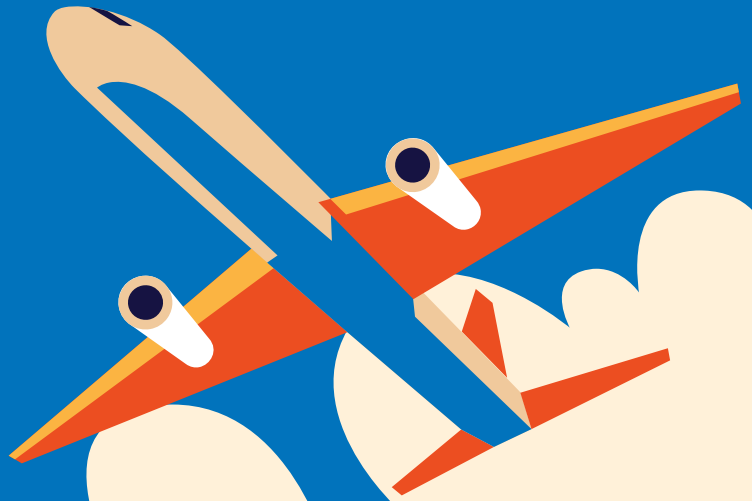
Group 19: Cloudy with a Chance of Pilots

Assignment 2: Concrete Architecture

Video Link



Report Assignments

Rishikesh Ramesh Menon (Leader): Discrepancies Between Conceptual and Concrete Arch., Lessons Learned

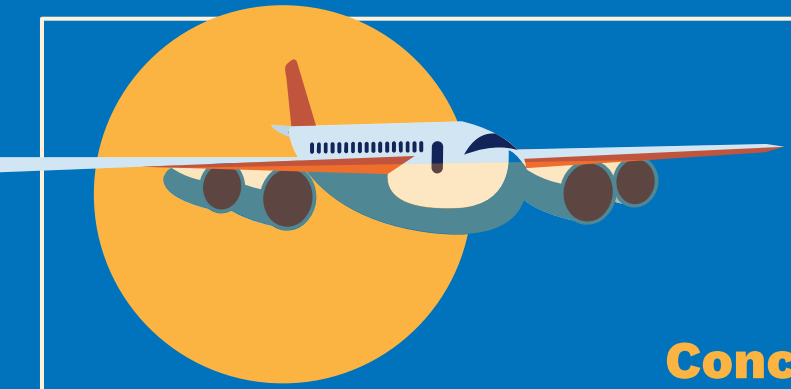
Zain Imam Zaidi (Presenter): Descriptions of Top-Level Subsystems

Manraj Singh Juneja (Presenter): Descriptions of Top-Level Subsystems

William Ban: Use Cases and Diagrams

Sara Jaffer: Analysis of Autopilot Subsystem

Vasuki Elamaldeniya: Derivation Process



Concrete Architecture

FlightGear

By Group 19 for CISC 322/CMPE 326

Agenda

Introduction

- This paper conducts an architectural analysis of FlightGear, comparing the initial conceptual design with the actual concrete architecture.
- The focus is on identifying and understanding the discrepancies between the envisioned design and the actual implementation.
- This analysis is essential for developers and project managers, offering insights into the software's evolution, strengths, and areas for improvement.

- We conducted an in-depth examination of the source code and documentation, utilizing tools like SciTools Understand for visualization.
- This included generating dependency graphs and mapping code to specific subsystems, providing insight into the core modules' relationships and dependencies.
- Initially, we explored the high-level architecture, focusing on modular implementation, concurrency, and event-driven responsiveness.
- We also examined the Autopilot subsystem, finding more complex interactions and dependencies than expected, underscoring the challenges of real-time control and data processing.
- Our analysis led to the discovery of new subsystems not outlined in the conceptual framework which evaluates the effectiveness of FlightGear's object-oriented, event-driven, and modular design principles as developed in the concrete architecture.

