FlightGear Concrete Architecture

CISC 322 W24 – Group: Chicken AI

https://youtu.be/l_67OSRaEiw

Team Members

Team Lead: Derek Youngman

- High-Level Conceptual and Concrete Architecture

Group Presenter: Marion Anglin

- Presentation, Conclusion, Lessons Learned

Group Presenter: Shrinidhi Thatahngudi Sampath Krishnan

- Presentation, Second Level Conceptual and Concrete Architecture

Team Member: Akash Singh

- High-Level Concrete Architecture

Team Member: Abbey Cameron

- Introduction, Overview, Abstract, Derivation Process

Team Member: Ximing Yu

- Use Cases

Our Derivation Process

- Using Understand software to analyze FlightGear’s source code
- Compare our conceptual architecture to dependencies in FlightGear’s source code
- Grouping source code components into subsystems matching our conceptual architecture and FlightGear’s dependencies
- Constructing our concrete architecture that matches FlightGear’s dependencies

FlightGear's Architecture & Reflexion Analysis

Figure 1: Updated Conceptual Architecture

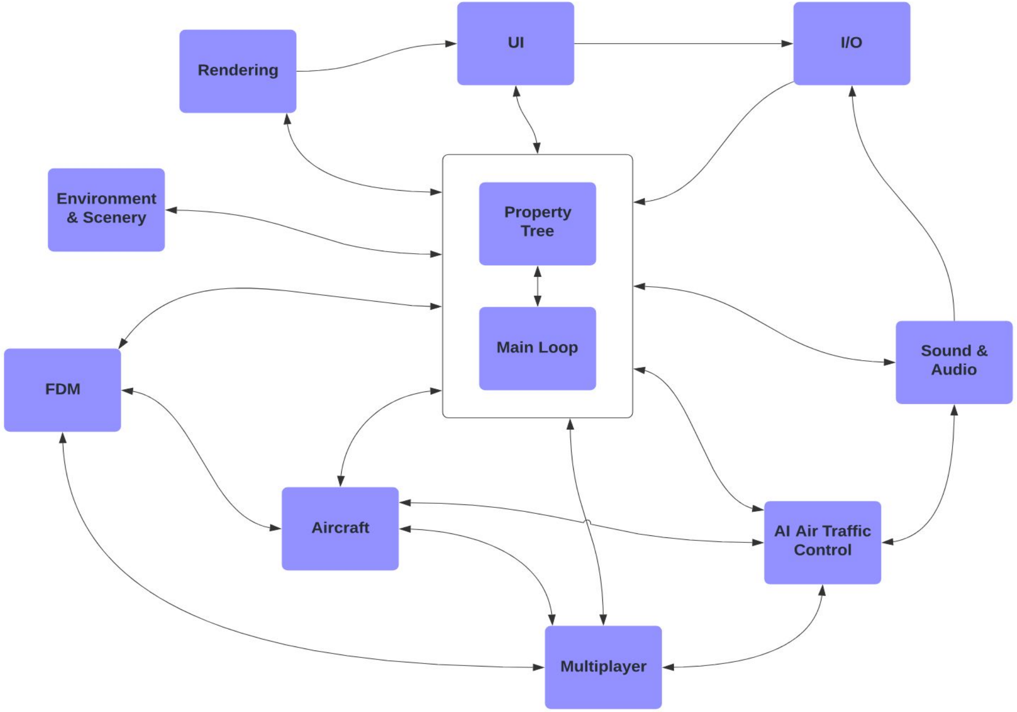


Figure 2: Our Concrete Architecture

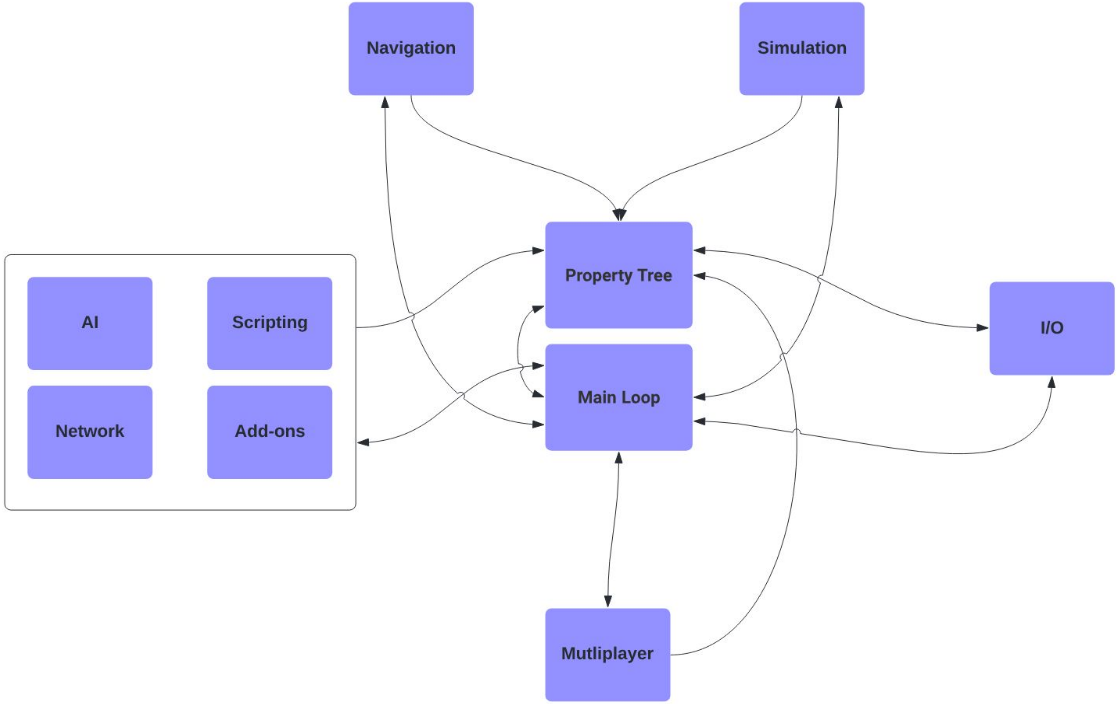
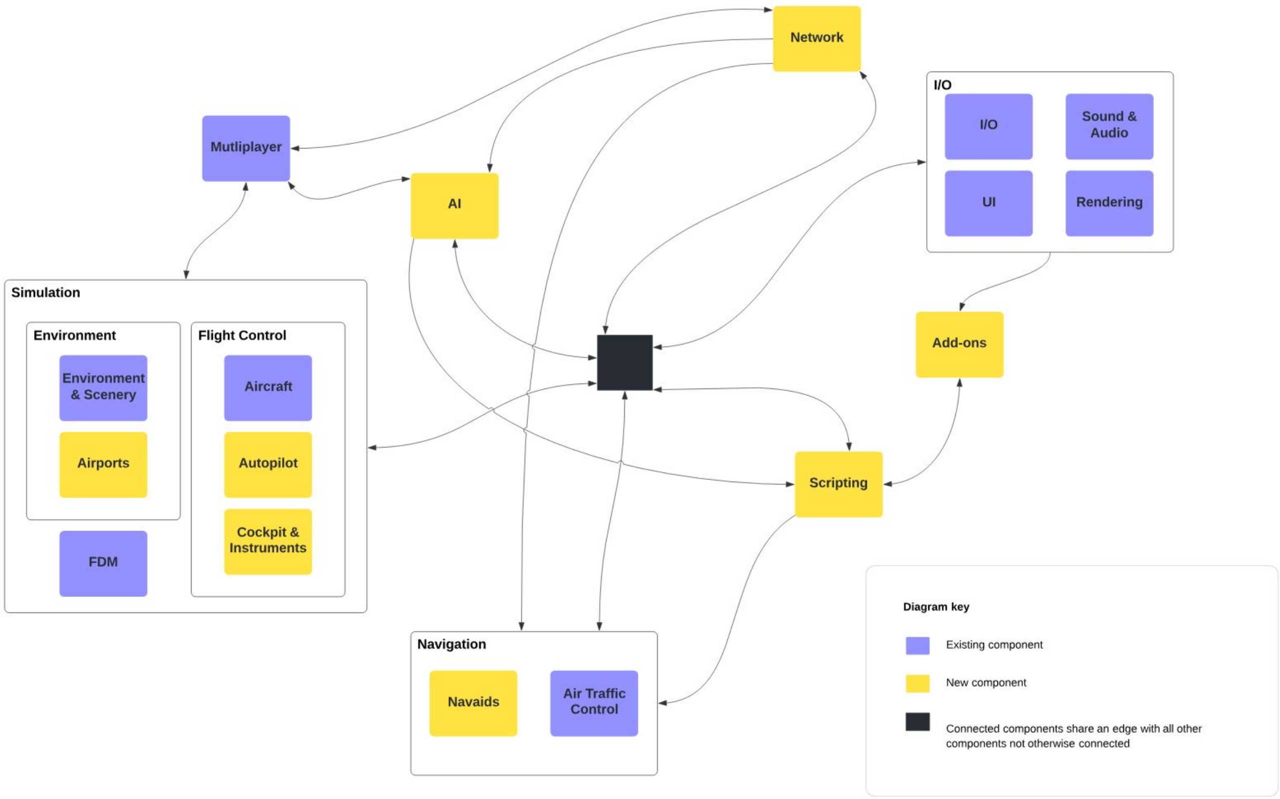


