FlightGear Conceptual Architecture

Assignment 1 Report
CISC/CMPE 322
Cloudy with a Chance of Pilots
Group 19

William Ban: 20287469 - 20wb3@queensu.ca
Vasuki Elamaldeniya: 20175895 - vasuki.elamaldeniya@queensu.ca
Sara Jaffer: 20290672 - 20sj68@queensu.ca
Manraj Juneja: 20288003 - 20msj5@queensu.ca
Rishikesh Menon: 20290618 - rishikesh.menon@queensu.ca
Zain Zaidi: 20257425 - zain.zaidi@queensu.ca

Table of Contents

Abstract

FlightGear is a simulation of flight that has open-source code. The developers of FlightGear are volunteers with global skills. FlightGear aims to create an advanced, open-source platform for flight simulation that can be used in research or academic institutions, pilot training facilities, or by individuals interested in flight simulation. This report analyzes and discusses the conceptual architecture of FlightGear. Group 19 will discuss the following aspects: the subsystems, use cases and diagrams, control and data flow, evolution of the system, and division of responsibilities. The team also reflects on the lessons learned.

Introduction and Overview

In the evolving technology of flight simulation, Flight Gear stands as a landmark of innovation and community-driven development. This paper takes a deep dive into the complex layers of FlightGear, an early open-source flight simulator that integrates the collective knowledge of volunteers worldwide with its functionality thus suitably serving research, academic use, pilot training, and the hobby of flight simulation enthusiasts. The goal of this exploration is to study the software architecture behind FlightGear by visualizing its subsystems and their data flow, analysing their evolutions and discussing the open-source community that sustains the project.

FlightGear's design is a prime example of the modern software based on the idea of modules and integration of different components, each contributing to the final simulation experience. At the core of FlightGear's functionality are its subsystems: the Input, Simulation Engines (drawn from Flight Dynamics, Visual Simulation, and Weather Simulation), Database (with Aircraft and Scenery models), User Interface, and Networking. This integration results in a replication of the intricacies of the aircraft flight and encounter of variable atmospheric conditions at altitude with beautiful realism.

The Input subsystem is the intermediary between the user and the virtual skies, converting physical actions into things occurring inside the simulation. The Simulation Engines now take control, starting with the Flight Dynamics module performing the intricate dance between the aircraft and the atmosphere, followed by the Visual Simulation engine displaying this interaction onto the user's screen, and with the Weather Simulation module, the whole thing is finalized with dynamic weather conditions.

The core of the FlightGear experience is its Database, which is full of intricate models of aircraft and an abundance of scenic variety that will enable pilots to explore a boundless virtual world. The User Interface however, keeps the interaction with this world user-friendly, so one can use the controls to modify the flight parameters, navigate through menus, and access essential flight information without any difficulty. Networking bridges the gap between a simulator and the network of pilots, unifying people in flying together to create a sense of community and friendship among the users around the world.

The sequence diagrams and use cases in this report depict the complex operations between the different subsystems, which can serve as an entry to the simulation flow. Ranging from the subtle manipulation of an aircraft using input devices to the cooperative, fixed possibilities of multiplayer sessions, these scenarios serve to reveal the true depth and breadth of FlightGear’s capabilities.

The open-source development journey of FlightGear is the embodiment of evolution in response to challenges associated with technological advancements and community aspirations. This evolution entails several pivotal features, the most notable of which is the addition of multi-threading to improve simulations smoothness, the incorporation of more complex rendering techniques to give higher visuals, and the expansion of the global scenery database which makes the experience more and more immersive. The FlightGear's modular architecture has been playing a crucial role in its evolution, allowing independent developments of the subsystems while preserving the full functionality.

The concurrency brought in by multi-threading has enabled FlightGear to take advantage of the multiple cores of modern processors and distribute the computing workload more efficiently. This technical jump has taken the flight dynamics and environmental rendering realism to the next level, which is translated into the best experience for users.

This advancement of FlightGear has been paralleled by an increase in the number of duties of its developers, including a large number of core developers, contributors, community maintainers, and resource providers. This decentralized system of development leverages the collective learning of its community which facilitates the innovation and keeps the simulator up-to-date with the latest flight simulation technology.

