Drone Delivery System Design Document

Our plan to crush the competition...

(img src: Iteration1Requirements.pdf)

Version 1.3.Iteration 2
21th November 2020

Company: ACME
Project: ANVIL
Website: www.ACME-ANVIL.com

Revision Log

Version	Actions	Author	Date
Version 1.1.Iteration 1	Iteration 1	Phyllis Gan	10/30/2020
Version 1.2.Iteration 1	Iteration 1 - peer reviewed	Phyllis Gan	11/12/2020
Version 1.3.Iteration 2	Iteration 2	Phyllis Gan	11/21/2020

Table of Content

1. **Introduction** Page 2, 3
    - 1.1 Purpose
    - 1.2 Background
    - 1.3 Scope

2. **Overview** Page 4, 5
    - 2.1 Background Information
    - 2.2 System Evaluation Process
        - 2.3.1 Iterative Model
            - 2.3.1.1 Iteration 1
            - 2.3.1.2 Iteration 2
            - 2.3.1.2 Iteration 3 (TBD)

3. **Design** Page 6, 7, 8, 9
    - 3.1 Design Overview
        - 3.1.1 Facade Pattern
        - 3.1.2 Factory Class
    - 3.2 Iteration 1 - Design Discussion
        - 3.2.1 Alternative/ Trade-off /Lesson Learnt
    - 3.3 Iteration 2 - Design Discussion
        - 3.3.1 Feature 1 - Drone Observer
        - 3.3.2 Teamwork
        - 3.3.3 Design Tradeoffs
    - 3.4 UML Diagram
        - 3.4.1 Iteration 1 UML Diagram
        - 3.4.2 Iteration 2 UML Diagram

4. **Conclusion** Page 10
    - 4.1 Summary
        - 4.1.1 Iteration 1 Summary
        - 4.1.2 Iteration 2 Summary
    - 4.2 Technology Forecast
    - 4.3 Constraints

5. **Citation** Page 11

1 Introduction

The advancement of technology and the drastic growth of drone technology had caused an increase in the demand for online delivery. People are keen to shop online instead of shopping physically. This scenario creates huge competition between the logistics companies. Thus, our main goal is to create a drone simulation system that is capable of competing with other logistics companies by using the third dimension delivery system. Today, you can order anytime, anywhere, and get it immediately at your doorsteps with almost no-wait-time. However, new technologies require overcoming significant challenges before implementation and deployment. Our design pattern will create a plan that takes consideration into the physics, logistics, route planning, malfunctions, security, and cost behind the drone delivery system. (2020, Iteration1Requirements.pdf)

(img src: Iteration1Requirements.pdf)

1.1 Purpose

The main purpose of this project is to allow the drone delivery system to compete with the other logistics system. We aim to replace the traditional delivery services that require heavy manpower and high cost. Now, package delivery could be done by a single drone.

1.2 Background

This is the first innovative idea to implement drones in a delivery system in history. The technology we created reduces carbon waste by omitting the use of petrol. We use drones over vehicles which are more sustainable and clean energy. As partners, we obeyed ACME Corporation’s three principles:

1. Deliver as fast as possible!
2. Deliver anything that a customer orders, no matter what it is.
3. Deliver anywhere we can.

(img src: google images)

Plus, We care for the environment!

1.3 Scope

The scope of this project will mainly be based on the University of Minnesota - Twin Cities, East Bank, and West Bank campus. The targeted customer will be the faculty, students, campus workers, etc. The routes for the package to deliver mainly focus on these areas:

1. Walter Library
2. Carlson
3. Alumni Center
4. Mariucci Arena

(img src: Iteration1Requirements.pdf)

2 Overview

2.1 Background Information

The drone delivery system is a huge project that requires heavy teamwork that creates an agile working environment. Thus, we are breaking into teams and working together to come up with a design plan and to develop code. Our design pattern is highly based on object-oriented design, application of UML diagram and uses the tools below to create a functional drone delivery system.

