FlightGear Enhancement Proposal

Prof. Amir Ebrahimi
Queen's University
CISC 322
12 April, 2024

Al-Barr Ajiboye (albarr.ajiboye@queensu.ca)
Kristina Arabov (20kah10@queensu.ca)
Jack Caldarone (21jsc5@queensu.ca)
Safowan Mostaque (20sbm4@queensu.ca)
Grace Odunuga (20goo@queensu.ca)
Jenna Wang (20jw99@queensu.ca)

Abstract

The FlightGear simulation software, renowned for its comprehensive flight dynamics and realistic aviation environments, is poised for an innovative enhancement: the integration of real-time airport flight information. This proposal explores the technical and architectural advancements necessary to incorporate live data on flight schedules, weather conditions, and airport traffic, directly enhancing the realism and educational value of the simulation. The enhancement will allow users to select from real or historically accurate flights at any given airport, supported by actual data on aircraft traffic and meteorological conditions corresponding to their selected date and time.

This enhancement requires some modifications to the existing architecture of FlightGear. The core of these changes involves the integration of an external database that provides up-to-the-minute flight information, necessitating a reevaluation of several subsystems within the FlightGear architecture, including Aircraft Characteristics, Weather, Navigation, and User Requests. To manage these changes, the proposed enhancement will introduce a new interface for data retrieval and synchronization, ensuring seamless communication between the simulation core and the external data sources.

Two distinct implementation methods are considered for incorporating this real-time data feature: connecting to an external, third-party managed database, and developing an internal database directly within the FlightGear system. Each approach is analyzed through the lens of the Software Engineering Institute's Software Architecture Analysis Method (SEI SAAM). This analysis focuses on assessing the impact on non-functional requirements (NFRs) such as performance, scalability, and maintainability, as well as the potential risks associated with each implementation strategy.

Major stakeholders, including end-users and FlightGear developers, are identified, and their primary concerns regarding the enhancement are addressed. For end-users, the enhancement must not impede the software's performance or complicate its usability. For developers, the implementation should facilitate easy integration and future modifications without disrupting existing functionalities.

The abstract concludes with a comparative evaluation of the two proposed methods. It highlights the advantages of using an external database in terms of scalability and minimal impact on the existing system architecture, against the benefits of an internal database that offers control and potentially faster data access but at the risk of increased system complexity and maintenance challenges. This analysis ultimately guides the decision-making process towards an implementation strategy that best aligns with the operational goals and technical constraints of the FlightGear project, setting a precedent for future enhancements and maintaining FlightGear’s position as a leader in flight simulation technology.

Introduction & Overview

The FlightGear flight simulator is embarking on an ambitious path to enhance its simulation environment by integrating real-time airport flight information, which will offer users an unprecedented level of realism and interactivity. This strategic enhancement is designed to provide up-to-the-minute details about flights, weather conditions, and airport traffic at user-selected airports, thereby elevating the educational and experiential value of the simulator. Such improvements necessitate thoughtful consideration of FlightGear's existing architectural framework to accommodate the seamless integration of live data into the simulation experience.

The proposed enhancement revolves around enabling FlightGear to access and display real-time or historically accurate flight information from a comprehensive external database. This capability will transform how users interact with the simulator, offering them the ability to engage with current and past flights, thus mimicking real-world operations and scenarios with greater fidelity. This enhancement is not merely an addition but a significant modification that will affect various subsystems within FlightGear, including but not limited to Aircraft Characteristics, Weather, Navigation, and Time, as well as the overarching Flight Simulation and User Requests subsystems.

To implement this feature, the current architecture of FlightGear will require several adjustments. Key among these is the development or integration of a new interface for interacting with external data sources. This interface must be robust and flexible enough to handle data synchronization between the simulator and the database effectively, ensuring that the flow of data does not disrupt the simulation's performance or user experience. Additionally, the existing modules related to airport operations, aircraft scheduling, and environmental settings will need to be revisited and potentially redesigned to accommodate the new streams of data.

Our enhancement proposal is structured to examine two distinct methods for integrating this real-time data feature. The first method involves establishing a connection to an external database managed by a third party. This approach promises ease of integration and reduced maintenance for the FlightGear team but relies on the availability and reliability of external service providers. The second method proposes the creation of an internal database directly managed within FlightGear's infrastructure, offering greater control over data management and potentially faster response times, albeit at the cost of increased complexity and resource allocation for maintenance and development.