In summary, FlightGear is a nearly perfect example of open-source development, a joint project that has achieved unprecedented complexity and versatility. This report focuses on the FlightGear architecture, presenting an in-depth analysis of its subsystems, internal workflow, and development path of its community.

Architectural Style

The ingenious architecture of FlightGear blends object-oriented, event-driven and modular design principles with the mixed networking model that combines client-server and peer-to-peer approaches. With its multi-faceted architecture, the system becomes a reliable, scalable and adaptable flight simulation platform.

Object-oriented design is the spine, making it easy to encapsulate and interact in terms of data and behaviours between objects. It enables the introduction/modification of aircraft models, simulation dynamics and effects with minimal impact on the existing code, enhancing the reusability and extendibility of the program.

Embedded into the framework is a dynamic event-driven architecture that enables FlightGear to be interactive when responding to user inputs and simulation events in real-time. This responsiveness is fundamental to simulate the flight environments and interactions, particularly with the aim of having those changes reflected instantly in the user's experience.

Moreover, modular architecture contributes to the scalability of FlightGear, splitting the simulation into the separate, replaceable parts. This modularity supports community contributions that allow integrating new features easily, such as aircraft models or environmental conditions, without affecting the core simulation.

Concurrency, realized through multi-threading, enables FlightGear to effectively handle and conduct multiple simulation tasks in parallel. This is critical to ensure smooth simulation performance while performing various flight dynamics, weather simulation and user interface updates at the same time.

Lastly, FlightGear uses the combination model of networking that is based on both client-server and peer-to-peer systems. The mixed approach makes multiplayer sessions scalable, data syncing more efficient, and direct user interaction possible, thus giving the user a cohesive and interactive simulation of flying.

Coming together, these architectural styles and strategies provide FlightGear with a robust, open-source flight simulator program, which is both high-performing and adaptable to the changing nature of aviation technology and user preferences.

Subsystems

Input

Input devices play a significant role in allowing users to interact with the simulator by controlling their aircrafts accurately and precisely. Several types of input devices can be utilized by FlightGear. These serve as the link, thus enabling pilots to give commands into the simulation, such as flying their aircrafts up in the skies or, adjusting cockpit controls, or navigating via menu systems/user interfaces.

Simulation engines

Flight dynamics simulation

FlightGear's flight dynamics module is responsible for replicating the complex physics and dynamics behind the aircrafts. It is the most crucial subsystem of FlightGear. The flight dynamics module calculates the aircraft's potential surroundings and their relation to the aircraft. Its goal is to generate realistic flight scenarios and conditions, which include imitation of dangerous situations, safety impacts, etc.

Visual simulation

The visual simulation engine converts the simulation data stream into pictures. This subsystem is responsible for translating simulation data into realistic imagery and creating aircraft, weather and atmospheric effects. The visual simulation engine creates a life-like simulation environment.

Weather simulation

Within FlightGear, weather simulation forms the essence of everything, contributing to the additional realism and complexity through the representation of changing weather patterns encountered by the pilots. These components include clouds, fog, precipitation, turbulence, etc.

Database

Aircraft

Flight simulators use even the shape of an aircraft to help predict the aircraft's aerodynamics. Therefore, FlightGear has a set of aircraft models. Simulation is done by using a carefully generated set of aircraft models. Each model is built to replicate the features and

characteristics of a real-life aircraft. The supported aircraft all interact with the simulation engines by providing input and receiving feedback.

Scenery

There is a scenery database in FlightGear. It constitutes all the virtual land flights that will soon fly over, which gives users the illusion of flying from a real place, over a real place, towards another place. The scenery database contains elevation data for the continents and islands of the world and everything that is arranged atop those elevations: the landmasses, lakes, ponds, airports, cities and towns, etc.

User Interface

Mediators are what FlightGear uses to facilitate the connection between users and the whole simulation. These work as intuitive tools for pilots in control of their planes through display screens and controls that allow them to fly without any disruptions. This section is comprised of various components such as cockpit displays, instrument panels, menus, etc. The user interface within FlightGear allows pilots to adjust flight parameters, browse through menus, or monitor critical flight data with ease.

Networking

The networking subsystem allows pilots to establish connections between each other. This way, the simulation states are synchronized across multiple instances of the application running on different machines; hence, users can actively interact with each other in real-time through this subsystem. This helps create and maintain a community of people who share their interest in flight.