Figure 3: Our Concrete Architecture



Network

Mutliplayer

AI

I/O

I/O

Sound & Audio

UI

Rendering

Add-ons

Simulation

Environment

Environment & Scenery

Airports

Flight Control

Aircraft

Autopilot

Cockpit & Instruments

FDM

Scripting

Navigation

Navaids

Air Traffic Control

Diagram key

🟪 Existing component

🟨 New component

⬛ Connected components share an edge with all other components not otherwise connected

Existing Components in Concrete Architecture

Property Tree

- Every component depends on Property Tree
- Bidirectional dependency with Input & Output
- Doesn’t depend on many components

Multiplayer

- Shares dependencies with Network & AI model
- Lacks dependencies with I/O & ATC

Navigation (ATC + NavAids)

- Lacks dependencies with Multiplayer

Input & Output (I/O + GUI + Rendering + Sound/Audio)

- Depends on other components to output info
- Bidirectional dependency with Property Tree
- Lacks dependency with Multiplayer
- Shares dependency with Simulation component

Simulation (FDM + Environment + Aircraft)

- Bidirectional dependency with Input & Output

New Components in Concrete Architecture

Main Loop

- Initializes & controls the system
- fgMainLoop() invokes each process, with aid from src/Main/globals.cxx that helps keep track of FG's global subsystems

AI Model (ATC + other AI methods)

- FG has implemented additional AI methods
- Different AI models (ATC + others) jointly managed within AI Model component

Network

- Standalone component that serves a bigger role than multiplayer functionality
- Shares dependencies with Multiplayer, Simulation, AI Model components

Scripting

- Scripting component "Nasal" supports reading & writing of internal FG properties
- Bidirectional dependency with Add-Ons
- Depends on Navigation component

Add-Ons

- Implements and imports additional .xml libraries to FlightGear, allows users to expand on FlightGear by adding their own modules or dialogue
- Depends on Scripting components, Input & Output depends on Add-Ons

FlightGear’s Architectural Style

Repository Style

- Facilitated by Property Tree, which acts as a database storing flight simulation data
- Every component depends on Property Tree by reading data from it (eg. fgGetNode()), but Property Tree mostly does not depend on other components
- Components are not decoupled from each other

Client-Server Style (Multiplayer Component)

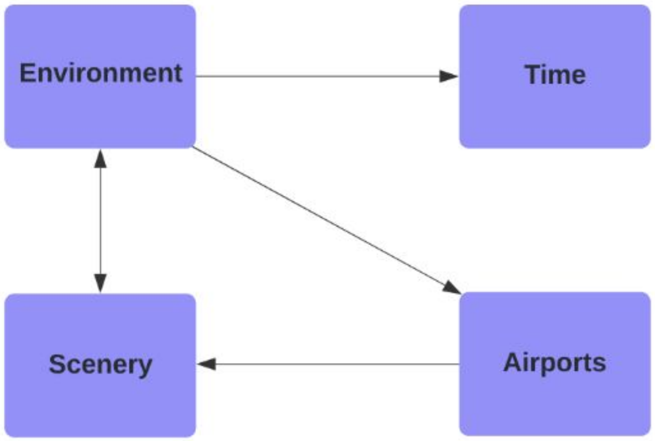
- Primarily present in FlightGear’s Multiplayer environment
- Client can connect to FlightGear’s Property Tree to receive flight simulation data