2.2 Tools used

1. WebAPI Containers - Docker
2. Programming Languages - C++
3. Communication Tools - Slack & Discord
4. Source Control Tools - GitHub
5. Design Tools - UML
6. Debugger - GDB
7. Testing - Google Test, Unit Test, Regression Test, Integration Test
8. Documentation - Doxygen, PDF

2.3 System Evaluation Process

In this project, the application **Iterative Model** is implied throughout the development process The iterative development process divided into three stages, Iteration 1, Iteration 2, and Iteration 3. In the first stage, it is proof of concept which is known as prototyping, we will then move into iteration 2, the development part. Finally, iteration 3 where we will be analyzing and optimizing the program. Further discussion of the iterative model will be provided below.

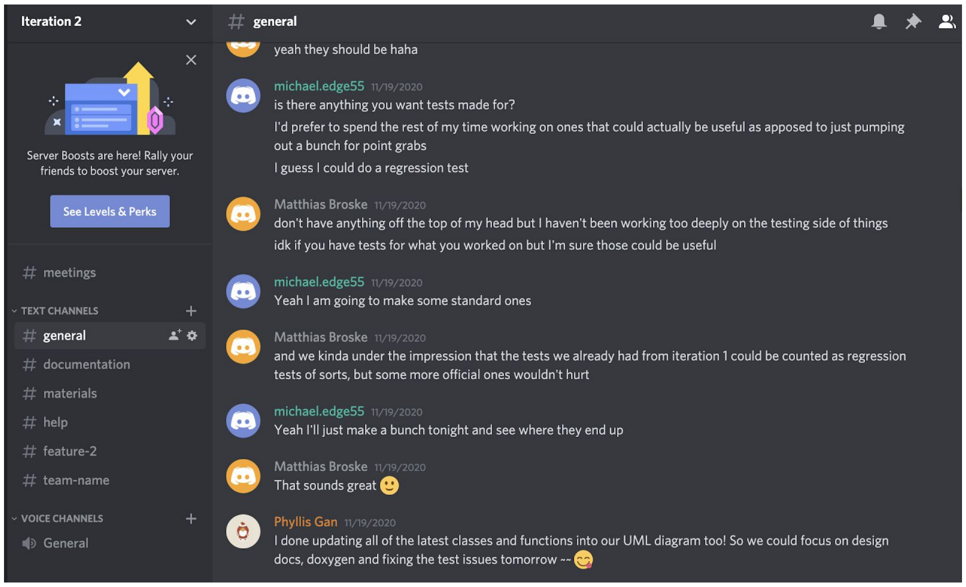
2.3.1 Iterative Model

2.3.1.1 Iteration 1

Currently, at this stage, also known as the proof of concept and prototyping stage. Students are expected to develop a prototype of the drone delivery system and using the design pattern of Facade and Factory pattern that will be further discussed in the **3.1 Design Overview** section. At this iteration, students will need to have a basic structure of a code base to ensure the drone is able to pick up and drop off the package. The implementation will be done individually.

2.3.1.2 Iteration 2

At Iteration 2 also the development stage, students will break into a team of 3 to 4, whereas, the team will distribute tasks based on priority and features. Students will need to work together to implement extra functions for the drone simulation system such as drone and package observer, using the Dijkstra algorithm to find the shortest path, drone pool, etc. In the two weeks of work, we were able to meet up at least 4 times a week to go over our work and showcase what we did to the group in Discord. Further discussion of group work will be discussed in the **3.3 Iteration 2 - Design Discussion** section.



general

yeah they should be haha

michael.edge55 11/19/2020
is there anything you want tests made for?
I'd prefer to spend the rest of my time working on ones that could actually be useful as apposed to just pumping out a bunch for point grabs
I guess I could do a regression test

Matthias Broske 11/19/2020
don't have anything off the top of my head but I haven't been working too deeply on the testing side of things
idk if you have tests for what you worked on but I'm sure those could be useful

michael.edge55 11/19/2020
Yeah I am going to make some standard ones

Matthias Broske 11/19/2020
and we kinda under the impression that the tests we already had from iteration 1 could be counted as regression tests of sorts, but some more official ones wouldn't hurt

michael.edge55 11/19/2020
Yeah I'll just make a bunch tonight and see where they end up

Matthias Broske 11/19/2020
That sounds great 🙂

Phyllis Gan 11/19/2020
I done updating all of the latest classes and functions into our UML diagram too! So we could focus on design docs, doxygen and fixing the test issues tomorrow ~~ 😋

(Evidence of our discord channel)

2.3.1.3 Iteration 3

(TBC…...)

