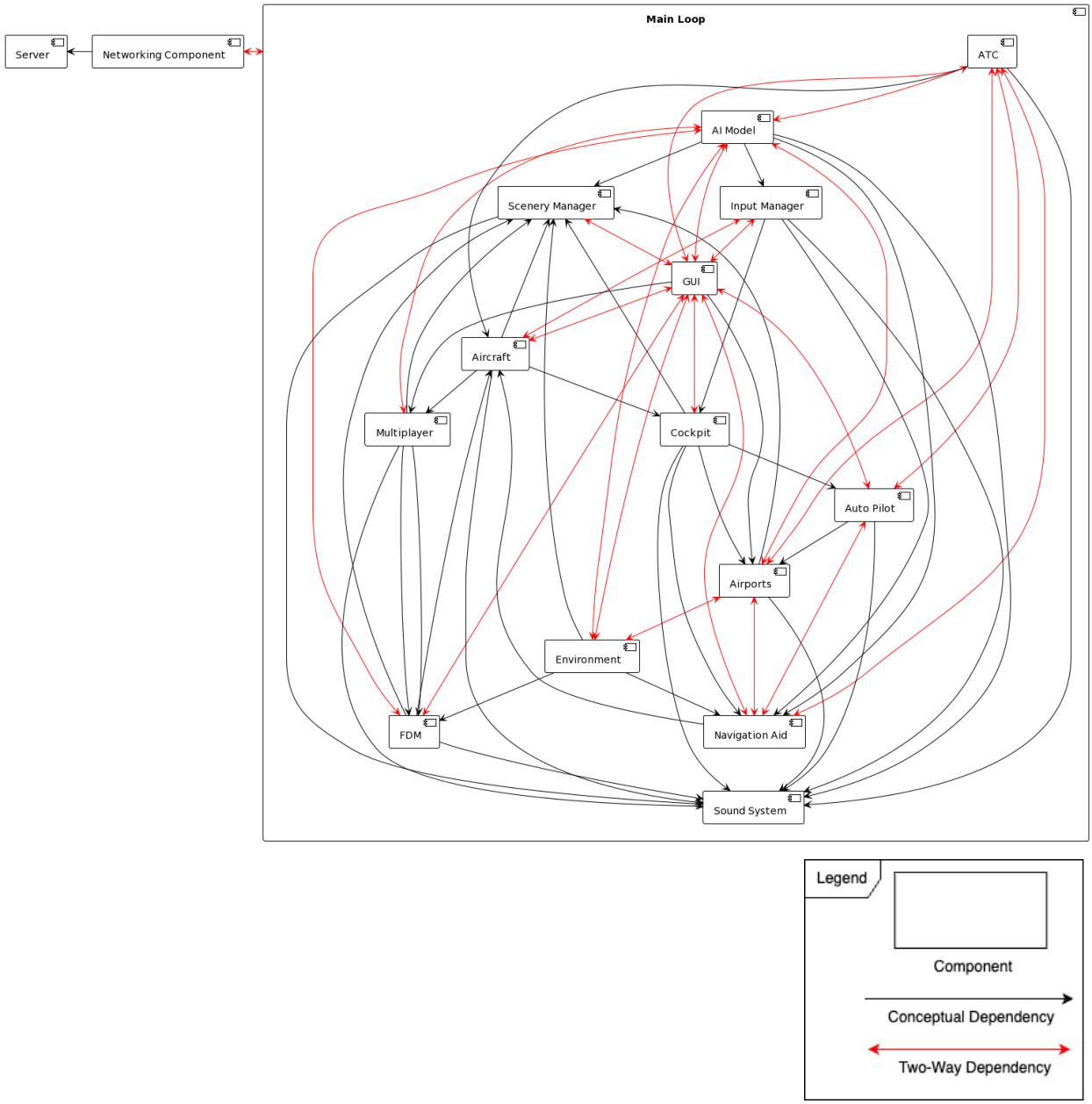

Figure 3. Concrete dependency graph

In Figure 2, the diagram illustrates the relationships between various components within our architectural design. Red arrows depict bidirectional conceptual dependencies, indicating a mutual exchange of information or functionality between components. Conversely, black arrows represent unidirectional dependencies, where one component relies on another without reciprocal interaction. This network of connections, especially prevalent within the Main Loop, is critical for facilitating seamless communication and ensuring the effective operation of the simulation.