Object-Oriented Style

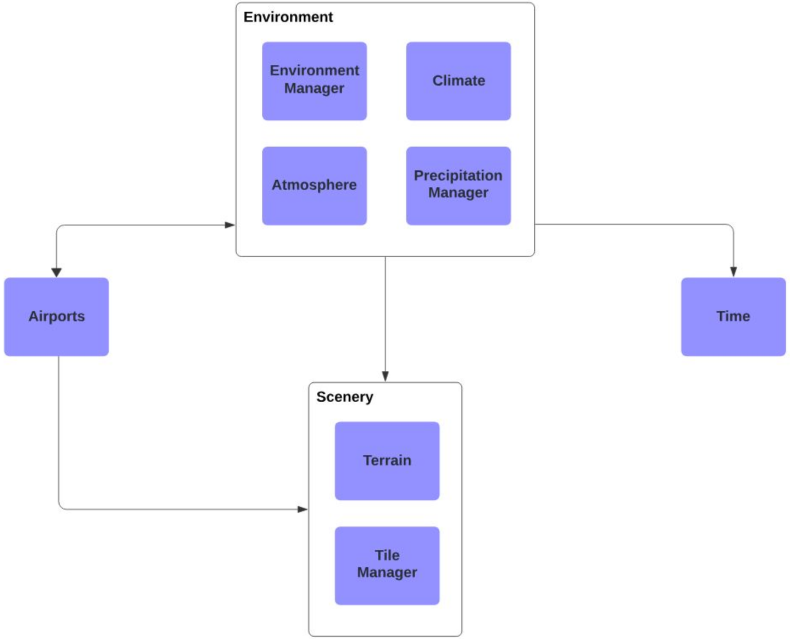
- FlightGear’s codebase uses C++ classes, which are also used for sharing data between components
- Components are very interconnected with each other

Architecture & Reflexion Analysis of One Second-Level Subsystem

Environment Subsystem



Conceptual Architecture



Concrete Architecture

Environment Subsystem Components

Environment

- Multiple subcomponents (atmosphere, climate, precipitation) that implement environment features and aid in FG simulation
- Environment manager component that communicates that implementation to other components within FlightGear and maintains structure of environment component
- Bidirectional dependencies with Airports, depends on Scenery and Time

Scenery

- Represents the landscape within FlightGear
- Terrain subcomponent responsible for pre-sampling of terrain roughness
- Tile Manager has routines for initializing tile manager subsystem, scheduling tile loading based on viewer's position, updating tile queues
- Airports and Environment depend on it