Derivation Process

- The process started with an engagement with FlightGear's conceptual architecture to understand the system's design.
- The team explored the source code and documentation to grasp the structure and relationships among core modules.
- Visualization of source code was achieved using SciTools Understand, generating a dependency graph of the system's architecture.
- This visualization aided in comprehending module dependencies and interactions.
- Source code was mapped to subsystems, identifying the architectural building

- Files were categorized based on functionality and system relevance, and dependency graphs highlighted module interdependencies.
- The conceptual architecture was compared with the concrete architecture, identifying differences and analyzing their rationale.
- After thorough analysis, the concrete architecture of FlightGear was finalized and comprehensively documented, including diagrams and explanations of subsystem interactions.

Concrete Subsystems

- FlightGear is an open-source flight simulator known for its complexity, versatility, and collaborative development effort.
- This section compares FlightGear's architecture as analyzed in-depth with the conceptual architecture from Assignment 1.
- The goal is to reveal FlightGear's design evolution, highlight new subsystems, and assess their contributions to the simulator's capabilities.
- The methodology involved leveraging Understand for static code analysis, dependency graph examination, and metrics evaluation.

- This approach provided a detailed view of FlightGear's codebase, identifying core and peripheral subsystems and their interdependencies.
- Both the conceptual and concrete architectures underscore FlightGear's commitment to an object-oriented, event-driven, and modular design.
- These designs meet the software's needs for flexibility, scalability, and maintainability.
- The concrete analysis showcases how these design principles are applied, demonstrating the architecture's adaptability to new technologies and user expectations.

Core Subsystems: Input and Simulation Engines

- The conceptual architecture accurately anticipated the subsystems foundational to FlightGear's operation.
- The concrete analysis reveals intricate internal mechanisms and sophisticated interactions within these subsystems.
- The Flight Dynamics Simulation incorporates advanced aerodynamic models and integrates with the Visual Simulation for realistic flight experiences.
- The Weather Simulation subsystem showcases the event-driven architecture by adapting weather conditions in real-time based on the simulation context.

Core Subsystems: Database

- The database's role in storing and managing aircraft and scenery models was well-conceived in the conceptual architecture.
- The concrete architecture reveals the database subsystem's dynamic nature, including mechanisms for updates and expansion.
- This reflects new data sources and user contributions, reinforcing FlightGear's open-source ethos.

Core Subsystems: User Interface

- Initially, the UI was conceptualized as a mediator between the user and the virtual flight environment.
- The concrete architecture reveals a more intricate and user-centric design for the UI in FlightGear.
- It's not just about navigation and control; the UI includes a dynamic interaction system that adapts to simulation context and user preferences.
- Advanced visualization tools, customizable control panels, and real-time feedback mechanisms are integrated to enhance user engagement.
- This subsystem underscores FlightGear's focus on usability and accessibility, catering to users of varying expertise.

Core Subsystems: Networking

- Beyond enabling multiplayer functionalities, the concrete architecture reveals a robust networking subsystem.
- This subsystem supports live data exchange for weather updates, synchronized world events, and collaborative community engagement in FlightGear.
- It employs advanced protocols and real-time data management techniques for consistency and performance in distributed systems.
- This marks an expansion from the conceptual framework, showcasing advanced networked operations integration, vital for an immersive shared simulation space.

Newly Identified Subsystems

- Newly identified subsystems include Plugin Architecture, Data Management and Caching, and Real-Time System Monitoring.
- These enhancements support FlightGear's extensibility, efficiency, and usability.
- The additions underscore the development team's responsiveness to technological advances and community feedback.
- This facilitates continuous improvement and adaptation of the simulator.
- Sheds light on the software’s evolutionary trajectory showing a commitment to performance, and user-centric design than we initially conceptualised

Plugin Architecture

- The subsystem introduces a layer of extensibility not anticipated in the conceptual architecture.
- It allows third-party developers to create and integrate plugins, enabling the incorporation of new functionalities without altering the core codebase.
- This architecture enhances FlightGear's modular design and fosters community engagement and innovation.

