Next Generation Air Traffic Control System - Results Document

Erik Orlowski

The Next Generation Air Traffic Control System (NGATC) was a challenging design problem for me. This was the first time I've designed a modular system (the system I work in professionally is a very monolithic system, although we're working to componentize it). As an embedded C++ developer, this is also the first time I've worked with REST APIs or designed an application using persistence through databases.

In this document, I'll comment on several aspects of the project and my learning experience in implementing it.

Design Patterns

I used several design patterns throughout this project.

In several places, I used the Singleton design pattern when there were classes that would only need one instance and I felt it would be easier for other classes to refer to the class as a singleton. This was mostly for API or manager related classes.

I used the Observer pattern in the System Monitor module for modules to monitor for status updates. This is usually the first design pattern I think to use because I have a lot of experience using it professionally and it solves a very clear problem.

I also used the Strategy design pattern for both of my GUI implementations. This is a design pattern I enjoy using when I have the chance because I find it to be a very lightweight pattern and a very elegant solution when it's appropriate.

Finally, I used the Visitor design pattern in the Static Map module to interact with the objects created in the module. I was surprised to find myself using this pattern, because throughout the course, I struggled to see how the pattern could be useful. In this case, this seemed like a perfect opportunity with business domain classes I wanted to leave alone, being used for a spreadsheet and a JSON report to other modules.

Modular Approach

The modular approach to this design problem was very helpful to me. For pedagogical and professional reasons, I think this was an important skill for me to learn. This is the direction the

PLC industry is moving, and my company is no exception, so I'm glad I had the opportunity to develop this skill in a large scale project.

From an engineering perspective, the modular approach helped me simplify the design problem I was trying to solve. By breaking down the system into layers of abstraction, even the most complex modules were still very reasonable design problems.

Updating Prior Modules

For most of the modules, I did not need to update prior module implementations, as my original notion of how the modules would work together worked out well. However, in two lower level modules, the Weather and Static Map modules, I had to make a fundamental change that was a great learning opportunity for me. For both of these modules, I had originally planned for them to push out updates to higher level modules. Once I got to implementing these lower level modules, I realized that would require me giving them knowledge of these high level modules, which would have negative effects on the portability of these lower level modules.

To fix this, I decided to have these lower level interfaces expose an interface for the higher level modules to request information from. This helped a lot to clean up dependencies and create a cleaner design.

This is an important learning opportunity for me moving forward, that lower level modules should expose APIs that higher level modules can then pull from.

Design Review

The design review was a big help in my design. My partners gave me great feedback about my component diagram and exception handling that I think added important details and clarity to the design.

Entitlement Service Usage

Re-using the entitlement service was a big benefit to the design. I found myself using it very frequently and the fact that it was designed with a portable architecture made it very easy for me to integrate. It was also great to save me from needing to develop a separate module to handle authentication in an already complex project.

Implementing the Design

I believe that I could implement the design and I was able to use the Claude Sonnet 4.5 model and Github Copilot to implement a compiling and running implementation of the design.

For a professional implementation of the system, I believe that there would need to be some more details added around the Kubernetes deployment and UI implementation, although those are details that could be added within the context of the existing design.

From a safety perspective, we divide the potential failures into random failures and systematic failures. This system is well protected against random failures (mostly hardware failures), which can be mitigated using homogeneous redundancy. However, for the level of integrity this system needs, I believe this system is not properly protected against systematic failures. While I described some ways to mitigate these failures in the design document, there is simply not a lot of industry support for truly safety rated distributed systems. This was an interesting takeaway from this class and something I'll be keeping in mind for my capstone project.

Peer Review Comments

My partners had excellent designs, so I did not have much constructive feedback for them. I did leave some comments about access modifiers in their class diagrams that were hopefully helpful for them.