This is where analysis and optimization of the program took place. Our task is to enhance the system and create a data driven system analysis.

3 Design

3.1 Design Overview

Since the drone delivery system requires the various classes to work together, creating a code so that each class works on a specific function and low coupling code function of code is the main aim for our design pattern. Implementation of **Facade Pattern** and **Factory Class** will be fully utilized in the drone delivery system and will be further discussed below.

3.1.1 Facade Pattern

As discussed in lecture, facade pattern allows hiding complex information behind a simple unified interface. The facade pattern is highly used in this project since there are separate projects ongoing and implementation of the facade pattern handles the interaction between Project ANVIL which is the simulation part, Project Rocket took care of the visualization, and Project Jet Powered is where E-Commerce came in place. In this project, there is a huge cross interaction between the Rocket and ANVIL since ANVIL will work on the creation of entities and simulation while Rocket only needs to access the functionality of DroneSimulation class only. Thus, DroneSimulation class will act as a concrete facade that inherits from the DroneDeliverySystem, which is the abstract facade and be the backbone of the projects by linking the subsystem. The implementation of facade patterns in this project is remarkably helpful to reduce coupling and simplify the entire design of the drone delivery system.

3.1.2 Factory Class

A factory class is a creational pattern that uses factory methods to deal with the problem of creating objects without having to specify the exact class of the object that will be created. There are two types of factory class, the concrete factory and abstract factory. In this project, I implement both concrete and abstract factories to create a drone, package and customer object. The *EntityFactory Class* is virtual and allows the concrete factory such as DroneFactory Class, CustomerFactory Class, PackageFactory Class to create objects subsequently. By having the implementation of both concrete and abstract factories, it allows the code base to extend over creation of other entities for future use.

3.2 Iteration 1 - Design Discussion

In Iteration 1, the code base is built based on both the design pattern as discussed above, the factory and facade pattern. First of all, I created an abstract factory class *EntityFactory* that allows inheritance for DroneFactory, CustomerFactory, PackageFactory Classes, and these classes will create each entity object subsequently. Next, each of them will be added to the DroneSimulation class (facade class) in order to show up in the Docker image/ web browser. The next step is to make the drone move by implementing the Euler Integration formula in the Move() function and repeatedly call the Update() function to update the drone position. Calls Update() in the DroneSimulation to update the visual whenever the drone position and package delivery status. To sum up, the drone simulation system highly relies on both factory and facade design patterns and proven from the discussion above.

3.2.1 Alternative/ Trade-off /Lesson Learnt

One thing that I would point out for the alternatives is to implement EntityFactory as a concrete class instead of an abstract class since there are only three entities: Drone, Package and Customer at this point in time. By having concrete classes, it could simplify the code base by reducing extra classes. However, the current code base allows the creation/extension of other entities besides package, drone and customer for future implementation.

3.3 Iteration 2 - Design Discussion

3.3.1 Feature 1 - Drone Observer

My task for this iteration is to create a drone observer which is labeled as Feature 1 in the Iteration2Requirements.pdf. At the early stage, I created AddObserver, RemoveObserver, Notify and GetStatus functions all in the drone class. After discussion with my team, we decided to move over my code into the observer_manager() class which was created by Vivek, who works on the package observer. The observer_manager() class consists of the AddObserver and RemoveObserver functions and both functions allow the creation of a drone and package object in the class. While the remaining notify function will be located in the drone class it will use the OnEvent function. Finally, create a vector of attributes named observer in the drone_simulation class in order to call the AddObserver() and RemoveObserver() in the observer_manager() class. On the other hand, as a member, I actively join every meeting up and help out teammates to resolve bugs. More discussion about teamwork will be in below in the **3.3.2 Teamwork** section.