Through the application of the Software Engineering Institute's Software Architecture Analysis Method (SEI SAAM), this report evaluates these methods based on several critical non-functional requirements (NFRs). These requirements include system performance, scalability, maintainability, and reliability. Each requirement is considered in light of its impact on the primary stakeholders: the users and the developers of FlightGear. The analysis aims to identify which method aligns best with the needs of these stakeholders while ensuring the long-term viability and growth of the FlightGear platform.

This introduction sets the stage for a detailed exploration of the proposed architectural changes, the sequence diagram of user interactions with the new features, and a comprehensive risk assessment. By the end of this report, we aim to present a clear and well-supported recommendation for the architectural enhancement strategy that will enable FlightGear to achieve its goals of increased realism and user engagement while maintaining the robustness and flexibility that its community of users and developers value. Through this enhancement, FlightGear is poised to redefine the standards of what flight simulators can achieve, further cementing its place as a leader in aviation simulation technology.

Proposed Enhancement

As explored in our previous reports, FlightGear has a wide variety of functionalities that allow its users to simulate flights with realistic factors in terms of weather, physics, and aerodynamics, as well as manipulate those factors in order to experiment in a sandbox-like environment. After analyzing these functionalities, our group has concluded that an enhancement FlightGear could further benefit from is the addition of real-time airport flight information. In FlightGear, airports are only primarily used as locations the user can land to or depart from, thus it is not possible for a user to see information on flights that are actually scheduled there in real life. Our proposed enhancement would allow users to view what actual flights are scheduled at an airport, the current weather there, and the state of aircraft traffic by using parameters input by the user (i.e. the date and time, and a selected airport). The user could then select a flight path from an airport that reflects current or existing flight information for a set date at a chosen airport. This means that the simulation's realism could be further enhanced by giving users the ability to be part of flights currently taking place or those that have been scheduled in the past at a specific airport. Our group believes this enhancement could be implemented through the use of an external database that is connected to the system.

Derivation Process and Motivation:

The idea for this feature came from examining what FlightGear does not allow the user to do with it. FlightGear allows the user which airports to depart from and fly to, but it is not always clear what flights are currently available at an airport. This presents an issue for FlightGear, it must be as realistic as possible, but the airports do not contain information about current or previous flights. Improving the realism of the simulation became the primary motive of the enhancement, but due to the large amount of data that would have to be stored for this realism to be achieved, a secondary goal became to archive data about real life flights that could be relived in the simulator. Before this enhancement, if a user wanted to simulate a flight that occurred in real life, they would need to set all the necessary parameters themselves, however with the enhancement, this could all be done automatically, greatly improving the ease of use of the software. Since the airports are so frequently visited during simulations, it just makes sense that the user should be able to do more with them other than simply departing and arriving.

Value of Enhancement:

The main goal of the enhancement is to provide a new feature that adds to the realism of the software, while also supplying users with information that helps them learn more about how flights are run in the real world. The users will be able to access a list of past flights that occurred at the airport they are stationed at, and are given the option to use the software to simulate the flight. These flights will be stored in an external database, so that the program can maintain a stable run. If the user decides to simulate one of these flights, they will not need to set any extra parameters, such as selecting weather, aircraft, or other variables if they do not desire to do so. This will increase the ease of use of the feature, however the user will still have the option to change certain settings. This will allow them to follow along the previous flight in order to potentially learn something from the way the pilot flew, but also allow the user to make changes and see how they affect the flight. Many users of FlightGear enjoy the software because of how much they can learn from it, so any feature that can help a user gain a deeper understanding of flights will be greatly welcomed by the user base. Additionally, this feature adds realism, which is a main goal of every simulation software.

Interactions:

It may be possible to implement our proposed enhancement in a way that will not require any major changes to interaction patterns between subsystems. The enhancement would only cause interactions between certain subsystems through the user's own requests regarding airport, date, and time selection. When applied to FlightGear's concrete architecture (see Figure 1), the enhancement would most likely interact with AircraftCharacteristics, Weather, Navigation, Time, and the overall FlightSimulation subsystem. The behaviour of the enhancement (i.e. what information is fetched for a certain airport) would be determined by the UserRequests subsystem. the enhancement would not dramatically affect FlightGear's system other than connecting it to a database that contains flight information of airports for specific dates and times.