Despite the introduction of several new components to solidify our architectural framework, it's important to emphasize that our revision has not led to the elimination of any components initially projected. Rather, we have undertaken some renaming and expansion of existing components to better align with our refined understanding and objectives. These foundational elements from our original conceptual framework remain integral to our more detailed and concrete architectural model.

5. Reflexion Analysis for High-level Architecture

In this section we will be discussing the new dependencies that were not present in our conceptual architecture. There will be an explanation for each and rationale to explain why they appear in the concrete architecture.

FDM (↔ AI Model , ↔ GUI , → Scenery Manager, → Aircraft)

For the FDM, there are multiple new dependencies. The AI Model integration simulates realistic autonomous aircraft behavior; GUI interaction displays real-time flight and environmental data from the FDM; Scenery Manager enhances flight realism with dynamic environmental effects; Aircraft dynamics ensure accurate performance and responsiveness for diverse aircraft models.

Multiplayer(↔ AI Model, → Scenery Manager)

For the Multiplayer system, integrating AI models enriches realism by adding autonomous aircraft to navigate alongside players which mimics real-world traffic scenarios through internet or local network communication. Similarly, the Scenery Manager ensures multiplayer sessions share consistent environmental settings like weather and terrain increasing realism and fairness for all players.

Scenery Manager → Sound System

The Scenery Manager's connection with the Sound System enhances simulation, linking visual environments to appropriate sounds. This ensures players' auditory experiences match the scenery, like hearing ocean waves near beaches, adding depth to the flight experience.

Navigation Aid(↔ Airports, ↔ ATC, → Aircraft)

The Navigation Aids (Navaids) like NDBs and VORs guide aircraft to Airports and assist ATC in airspace management, ensuring safe and efficient flight paths. ATC uses Navaids to direct the aircraft, maintaining separation and efficient routing. The Aircraft relies on onboard instruments to interpret these signals for navigating airways and approaches accurately.

Airports(↔ Environment, ↔ AI Model, ↔ ATC, → Sound System, → Scenery Manager)

The new dependencies in Airports are connected with the environment, AI behavior, ATC, sound systems, and the Scenery Manager. They reflect real-world conditions and geography, host AI aircraft operations, interact with ATC for navigational guidance, contribute to the game's soundscape with airport-related noises, and are detailed through the Scenery Manager for visual realism.

Auto Pilot(↔ Navigation Aid, ↔ GUI, ↔ ATC, → Airports, → Sound System)

The Autopilot component has a dependency with Navigation Aids for route following and instrument approaches. The GUI controls pilot interaction which allows for detailed control and mode selection. Interaction with ATC influences flight paths in response to instructions for airspace management. Accurate airport data is important for the autopilot during Airports approaches and departures. The Sound System enhances situational awareness with auditory cues for auto pilot actions.

Cockpit(→ Airports, → Auto Pilot, → Sound System, → Navigation Aid,→ Scenery manager)

For the Cockpit component there is an unexpected dependency with Airports for representations of realistic navigation and operations within the cockpit from terrain modeling libraries. The autopilot system's dependency keeps flight control directly from the cockpit. Sound systems powered by libraries like OpenAL are there to provide auditory feedback for cockpit alerts and environmental awareness. Navigation aids are tied into cockpit instruments for route planning and execution. The Scenery Manager's delivery of visually accurate landscapes aids in cockpit-based visual flight rules (VFR) navigation.

Input Manager(↔ GUI, ↔ Aircraft, → Navigation Aid, → Sound System, → Cockpit)

The GUI integration ensures user inputs dynamically affect the simulation. Aircraft control dependency reflects real-time user actions for realism. Navigation aids are crucial for user-led in-flight guidance. The sound system provides immersive feedback related to user inputs. Cockpit interactions allow realistic operation of aircraft systems by the user.

GUI(↔ Navigation Aid,↔ ATC,↔ AI Model,→ Airports,→ Multiplayer)

For the GUI component there is new dependency with Navigation Aids, managing waypoints and flight paths to navigate the skies. It also serves as an interface for ATC interactions where pilots communicate with controllers for organized flights. The GUI extends functionality to controlling AI Models allowing for dynamic virtual environments. GUI to Airports provides necessary information for takeoffs, landings, and taxiing. The Multiplayer functionality through the GUI connects pilots online.