Control and Data Flow

The structure of FlightGear is a sophisticated blend of different subsystems that collaborate to replicate the detailed sensation of flying. Essential to the simulator operation is how control and information flow among these subsystems guaranteeing a lifelike flight experience. This segment explores the complexities of control and information flow within FlightGear explaining how these procedures contribute to the efficiency of the simulator's functions and user involvement.

Control Flow

The foundation of FlightGear's control flow is rooted in its event-driven architecture, enabling interaction between users and the simulation. Devices like joysticks, yokes, pedals and keyboards act as interfaces for user commands. These devices are linked through the Input system, which captures and interprets user inputs into commands, within the environment. This translation process is crucial as it ensures that user actions are accurately mirrored in the simulation with minimal delays, thereby enriching the realism and responsiveness of the flight encounter.

The Flight Dynamics Module (FDM) serves as the nucleus of the simulation engine by receiving processed input data to dynamically calculate how an aircraft responds to flight conditions, control inputs and environmental influences.

The Flight Dynamics Model (FDM) uses models to accurately replicate real-world physics taking into account factors, like lift, drag and thrust. It's crucial for the control flow of this module to constantly update the aircraft status based on input data and environmental conditions in time to ensure the simulation remains faithful.

Data Flow

Flowing of data within FlightGear is quite intricate involving the transfer and conversion of information between subsystems to create a flight simulation experience. Various data sources contribute to this process, including user inputs, preset aircraft models, weather conditions and the global scenery database. As this data progresses through the simulation pipeline, it undergoes transformations that help shape a virtual environment.

From Input to Simulation

User inputs serve as the initial data input that dictates how the simulation operates. Once these inputs are received by the Input subsystem, they are transmitted to the FDM which governs flight dynamics in response. The FDM processes these inputs to adjust the state of the aircraft accordingly. This processed data plays a major role in maintaining accuracy and realism in the simulation by determining how well the aircraft reacts, to commands and external influences.

Environmental and Scenery Information

Alongside processing, the user inputs, the Weather Simulation and Scenery systems provide details for creating the flight environment. The Weather Simulation system dynamically generates data that mirrors real meteorological conditions like wind speed, turbulence, precipitation and cloud formations. At the same time, the Scenery subsystem offers topographical and geographical data to set up a rich background for the simulation. This environmental data is seamlessly integrated into the simulation to influence flight dynamics and visual representation enhancing the user’s experience.

Visualization and Feedback

The Visual Simulation engine takes input from both Flight Dynamics Model (FDM) and environmental systems to transform it into displays. This engine plays a role in rendering the aircraft, scenery and weather elements by converting data into visually appealing images shown to users. The feedback loop created is essential as it enables users to interpret visually how the simulation is progressing, allowing them to adjust their actions based on what they see in terms of environment and aircraft behaviour.

Real-Time Data Sharing

In multiplayer setups, the Networking system handles communication between instances of FlightGear. It ensures that all participants' actions and surroundings are accurately synchronized in time across users, within the simulation scenario.

The management and flow of information, in this scenario are quite intricate calling for communication protocols and coordination methods to ensure consistency and minimize delays in the shared environment.

Impact on System Design

The control and data flow within FlightGear have implications for its system design. The necessity for real-time processing and high levels of interaction necessitates a framework capable of managing intricate data transformations and swift event handling. This architecture supports scalability allowing for the integration of aircraft models, environmental effects and simulation features without compromising performance. Furthermore, the clear delineation of duties, among subsystems promotes development efforts by enabling contributors to concentrate on aspects of the simulator while upholding overall coherence and functionality.

In summary, the management of control and data flow is crucial in FlightGear, playing a role in delivering an engaging flight simulation. By employing data analysis, real-time calculations of dynamics and effective methods, for handling tasks simultaneously FlightGear ensures an authentic aviation experience.

Concurrency

FlightGear is able to manage and execute the multiple processes simultaneously using concurrency which greatly improves the simulation performance and reality. Concurrency is a critical part of a simulation that requires working with intricate elements like airline dynamics, environmental factors, and user inputs. These complex factors have to be processed in a real-time manner so as to ensure an accurate and immersive simulation.