AircraftCharacteristics – Our feature would cause an interaction between UserRequests and AircraftCharacteristics to determine what aircrafts are scheduled at an airport for a certain time, based on the information fetched from the enhancement's database determined by the user's selections. The user can select an aircraft from the given output and fly it from that airport. For this to work, the user should also have the aircraft model for the flight installed on their system.

Weather – As a result of the user's selections for flight information at a set airport, the UserRequests subsystem will interact with the Weather/Environment subsystem to establish the weather conditions at an airport for a selected date and time. This subsystem will then establish the behaviour of the Aerodynamics, Images, and Navigation subsystems [1].

Navigation – The components in the Navigation subsystem that are most affected by this enhancement are Airports and Traffic. The Airports component would need to be able to contain or connect to an external database containing flight information related to a specific airport. As well, the traffic would be established by the user's requests and flight information output by the Airports component; it would use real-time data related to the traffic for a given day in order to further enhance the realism of FlightGear. The Navigation subsystem also requires input from Aerodynamics and AircraftCharacteristics to ensure realistic aircraft behaviour [1].

Time – This subsystem sets the time within the flight simulation, thus affecting various subsystems related to it, such as Navigation (i.e. traffic) and Weather (i.e. the environment). The time is set by the user through UserRequests and assists in determining what flights the user can make from an airport at a specific time.

FlightSimulation – One of the main components affected by this enhancement would be AIModel, which establishes how other objects behave in the simulation based on the user's selections (e.g. real-time aircraft traffic and the paths of those flights). Secondly, the Network component would also be affected since the enhancement would most likely require FlightGear to connect to a database that contains the relevant flight information for all airports.

UserRequests – The user will input their selections for their chosen airport, time, and date in order to select a specific flight which would set all the relevant changes to the flight simulation, as discussed above. Additionally, the ATC component of this subsystem would help manage the user's interactions with the simulation's traffic as a result of their selections [1].

Effects on the System:

In order for our proposed feature to be a viable enhancement to FlightGear, it should have a minimal effect on the overall system. The only subsystem that would be dramatically changed would be the Airports component, in which our proposed enhancement would require the source code to add new functionality that fetches data from a database and applies it to the relevant airports within FlightGear. If the proposed feature was implemented, our group hypothesizes that the software's maintainability would not be severely impacted as long as the data for real-life flights from an airport exists in an easily accessible external database. If it is otherwise in an internal database, there would most likely be a negative impact on the maintainability and performance due to the large amount of data that could slow down the execution of the software, or make it difficult for developers to maintain the system. Next, because the enhancement requires a database that frequently updates to include real-life flight information, the system would have excellent informational evolvability due to its accuracy, and source code evolvability since it would not interfere with updates and modifications. The system's testability would also not be impacted as it would be possible to simply interact with the enhancement and test if it gives the correct output, based on current and existing flight information for an airport. However, one possible negative effect on the system would potentially be within the software's security, as the Airports component would become dependent on the flight information database to perform its full functionality and issues may arise if that database is deleted/lost, but this could be avoided by giving the user to enable or disable connections to that database.

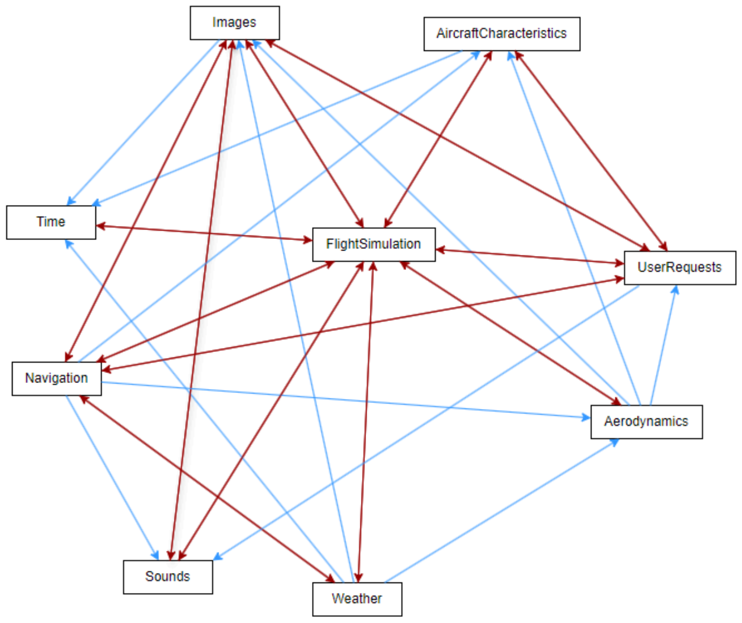

Figure 1. Diagram of the concrete architecture with categorized subsystems, from our previous report.