Data Management and Caching

- Essential for managing extensive data, this subsystem employs sophisticated caching strategies to optimize performance.
- It ensures high-fidelity terrain and weather data are readily available, minimizing latency and enhancing user experience.
- This exemplifies the concrete architecture's focus on efficiency and responsiveness.

Real-Time System Monitoring

- Aimed at monitoring and optimizing simulation performance, this subsystem adopts a proactive approach to maintain high performance and stability.
- It tracks system metrics like frame rates and resource utilization, allowing FlightGear to dynamically adjust simulation parameters for a seamless user experience.
- This reflects an underlying emphasis on user satisfaction and software reliability in the concrete architecture.

- The exploration of FlightGear's concrete architecture confirms foundational principles and reveals significant evolutions and enhancements.
- Additional subsystems enrich functionality and exemplify FlightGear's dynamic and adaptive development.
- The analysis highlights the importance of flexibility, community engagement, and a forward-looking approach in managing open-source projects.
- As FlightGear evolves, its architecture stands as a testament to collaborative innovation and strategic planning in simulation software development.

Analysis of Autopilot Subsystem

Overview

- **Purpose**: To guide aircraft autonomously based on user settings.
- **User Interaction**: Initialization through cockpit instruments or GUI, setting properties like heading, speed, altitude, and waypoints.

Conceptual Architecture

- **Architecture**: High-level abstraction focusing on dependency and functionality.
- **Key Components**: Autopilot Manager and Autopilot Components Manager
- **Event-Driven Approach**: Sensor changes and user inputs as event producers, triggering the Autopilot and Components Manager for adjustments.

Concrete Architecture

- **Hierarchical Property Tree**: Organizes information about the aircraft and environment, facilitating data access and manipulation.
- **Component Subdivision**: Base, Analog, and Digital Components for specialized functionalities.
- **Control Mechanisms**: Implementation of PID and PI controllers for error correction based on sensor readings.

Event-Driven Architecture and Real-Time Response

- **Architecture Consistency**: Event-driven approach as conceptualized.
- **Dynamic Response**: Real-time reaction to sensor data and user inputs using PID controllers and component-specific actions.
- **Realism and Reliability**: Ensures realistic and reliable flight simulation through accurate and responsive control mechanisms.

Reflexion Analysis

High-Level Architecture
Modular and Object-Oriented Implementation

- Conceptual Architecture: Independent modules for subsystems like Flight Dynamics, Visual Simulation, Weather Simulation.
- Concrete Architecture: Subsystems more interlinked than anticipated; example - Weather impacting Visual Simulation.
- Rationale: Flight simulation's realism needs led to tight interdependencies, not fully anticipated in the original design.

High-Level Architecture
Event-Driven and Real-Time Responsiveness

- Conceptual Architecture: Efficient real-time event handling and seamless subsystem integration.
- Concrete Architecture: Performance challenges in complex simulations, leading to reduced responsiveness.
- Rationale for Divergence: Probable underestimation of the complexity of concurrent process management.

High-Level Architecture Concurrency and Performance

- Conceptual Architecture: Distributed computational demands for real-time simulation tasks.
- Concrete Architecture: Performance bottlenecks in high complexity scenarios.
- Rationale for Divergence: Original concurrency design may not fully address real-time processing intricacies.

Autopilot Subsystem Architecture
Component Interaction and Dependencies

- Conceptual Architecture: Planned reliance on external inputs like sensors, UI, and Navaids Subsystem for dynamic adaptation.
- Concrete Architecture: More pronounced dependencies in practice; complex web in property tree structure necessitating real-time adjustments.
- Rationale for Divergence: Underrepresentation of the complexities in real-time data processing; interactions for flight accuracy more intricate than envisioned.