ATC(↔ AI Model,→ Aircraft,→ Sound System)

The integration of an AI Model with the Air Traffic Control (ATC) system in FlightGear allows for the simulation of intelligent behavior by non-player characters (NPCs) such as other aircraft in the vicinity. The dependency between ATC and Aircraft provides instructions, clearances, and information to pilots. It might be created through direct communication channels and protocols within the simulation, allowing for real-time interaction between the pilot (player) and ATC. The Sound System dependency makes sure that communications from ATC are audibly presented to the player.

Aircraft(→ Multiplayer,→ Scenery Manager,→ Sound System,→ Cockpit)

The Aircraft to Multiplayer dependency allows for real-time interactions between users flying in the same virtual space which use networking protocols to synchronize aircraft states across multiple instances of FlightGear. The Aircraft to Scenery Manager dependency makes sure that the virtual environment is dynamically rendered in response to the aircraft's movements which provide visual cues essential for navigation. The integration with a Sound System introduces realistic auditory feedback from engine noises to environmental sounds. The Aircraft to Cockpit dependency facilitates user interaction with the simulation through detailed cockpit instruments and controls.

AI Model(↔Environment,→Sound System,→Scenery Manager,→Input Manager,→ Navigation Aid)

The AI component has an unexpected dependency with the simulation's environment and systems. They follow predefined flight plans and weather patterns which increase the environmental immersion. Sound effects are tied to the AI models like aircraft noise or weather sounds to possibly increase realism. These models navigate through detailed sceneries, including airports and terrain to mimic the behavior with the virtual world's geography. Although no strong rationale, user inputs can indirectly affect AI behavior through simulation control settings. AI

models use navigation aids for sticking to the simulated world's navigational structures for realistic flight execution.

Environment (→ FDM, → Scenery Manager, → Navigation Aid)

The concrete Environment component introduces dependencies like JSBSim and YAsim for realistic flight dynamics, accommodating a range of aircraft and flight scenarios. TerraGear and OpenSceneGraph which support complex 3D scenery, enhancing realism with detailed environments. Lastly, SimGear aids in navigation accuracy for realistic pilot training and educational purposes by managing simulation data and supporting real-world navigation systems within the flight environment

Multiplayer → Sound System

For Multiplayer, the dependency with Sound System enhances immersion by sharing sound effects based on multiplayer properties, like engine noises or environmental sounds, across the network. This setup creates a synchronized auditory experience for players which adds depth to the virtual aviation environment. Customizable through PropertyList XML files, these sound configurations allow for detailed adjustments.

6. Second Level Subsystem: Scenery and Environment

The scenery and environment subsystem is responsible for the simulated world that the player views. There are multiple methods on how the data for the virtual world can be obtained being either locally downloaded or Terrasync. This is as when you initially download the game, there is not much scenery there and it’s only through manually downloading more or TerraSync that you can add more scenery. However as the manual download requires a lot of space and takes a lot of time to download, TerraSync is the option preferred by most players, this is the method for how the data will be obtained that we will focus on.

6.1. TerraSync

When terrasync is enabled it reads your position and direction and downloads the necessary files as you fly around in the virtual world of FlightGear. Terrasync operates in the background downloading files from a regular HTTP server with a well known directory structure. Once TerraSync has acquired the appropriate files related to your position, it sends the information to TerraGear, which we will touch upon shortly. The architecture within the scenery and environment subsystem is Client-Server, this is not to say the whole module is like this but this is how the TerraSync portion does.

6.2. Scenery & Environment Subsystem

The scenery subsystem renders by dividing the world into tiles to stop it from rendering unnecessary areas. These tiles are aligned with the longitude and latitude lines. Each tile has a point of reference in its center, establishing a local coordinate system for the placement of each object within the tile. When rendering a scene the scenery manager loads all visible tiles translating each near origin to reduce floating-point precision issues. By moving the tiles close to the origin while preserving their relative positions and orientations, FlightGear minimizes errors that can occur due to limited floating-point accuracy.