SEI SAAM Analysis

By understanding the proposed functionality of the enhancement and what it needs to achieve, the next step is figuring out how it may be implemented. Since the enhancement needs to be able to access existing and scheduled flight information for various airports, the best way to integrate this feature would be through the use of a database that can be filtered through by the interface. There are two possible methods for implementing this type of database within FlightGear. The first method would be to connect FlightGear to an external database, most likely managed by a 3rd party. This method would allow the software to focus on expanding its functionality and manageability without having it overshadow the sandbox element of FlightGear. The second method would be to internally implement the database within FlightGear's source code so that the software can more readily access it. This method would allow FlightGear developers to avoid relying on a 3rd party database and focus on the supportability of the software. By utilizing the SEI SAAM methods to analyze the most important quality attributes related to the feature and the degree to which the architecture provides support for the task [2], we can draw a conclusion as to which method is the safest for the enhancement's implementation.

Concerns	Method 1	Method 2
Major stakeholders	**Users** – FlightGear's users are stakeholders regarding this enhancement since the feature can affect FlightGear's ease of use and performance. Most users are simply interested in simulating flights and experimenting with the software, so the enhancement should not impede their enjoyment. **Developers and Contributors** – FlightGear's developers and contributors should be able to easily integrate any updates/modifications to the software's code without breaking the functionality of the enhancement.	**Users** – Similarly to method 1, an internal database could greatly impact the performance of FlightGear, meaning that the users of the program may suffer if the internal database is not implemented correctly. Additionally, this will be the group that gets the most use out of the new feature. **FlightGear Developers** – When implementing the database internally, it is important to not interfere with other parts of the software. If other functionalities are disrupted, it could lead to the breaking of a large amount of personal modifications made by the FlightGear community.
Major NFRs	**Performance** – How well and how quickly the software responds to changes in the simulation caused by the user and internal functions. **Manageability/Supportability** – How easy it is to control the new feature and the software once the enhancement has been implemented using this method. **Integration** – How easy it is to implement the enhancement into the software using the given method. **Reliability** – How successful the enhancement is at meeting the user's requests and interacting with the overall simulation.	**Performance** – How quickly FlightGear will be able to retrieve data from an internal database. **Scalability** – How well FlightGear will be able to manage an extra internal database with potentially thousands of data points. **Maintainability** – How easy it is to import new information about airports and flights into the internal database. **Availability** – How often FlightGear will be able to utilize the enhancement, under multiple different circumstances.
Impact on stakeholders	**Users** – The external database allows users to easily access flight information through FlightGear's interface without worrying about the database impacting the software's performance. Users can also choose whether or not they want to enable a connection to the database to	**Users** – An internal database allows for the users to interact with the new enhancement entirely offline. Due to the database being internal, users could experience some issues with performance if it is implemented incorrectly.

	access real-time flight information. **Developers and Contributors** – This method would make the source code easy to modify and maintain without impacting the flight database.	**FlightGear Developers** – Due to the database being internal, it could have an effect on certain modifications. It would become harder to maintain the source code without affecting the database.
Impact on NFRs	**Performance** – The enhancement would likely not negatively affect the performance of the flight simulation since the database is externally connected to the software, and can be enabled/disabled by the user. **Manageability/Supportability** – An external database allows information to be easily added and managed. FlightGear's developers would not need to add flight information themselves if a 3rd party manages it. **Integration** – Connecting to an external database is easier than creating and managing a database from scratch. **Reliability** – The enhancement interacts with the software as a result of the user's selections and requests. It is expected to successfully filter out information to obtain flights at a selected airport at a specific time, as well as update information on a regular basis. **Potential issue** – Since the enhancement would require an external connection to a 3rd party database, there is a possibility that the enhancement could become unusable if that database is down or deleted at any point.	**Performance** – The database should not negatively affect the performance too much, so long as it has been optimally implemented. Due to it being internal, it may have faster response times than an external database. **Scalability** – As large amounts of data entries are added to the database, there may end up being some issues with performance. An internal database is more likely to interfere with source code than an external one, so adding lots of data will amplify the possibility. **Maintainability** – It should be very simple to add new information into the database, due to the fact that it is all internal. This would require an internet connection, and the developers could simply release an update that contains new information whenever the time comes to add more. This would require users to download the update, which would be unnecessary if the database was external. **Availability** – Due to the database being entirely internal, users will be able to use the enhancement whenever, even if they do not have an internet connection.