Enhanced responsiveness through multi-threading

FlightGear uses multi-threading which is one form of concurrency through which there are many threads in a single process concurrently performing different tasks. This technique delivers the ability to effectively use a multi-core processor, splitting its workload on several cores to increase the simulation smoothness and its responsiveness. For example, different threads may work on flight dynamics calculations in one task, scene loading in another, weather updates in the third and user interface interactions in the fourth. Thus, these components can be processed simultaneously, which eliminates potential bottlenecks.

Enhanced Realism through Concurrent Processing

Creating an accurate environment for the simulation of flight dynamics and environmental changes requires instantaneous processing. The use of concurrency allows FlightGear to simulate the complex interplay between the aircraft and its surroundings, like weather changes or landscape features, with minimal performance degradation. This parallel processing ensures that pilots encounter true flight behaviour and environmental responses, thus, the educational and training values of the simulator are accomplished.

Networking and Multiplayer Synchronization

Concurrency in multiplayer sessions is a very important factor in synchronizing the states of the various simulations of multiple users. Through managing network communications in a

different thread, FlightGear can be sure that information exchange between the users occurs fast and prevents the unexpected disconnections and maintaining the quality of the shared simulation environment. This particular aspect of concurrency is important for developing a well-integrated and interactive multiplayer game, in which the actions initiated by one pilot are instantaneously applied to the simulations of others.

Challenges and Solutions in Concurrency

Working with concurrency in a complex system like FlightGear is also accompanied by its own set of challenges such as race conditions, deadlocks, and the fact that multi-threaded applications are hard to debug. This is done through design and implementation of such practices as using mutexes and semaphores to manage access to shared resources and using debugging tools specialized for concurrent applications.

Use Cases and Diagrams

Use Case 1: The user turns the aircraft by using the ailerons to roll in the desired direction.

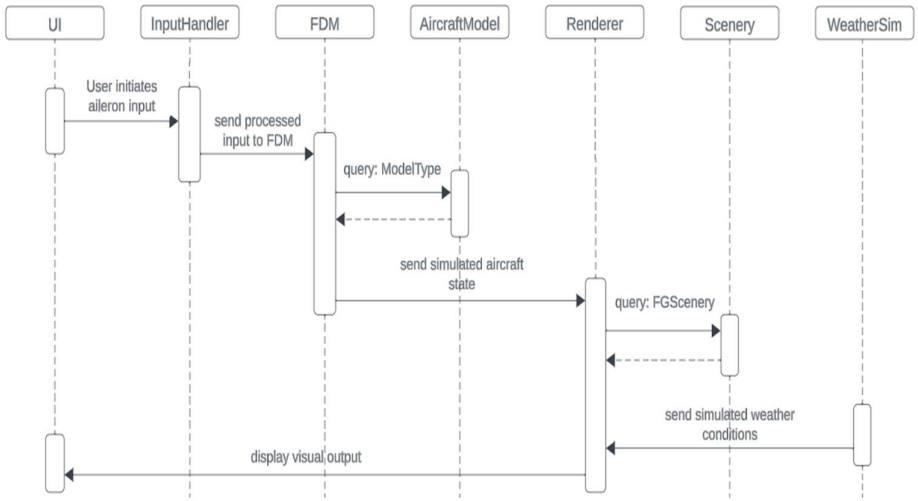
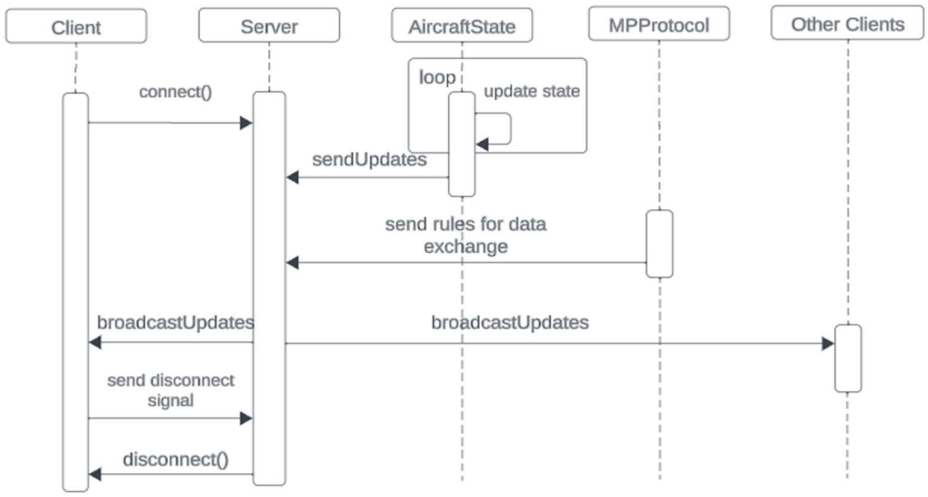