Autopilot Subsystem Architecture
Autopilot Components and Management

- Conceptual Architecture: Straightforward division between Autopilot Manager and Components Manager for managing flight properties.
- Concrete Architecture: Subdivision into Base, Analog, and Digital Components for handling specific functions with precision.
- Rationale for Divergence: Complexity in managing autopilot functions demanded more detailed categorization for accurate flight property management.

Autopilot Subsystem Architecture
Event-Driven Architecture

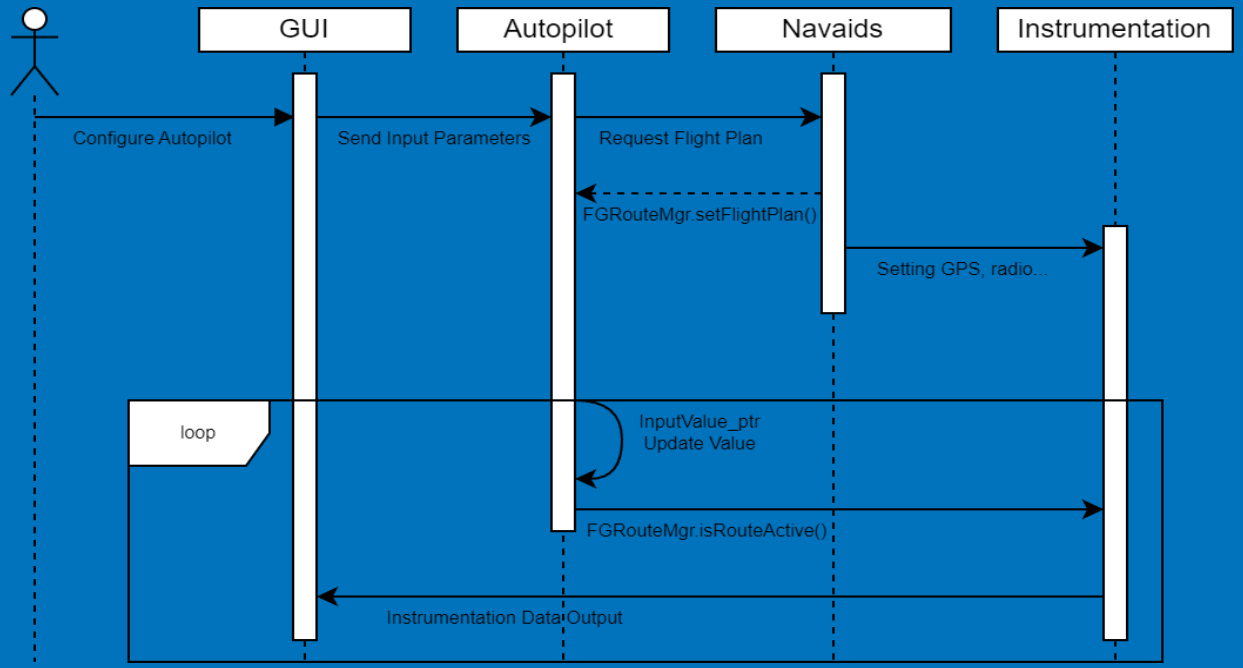
- Conceptual Architecture: System adjustments triggered by aircraft state or environmental changes; focus on immediate response.
- Concrete Architecture: Enhanced with sophisticated control mechanisms like PID and PI controllers for higher precision.
- Rationale for Divergence: Implementation of advanced control algorithms indicates an adaptation to the nuanced real-time flight dynamics.