Table 1. SEI SAAM analysis of the two implementation methods for the proposed enhancement.

It can be hypothesized that the first method would most likely not have a negative effect on the software. While there is a small possibility that the external database connection could become obsolete, it is possible to handle that issue since it is simply an external connection that can be easily enabled or disabled. On the other hand, an internal database has no issues with internet connections (other than when adding new information to the database), however it would fall short in some areas. For example, there is a chance of interference with the source code that increases when large amounts of data is entered into the database. Based on our analysis, the best implementation method for this enhancement would be to implement it externally. This is due to the scalability concerns with internal databases. In order to reduce the impact on both overall performance and ease of use for developers, it makes more sense to use an external database, which is more suited to deal with scalability requirements than an internal database is.

Architecture

Changes to Architecture:

When considering that the software would need to introduce a connection to an external database that contains real-time flight information at certain airports, FlightGear's concrete architecture (Figure 1) would need to be modified. Using FlightGear's multiplayer functionality as a reference point, our group proposes that another section of the architecture should be composed of a client-server architectural style [3]. The use of this architectural style would allow FlightGear to access servers and databases that are not directly implemented within the software, which is beneficial for its long-term maintainability. Since the database would be connected through a network that the user can enable or disable, the FlightSimulation subsystem would need to establish a new connection to the database (Figure 2). This connection would ensure that all the affected subsystems are given the data relevant for their execution of the enhancement. Aside from this modification, there do not need to be any other dramatic changes to the software as FlightGear would work the same – it will just now have access to an external database that can determine what real-time flights the user can select from an airport at a certain time. As explored previously, there would be new interactions between multiple subsystems within the concrete architecture, so the impacted subsystems and their relevant directories, in parentheses, would be:

- AircraftCharacteristics (Aircraft)
- Weather (Environment)
- Navigation (Airports, Traffic)
- Time (Time)
- FlightSimulation (AIModel, Network)
- UserRequests (ATC)

Alternatives to Implementation:

The proposal explores two primary methods of implementation: connecting to an external, third-party managed database or constructing an internal database within FlightGear itself. Each approach presents its unique set of challenges and advantages, requiring a thorough examination of their influence on the architecture and overall system performance.

Opting for an external database introduces FlightGear to a client-server architecture, where FlightGear functions as a client retrieving data from a server managed by a third-party provider. This new model allows users to access the most current flight information without hefty data storage demands on the system. By leveraging the capabilities of an established data provider, this approach significantly reduces the necessity for extensive modifications to FlightGear's existing architecture.One of the advantages of this method is its efficiency in delivering up-to-date information, therefore preserving the simulation's performance and not overburdening system resources. However, reliance on an external database is not without risks. One major concern is the dependency on third-party data availability and reliability. Any disruptions in the data source, whether due to technical issues or service discontinuations, could significantly impact the functionality of this enhancement. Also, data privacy and security emerge as critical considerations, as sharing information across networks could expose users to potential vulnerabilities.

Creating an internal database within FlightGear shifts the architecture towards a layered model that separates data management functions from the core operational layers of the simulation. This architectural change boosts modularity and allows targeted optimization of the data layer, enhancing both performance and security. A dedicated layer for real-time flight information streamlines data retrieval, ensuring smooth integration with the simulation and customization according to user preferences. This strategy not only speeds up data access but also enriches the user experience by giving FlightGear precise data management control. On the contrary, an internal database poses several challenges for FlightGear, impacting both its architecture and operational effectiveness. It significantly increases the system's architecture complexity, requiring extensive development work that could impact performance and complicate future maintenance and scalability. Also, the internal management of such a database demands constant efforts in data updates and security, potentially diverting resources from other crucial development tasks.