Figure 1: Sequence diagram for use case 1

The first thing that happens is the system receives the input given by the user's input device. This input is handled by the input handler, which processes this input with the input subsystem associated with the specific input device the user is using. The processed input data is then used to update the Flight Dynamics Model (FDM) which will simulate the physical behaviour of the aircraft depending on which aircraft model it is. The renderer then uses the simulated aircraft's state given by the FDM to update the visual simulation of the aircraft. It will also use the scenery and weather simulation to generate the visual output which will then be displayed on the user's screen.

Use Case 2: User connects to a multiplayer server and then disconnects.

Figure 2: Sequence diagram for use case 2



The client connects to a multiplayer server which is a session that is hosted by another user. While on the server, aircraft states of each client are regularly updated in order to ensure synchronization across the simulation. These aircraft states include things like aircraft position, orientation, velocity and control inputs. These updates are then sent to the server, which broadcasts them to all other clients using the defined set of rules for data exchange in the multiplayer protocol. The client then disconnects from the server once a disconnect signal has been sent to the server.

Evolution of the System

The evolution of FlightGear into a successful open-source flight simulator is a shining example of how software development works and how collaborative local processes help it thrive. Similarly, the conceptual architecture of FlightGear has also grown since its inception in order to address the challenges and opportunities that have emerged in the course of the project's history.

The FlightGear simulator was initially conceived as a versatile and simple simulator. In the long run, the fact that the community had bigger goals and plans became evident, and this meant that the need for more scalability was needed. In its early years when the computational power was limited, FlightGear faced the problems of providing realistic visual features and managing dynamic in-flight models in real time. Advancement in computing technology brought about leading to the strategic adoption and incorporation of new technologies by the FlightGear development team to meet the challenges.

FlightGear employed dynamic scenery loading strategies and advanced LOD managements to deal with the high volume of data for the global scenery. This system allows FlightGear to load higher-detail scenery for areas close to the user's current position while simultaneously unloading distant scenery or rendering it at lower detail levels. This technique enables the rendering of only the most relevant data at a particular time, which eventually leads to a reduction in memory usage and a rise in frame rates. The LOD system works in a manner that can adapt to the user's hardware capabilities to guarantee that FlightGear remains available to users having varied computing specifications.

The development team implemented modern OpenGL concepts and shading methods to improve the rendering operations. The GPU was loaded with more graphical processing, thereby allowing to produce highly complex scenes with improved fidelity and better frame rates. These revisions include using latest graphics advancements like real-time shadows, effects, weather effects within the engine while maintaining scalability across hardware platforms.

Being able to integrate with open geographic data sources, such as OpenStreetMap and USGS data for terrain elevation, FlightGear could give users the most realistic and up-to-date global scenery without users having to manually update it. The approach of scalable expansion of the environmental database of the simulator became possible, as data could be added without any trouble and making the process automatic.

At the heart of FlightGear's evolution is its modular design, which allows different subsystems such as the flight dynamics model, the simulation of weather conditions, and the rendering engine, to evolve independently. This modularity is crucial for integrating new aircraft models, each with unique flight characteristics and systems. For instance, the introduction of the JSBSim flight dynamics engine offered a more accurate representation of flight physics, catering to both professional and casual needs by supporting a wide range of fixed-wing and rotary-wing aircrafts.vi

Future development of FlightGear might also be affected by the progress of VR and AR, which gives a more immersive simulation experience. Incorporating these technologies necessitates reassessing the user interaction models and tuning the applications to bear the load of increased data and service demand. Additionally, the advent of AI and machine learning technologies provides chances for more dynamic and intelligent modelling that can be witnessed in adaptive weather systems and AI-controlled air traffic control which are pushing the limits of what is possible in-flight simulation.