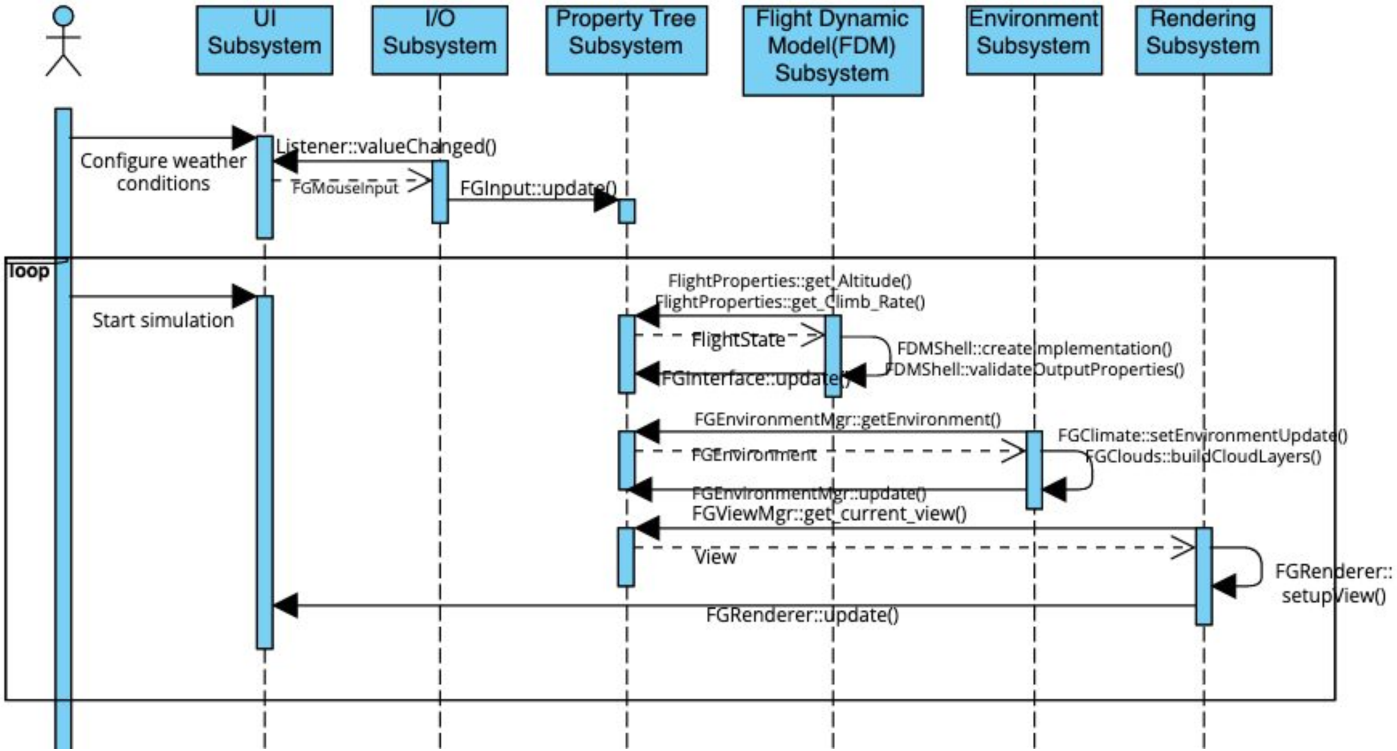
Airports

- Provides the physical outdoor layout of airports, facilitates ground aircraft navigation
- Dependent on Scenery, bidirectional dependencies with Environment

Time

- Provides time-of-day modelling, real local times imported, world time updated relative to frame rates & according to user specifications
- Environment depends on it

Use Case


UI Subsystem
I/O Subsystem
Property Tree Subsystem
Flight Dynamic Model(FDM) Subsystem
Environment Subsystem
Rendering Subsystem
Configure weather conditions
Listener::valueChanged()
FGMouseInput
FGInput::update()
loop
Start simulation
FlightProperties::get_Altitude()
FlightProperties::get_Climb_Rate()
FlightState
FDMShell::createImplementation()
FDMShell::validateOutputProperties()
FGInterface::update()
FGEnvironmentMgr::getEnvironment()
FGClimate::setEnvironmentUpdate()
FGClouds::buildCloudLayers()
FGEnvironment
FGEnvironmentMgr::update()
FGViewMgr::get_current_view()
View
FGRenderer:: setupView()
FGRenderer::update()

Sequence Diagram for **Weather Configuration and Simulation**

Conclusion, Limitations, & Lessons Learned

Conclusion

- Our conceptual architecture was shown to be mostly incorrect, a further updated conceptual architecture would look very similar to our proposed architecture
- Repository, client-server, object-oriented architectural style
- FlightGear's concrete architecture has 10 components, main one is property tree which all other components depend on
- Newly added main loop component that initializes and controls the system allowing FlightGear's processes to run
- Environment and scenery subsystem interacts with time and airports components
- Many components that are interconnected and depend on each other in different ways
- Open-source dev team made architecture harder to recover

Limitations

- Components are all very interconnected so it was difficult to come up with an insightful concrete architecture

Lessons Learned

- Problems in maintaining software architecture (FlightGear's large and highly interconnected code base made it hard to understand the dependencies between components and how they interact)
- How to perform reflexion analysis
- How to compare conceptual and concrete architectures
- Commenting & keeping a record of functionality is important to being a software developer

Resources

[1] FlightGear - Wikipedia: https://en.wikipedia.org/wiki/FlightGear

[2] FlightGear - About: https://www.flightgear.org/about/

[3] FlightGear - Source Code: https://github.com/FlightGear/flightgear

[4] The FlightGear Main Loop: https://wiki.flightgear.org/The_FlightGear_Main_Loop

[5] AI Systems - FlightGear wiki: https://wiki.flightgear.org/AI_Systems

[6] Dijkstra’s Algorithm - Wikipedia: http://en.wikipedia.org/wiki/Dijkstra's_algorithm

[7] TerraGear - FlightGear wiki: https://wiki.flightgear.org/TerraGear