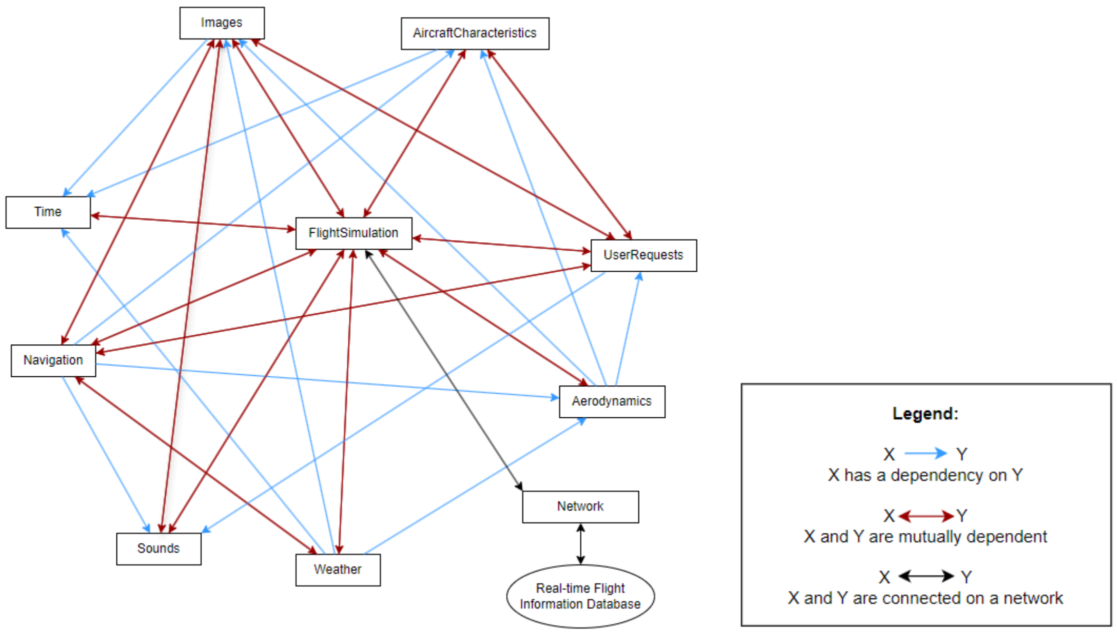

Figure 2. Updated diagram of FlightGear's concrete architecture using the Object-Oriented style, with the addition of a Client-Server style segment that handles the connection to the real-time flight database for the enhancement.

Use Cases

Case 1: Viewing flight information

Assume a user is already running a flight simulation, and they choose to view information about current flights at a nearby airport. They will make this request through the user interface, which will submit their request to the user requests subsystem. The user request subsystem will process this, and then communicate with the main flight simulation subsystem, which is where the software will communicate with the network. If it can connect to the network, it will retrieve information from the database and send it over the network to the flight simulator, which will display it to the user interface. If it cannot connect to the network, it will display a network error instead.

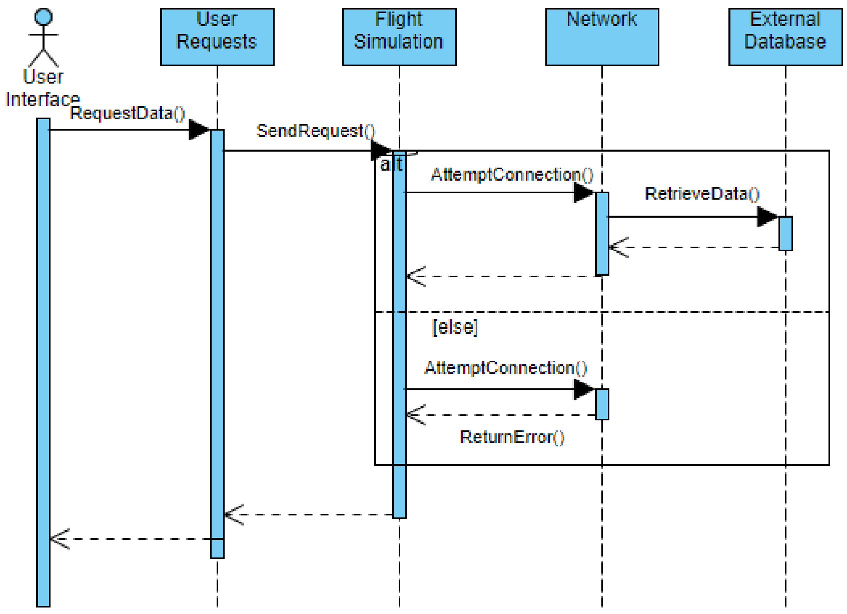

Figure 3. Sequence diagram of the first use case.