The base terrain rendered is a 3D mesh, with different textures painted over it, then 3D objects are rendered overtop. The environment subsystem handles the environment settings which are set by the environment variables. Some examples of environment variables for FlightGear include:

FG_ROOT: This setting refers to the path of the main data directory for FlightGear, most importantly for the scenery module it contains the *base package*. Which contains all the default aircraft, default scenery, sounds, ed models, etc.

FG_AIRCRAFT: Specifies additional aircraft directory paths that FlightGear can use to access aircraft models other than the default directories. By setting this variable, users can provide FlightGear with alternative paths to locate aircraft models that have been created or downloaded.

$FG_HOME: Indicates the main location where user-specific FlightGear data is stored, not application data. This variable is where FlightGear writes information between sessions, including configuration/preferences, properties marked with user-archive, and aircraft-specific settings.

7. Use Cases

Based on the architecture, we have come up with two use cases.

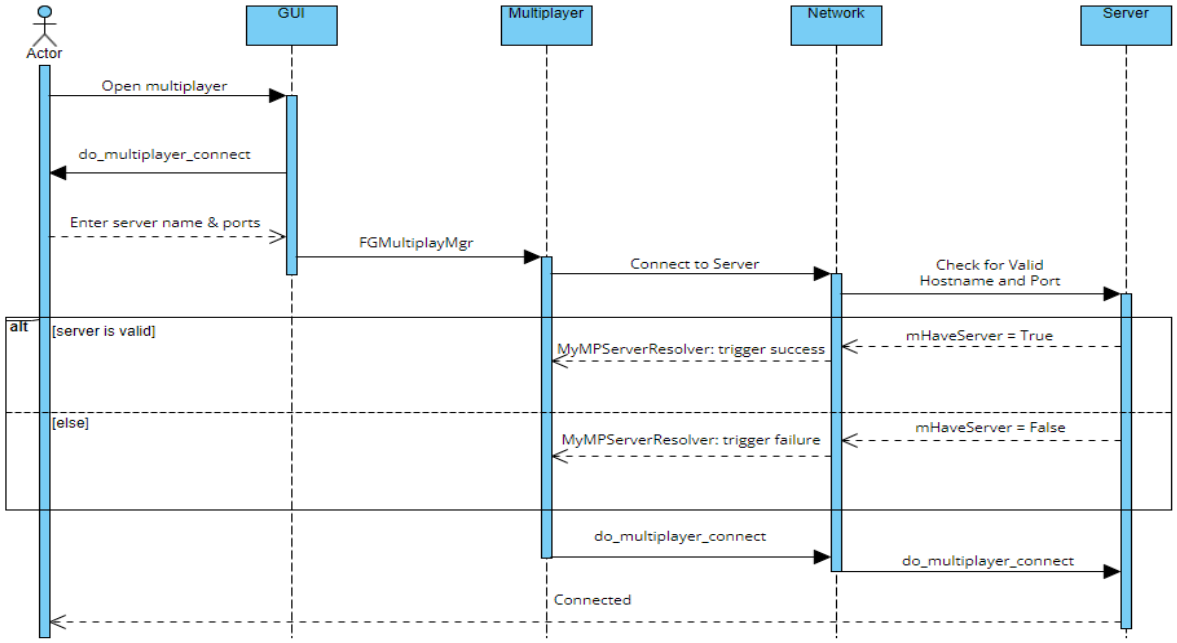

Figure 4. Use case for connecting to a multiplayer server

This use case will start when the user decides to launch Flightgear with the multiplayer option checked. To connect to a multiplayer server, the client will request the user to enter a callsign, hostname, and in/out port (both set to 5000). The hostname is in the form "mpserverXX.flightgear.org" where XX is the server number. The client checks with the server to see if the hostname and ports are valid and if so, then the user is connected to the multiplayer server.