Division of Responsibilities

FlightGear developers are mainly divided into two categories: core developers and contributors. The main developers of the project focus on the development and maintenance of internal core functions, and they will be responsible for designing and optimizing the entire software architecture. Contributors include code submitters, community maintainers, resource providers, etc. The users are mainly researchers who use the project as a tool or enthusiasts.

Core Developer/Maintainer

These maintainers are key contributors to the project and typically have deep technical knowledge and a comprehensive understanding of the project's architecture. They are responsible for defining the project's technical roadmap, developing core functionality, and ensuring the technical health of the entire project. This includes, but is not limited to, performance optimization, code quality assurance, integration of new features, and maintenance of existing features. In addition to developing internal core modules and maintaining software architecture. They also need to consider its scalability, robustness and

programmability to provide users with the ability to import their aircraft models and debug them.

Contributor

- Code Submitter

  Submit code to the project for new features or fixes for existing issues. These contributions are usually reviewed and integrated into the main project by core developers.

- Community Maintainer

  FlightGear has an active community forum that guides users on how to use FlightGear and allows users to share resources and experiences on the forum. Community maintainers are responsible for managing forums and social media platforms, ensuring the enthusiasm and health of discussions, and promoting communication and cooperation among users. Community maintainers are often the bridge between project developers and users, able to effectively collect feedback and suggestions from the community, and convey them to the development team, thereby helping the project better meet user needs and guiding the development direction of the project.

- Resource Provider

  Due to the professionalism of FlightGear, also needs many kinds of professional data and resources, including aircraft models, internal displays of aircraft cockpits, professional sound effects, and airport information. These resources are not only implemented through direct contributions but also need to be extended through interfaces provided by the main maintainers, allowing these data to be imported into the software for use.

User

Most of the users play a central role in FlightGear. Its user base mainly includes professionals who use the software for simulation experiments, research or teaching. These activities are crucial to the continuous improvement of the project.

Lessons Learned

The development journey of FlightGear, an open-source flight simulator, offers valuable insights into software project management, architecture, and community collaboration. Key lessons learned from this analysis include:

1. **Community Collaboration is Key:** FlightGear's success demonstrates the value of such an active and involved community. Collaboration and communication that is effective is important because they provide means to channel collective knowledge and efforts of the volunteers globally. Priority for the next projects would be development and running of a community to promote innovation and resolve problems effectively.
2. **Modular Architecture Promotes Flexibility:** The modular design of FlightGear has enabled it to scale and adapt to new features easily by allowing modules to be updated independently and new features to be integrated. In a modular architecture structure, an early investment is a key factor for the long-term project sustainability and growth.

3. **Performance Optimization is Crucial:** The balance between simulation realism and performance requirements is difficult. The development of FlightGear emphasizes the great necessity for focusing on performance optimization from the very beginning, if possible, using techniques like concurrency, which is a tool for improving efficiency with the growing complexity of the realistic simulations.
4. **Stay Adaptable to Technological Advances:** VR and AI integration in FlightGear shows the importance of projects' flexibility and ability to adopt next-gen technologies. This can significantly increase the functionality and the user experience, while at the same time puts a good deal of foresight into the system design.

These lessons from FlightGear's development journey offer a roadmap for managing complex, community-driven software projects, emphasizing the importance of collaboration, architectural foresight, continuous optimization, and technological adaptability.

Conclusion

In conclusion, the conceptual architecture of FlightGear is a stellar example of the creativity and team spirit of its open-source community. This report has examined the depth and complexity of FlightGear's architecture, unveiling a balanced mix of object-oriented, event-driven, and modular design patterns, coupled with an innovative networking model. These architectural decisions lie at the foundation of the simulator's ability to provide a realistic, immersive flight experience that is both high performing and flexible.

The modular architecture guarantees that FlightGear remains flexible and scalable, allowing the inclusion of new aircraft models, environmental effects, and other features without violating the integrity of the core simulation itself. While the object-oriented approach supports the reusability and extensibility, which are necessary for encouraging innovation within the community. The event-driven architecture improves the simulator's reactivity for the dynamic scenario. Concurrency, achieved by multi-threading, guarantees that FlightGear can cope efficiently with the complex, real-time calculations needed for flight dynamics, weather simulation, and user interaction.