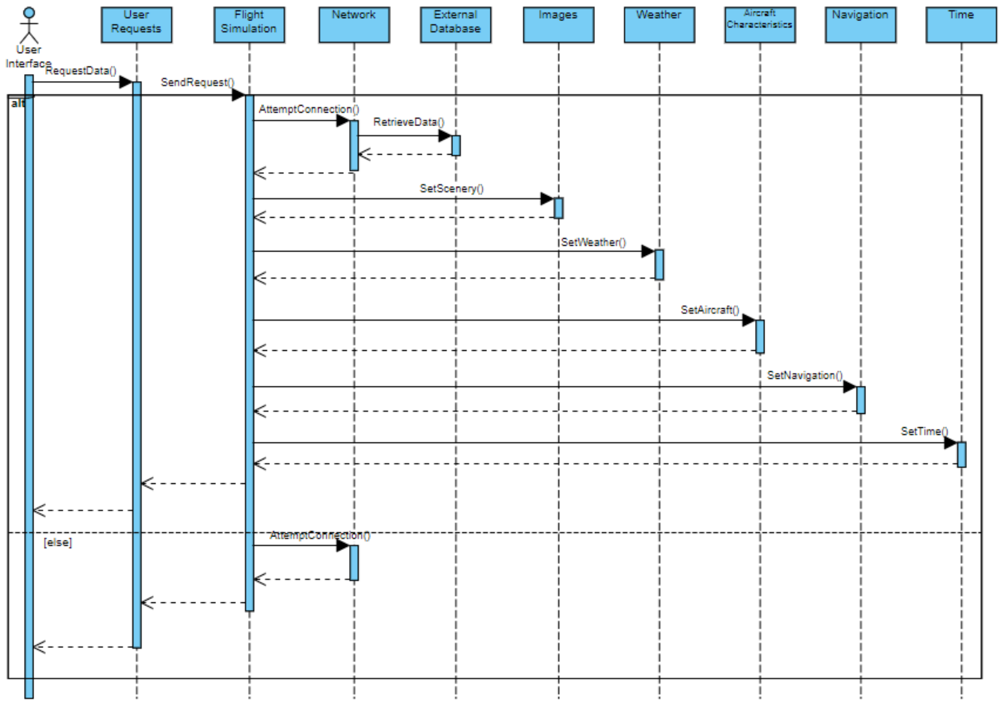

Figure 4. Sequence diagram of the second use case.

Case 2: User selects a previous flight from an airport menu after viewing options

Assume a user is already running a flight simulation, and they have already retrieved a list of options from the database. They will select which flight they want to simulate through the user interface, which sends the request to the user requests subsystem. Similarly to the last use case, the request is sent to the flight simulation system, which attempts to connect with the network. If successful, it retrieves the required information, otherwise an error is displayed and the case ends early. After retrieving the required information, the flight simulation system will communicate the new data with many other subsystems in order to update the environment to fit with the selected flight. The subsystems it will communicate with are: images, time, weather, navigation, and aircraft characteristics. After this, all the changes will be reflected in the user interface.

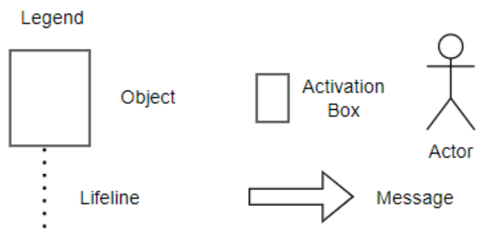

Figure 5. Legend for the sequence diagrams.

Data Dictionary
Availability. The readiness and accessibility of a system or service to users at any given time.
Contributors. Anyone who works on the development of a project or any add-on of a project who is not a member of the team.
Database. An organized collection of structured information, or data, typically stored electronically in a computer system.
Developers. A member of the team who works on a project's source code.
Integration. The process of making one software application compatible with another.
Maintainability. The ease of which a system can remain well-supported as time goes on.
Modularity. A logical partitioning of components that makes otherwise complex software manageable in terms of implementation and maintenance.
Performance. How efficiently a software can accomplish its tasks.
Reliability. The probability of failure-free operation of a computer program.
Scalability. The capability to increase or decrease system performance and cost in response to changes in demand.
Stakeholders. The people or groups affected by a software development project.
Supportability. How easy it is to support a new feature or software.
Synchronization. The coordination of multiple processes in a multi-process system.
Users. An individual that interacts with a system, program, or product.