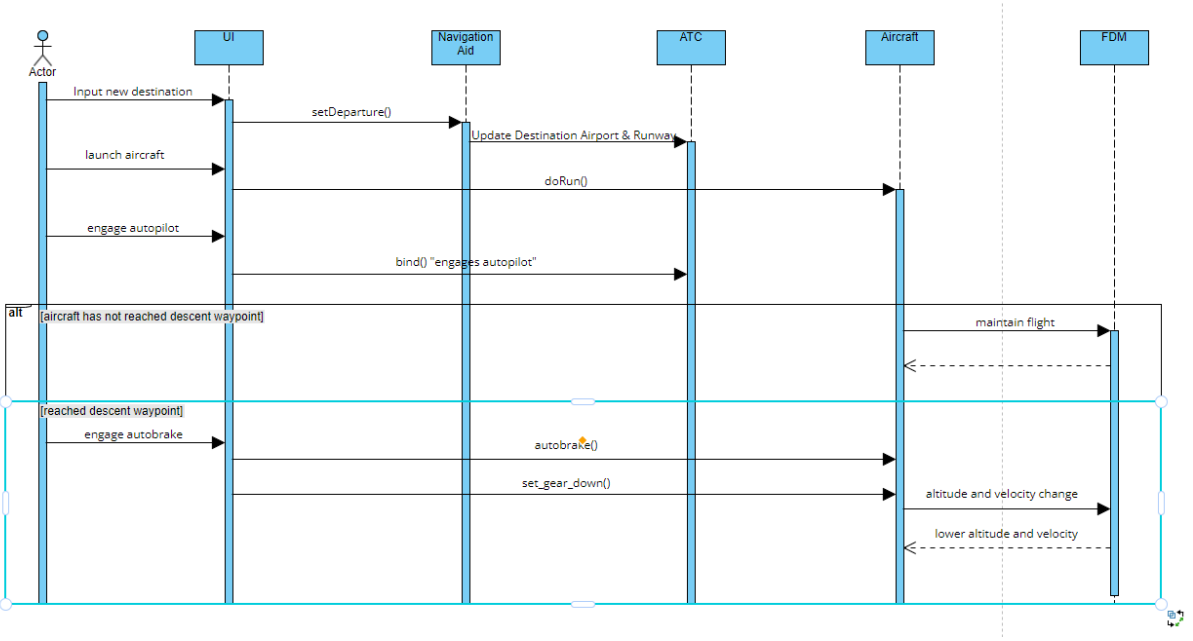

Figure 5 Use case for auto landing an aircraft

This is the use case for auto landing an aircraft. The user begins the flight by entering a destination, then launching the aircraft, then engaging the autopilot. When a destination is inputted, ATC updates its destination, airport, and runway for the flight. The aircraft is then launched and autopilot is engaged when in the air. While needed, the aircraft will maintain flight until it has to begin its descent. When descent begins, the aircraft's autobrake and landing gear are engaged to prepare for a landing.

8. Conclusion and Lessons Learned

The conclusion of our detailed analysis on FlightGear's architectural framework emphasizes the intricate balance and interconnectivity of its components, which together create a sophisticated flight simulation experience. Transitioning from an Implicit-Invocation design to a Client-Server architectural style has been a critical step in aligning the simulator's structure with its operational needs, particularly in supporting multiplayer functionality. This shift, accompanied by the integration of additional components, has significantly expanded our understanding of the system's architecture. Throughout the analysis, the systematic derivation of the concrete architecture, using tools like Understand, facilitated a direct correlation between the theoretical models and the actual system implementation. This process not only confirmed the initial architectural style but also revealed the complex network of dependencies and interactions within the simulator, especially within the Main Loop. From this comprehensive study, several key insights were gained. The process underscored the importance of adopting a holistic perspective in architectural analysis, considering not just individual components but their dynamic interrelations within the entire system. It also highlighted the evolving nature of system design, where initial conceptual frameworks undergo refinement and adaptation when confronted with practical implementation challenges. Moreover, the effectiveness of analytical tools in bridging the gap between conceptual and concrete architectures was evident. These tools not only aid in visualizing and understanding complex dependencies but also in validating the architectural decisions made during the design phase. In conclusion, the journey through FlightGear's architectural analysis has been enlightening, offering a deeper appreciation for the nuanced and layered aspects of software architecture and system design. The lessons learned extend beyond the specifics of FlightGear, providing valuable insights into the principles of architectural design and analysis that are applicable in a broader context.

References

1. https://wiki.flightgear.org/
2. https://ieeexplore.ieee.org/document/5364558
3. https://ieeexplore.ieee.org/document/9336527
4. https://scitools.com/