Autopilot Subsystem Architecture
Real-Time Responsiveness and Control

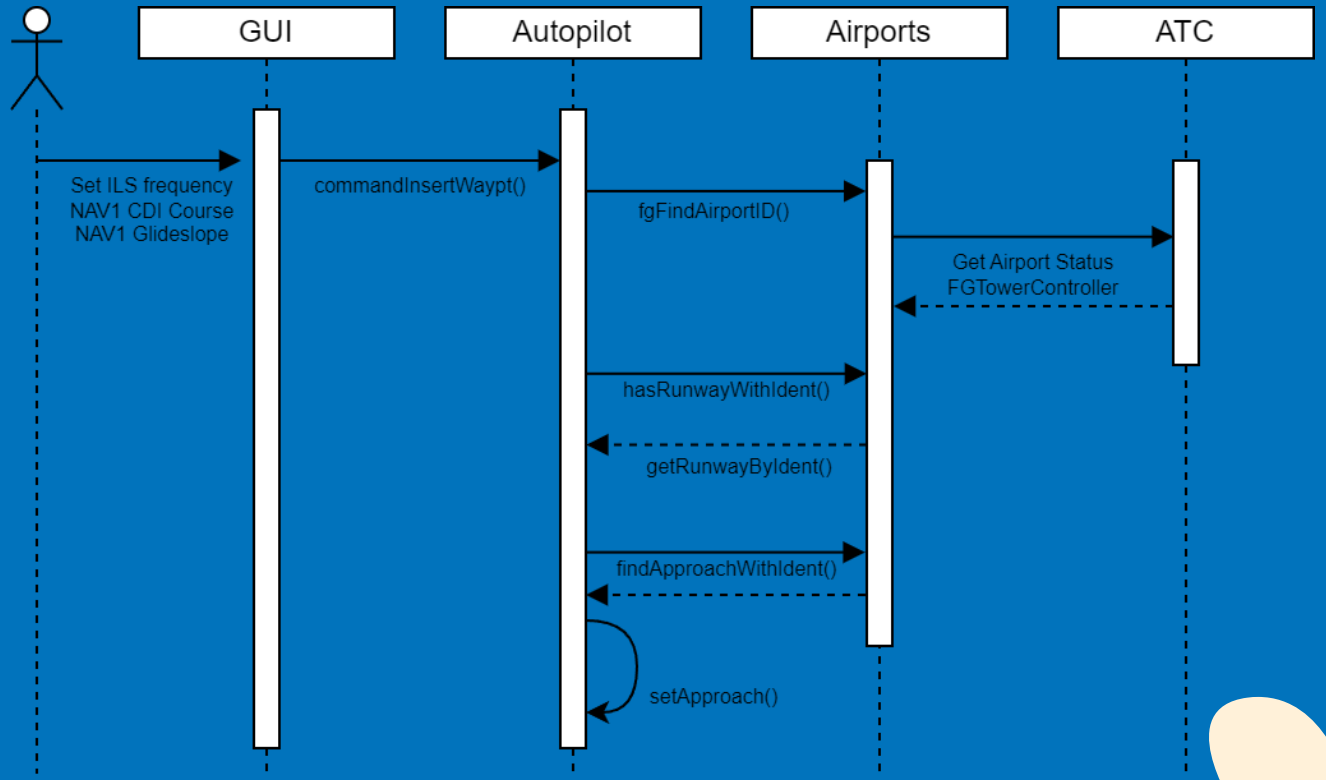
- Conceptual Architecture: System anticipated to react instantly to changes, maintaining stability and user settings.
- Concrete Architecture: Implementation of a PID controller for real-time adjustments, providing precise control over flight dynamics.
- Rationale for Divergence: Concept identified the need for responsiveness, but practical implementation demanded a more complex control mechanism, reflecting advanced understanding of dynamic flight environment control needs.

Use Cases

Use Case 1: Autopilot Configuration



Use Case 2: Choose an airport for landing in midway



Conclusion

- Our investigation underscores FlightGear's evolution from concept to implementation, highlighting community, technological advancements, and commitment to excellence.
- Explored core and new subsystems, showcasing FlightGear's adaptability and enhanced user experience.
- The comparison of conceptual and concrete architectures revealed significant design and functionality developments.
- Detailed use cases illustrate FlightGear's complex operational dynamics.
- FlightGear's architecture exemplifies collaborative innovation and strategic planning in open-source development.
- Offers valuable insights for future enhancements, affirming the importance of flexibility, community engagement, and a forward-looking development approach.

Thank you!