Naming Conventions
NFR. NFR stands for "non-functional requirements."
SEI SAAM. SEI SAAM stands for "Software Engineering Institute's Software Architecture Analysis Method."

Lessons Learned

Our journey with the FlightGear Enhancement Proposal was a profound learning experience. Working together, we not only gained a deeper understanding of software architecture and its complexities but also discovered the value of teamwork, research methodologies, and diverse perspectives in solving complex problems.

One of the first lessons we learned was the significance of diverse viewpoints and collaborative effort. Each member brought unique insights to the table, especially when it came to deciding on the architectural style that best suited our proposal. Where one member saw the benefits of a client-server model for accessing real-time airport data, another advocated for the control and speed an internal database could offer. Through healthy debate and constructive feedback, we found a balanced approach that brought together the strengths of both ideas.

Exploring FlightGear's architecture, we were amazed by the complexity and depth of its software design. On the surface, it looks like simple simulation software, but it's actually built on layers of intricate subsystems. Figuring out how adding real-time airport information would affect different parts of FlightGear was both tough and eye-opening. We had to think deeply about data flow, system performance, and user interaction. This y showed us how careful planning and foresight are essential in software development, and how even small changes can significantly impact the entire system.

One of the biggest lessons we learned is that learning is an ongoing process. Our views on what makes an effective enhancement for FlightGear changed dramatically as the project progressed. Initially, we were all about adding new features for innovation's sake, but we eventually focused more on user experience, system reliability, and maintainability. This change came from our deeper look into the architecture, user scenarios, and stakeholder analysis.

Conclusion

In conclusion, the enhancement proposal for FlightGear aims to significantly augment the realism and educational value of the flight simulator by introducing real-time airport flight information. This feature leverages an external database to provide users with accurate, up-to-the-minute details on flights, weather conditions, and air traffic at selected airports. Through comprehensive analysis, including the derivation of the proposal, assessment of its value, interactions within the system, and architectural considerations, we have determined that integrating this enhancement will offer a richer, more immersive simulation experience without compromising the system's performance or maintainability.

Our examination through the SEI SAAM analysis highlighted the advantages of connecting FlightGear to an external database managed by a third party. This approach not only simplifies the implementation process but also ensures that FlightGear remains a dynamic, evolving platform for flight simulation enthusiasts. The choice to rely on an external database minimizes the impact on the system's architecture and avoids the potential pitfalls associated with data management and security inherent in an internal database solution.

By embracing this enhancement, FlightGear will stand at the forefront of flight simulation technology, offering unparalleled realism that extends beyond the cockpit. Users will gain insights into the operational aspects of airports and flights, enhancing their understanding of aviation in a global context. This proposal, therefore, not only aligns with FlightGear's mission to deliver a comprehensive flight simulation experience but also sets a new standard for what can be achieved in the realm of virtual aviation.

The collaborative effort of our team, guided by Prof. Amir Ebrahimi at Queen's University, underscores our commitment to advancing FlightGear's capabilities. We believe that by implementing this proposal, FlightGear will continue to inspire, educate, and captivate users around the world, further cementing its position as a leading flight simulator. As we move forward, we are excited about the potential of this enhancement to enrich the FlightGear community and contribute to the broader field of aviation simulation. We extend our gratitude to all stakeholders involved in this proposal, from the conceptualization to the detailed analysis, for their invaluable contributions and insights. Together, we are not only enhancing a flight simulator but also fostering a deeper appreciation and understanding of the intricate world of aviation.

Works Cited

[1] *FlightGear Source Code.* Github. https://github.com/FlightGear/flightgear/tree/next/src.

[2] R. Kazman, L. Bass, G. Abowd and M. Webb. (1994). SAAM: a method for analyzing the properties of software architectures. *Proceedings of 16th International Conference on Software Engineering*, pp. 81-90. https://ieeexplore.ieee.org/document/296768.

[3] Ebrahimi, A. (2024, Jan.). *Week 3's lecture slides* [Lecture notes]. Queen's University. onq.queensu.ca/d2l/le/content/861602/viewContent/5155034/View.