Moreover, the hybrid networking model of FlightGear, which uses both client-server and peer-to-peer systems, creates the capacity for scalable multi-player sessions and efficient data synchronization, thus users can enjoy the interactive and shared simulation environments. The complexity of this architecture is not only to satisfy the needs of users ranging from researchers to amateurs, but also to reflect the project's adherence to the principles of open source, which in turn makes the project more open to community improvement.

Ultimately, the conceptual structure of FlightGear shows how the most advanced architectural design can become the basis for creating complicated, community-driven open-source software. It serves as a blueprint for future projects that conveys the prospects of achieving technical excellence via joint efforts and shared knowledge. With the constant evolution of FlightGear, its architecture will constantly adapt to integrate new technology and methodologies, cementing its role as the top platform in flight simulation.

Data Dictionary

1. FlightGear: An open-source flight simulation program developed by volunteers worldwide for research, academic, pilot training, and aviation hobbyist use.
2. Subsystems: Distinct parts of FlightGear, including input, simulation engines, databases, user interface, and networking, each handling specific simulation aspects.
3. Input: Manages user interactions with the simulator through devices like joysticks and keyboards, translating physical actions into simulation commands.

4. Simulation Engines: Key modules for simulating flight physics (Flight Dynamics), visual aspects (Visual Simulation), and weather conditions (Weather Simulation).
    a. Flight Dynamics Simulation (FDM): Calculates aircraft behaviour based on aerodynamics, control inputs, and environment.
    b. Visual Simulation: Renders the simulation's visual elements, including aircraft and environments.
    c. Weather Simulation: Generates dynamic weather effects within the simulation.
5. Database: Stores detailed information on aircraft models and environmental scenery used to create realistic simulation scenarios.
    a. Aircraft Models: Contains detailed configurations for various aircraft.
    b. Scenery: Includes geographical data and visuals for the simulated environment.
6. User Interface (UI): The graphical interface through which users interact with FlightGear, including controls and display panels.
7. Networking: Supports multiplayer functionality, allowing users to connect and interact in shared simulations.
8. Architectural Styles:
    a. Object-Oriented Design: Enhances modularity and reusability through encapsulation and object interaction.
    b. Event-Driven Architecture: Enables real-time response to user inputs and simulation events.
    c. Modular Architecture: Divides the system into independent, updatable parts for scalability and feature integration.
9. Concurrency: The ability to execute multiple simulation processes simultaneously, crucial for realistic and efficient simulation.
10. Networking Model: Combines client-server and peer-to-peer approaches for efficient and scalable multiplayer experiences.
11. Use Cases: Scenarios illustrating user interactions with FlightGear and subsystem integration.
12. Evolution of the System: The development history of FlightGear, highlighting technological and architectural advancements.
13. Division of Responsibilities:
    a. Core Developers/Maintainers: Responsible for core functionality and architecture maintenance.
    b. Contributors: Include individuals submitting code, maintaining the community, and providing resources.

References

1. G. Liu, Q. Huang and J. Han, "Architecture Development of Research Flight Simulator Based on COTS," 2009 International Conference on Information Engineering and Computer Science, Wuhan, China, 2009, pp. 1-4, doi: 10.1109/ICIECS.2009.5364558.
2. V. Garousi et al., "Automated Testing of Simulation Software in the Aviation Industry: An Experience Report," in IEEE Software, vol. 36, no. 4, pp. 63-75, July-Aug. 2019, doi: 10.1109/MS.2018.227110307.
3. Z. Guozhu and H. Zhehao, "Flight Simulator Architecture and Computer System Design and Research," 2020 IEEE 2nd International Conference on Circuits and Systems (ICCS), Chengdu, China, 2020, pp. 35-39, doi: 10.1109/ICCS51219.2020.9336527.
4. *FlightGear is an open-source flight simulator.* FlightGear Flight Simulator. (n.d.). https://www.flightgear.org/legacy/legacy.shtml
5. *FlightGear Wiki*. FlightGear wiki. (n.d.-b). https://wiki.flightgear.org/Main_Page