3.3.2 Teamwork

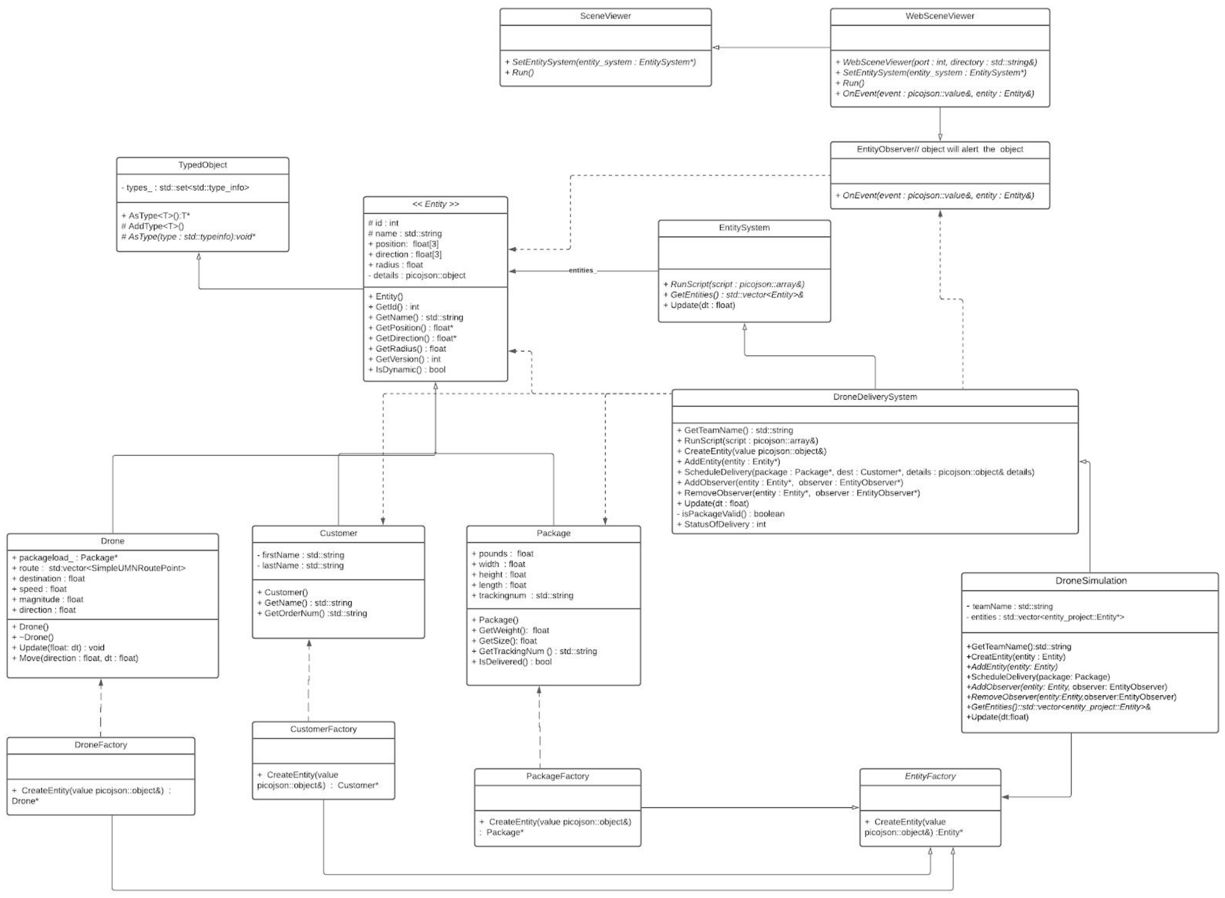
“The Most” is the name of our team and it consists of four people, Matthias, Vivek, Michael, and Phyllis. We all meet up often and discuss any changes/design ideas we had. The best part about us is that we are always happy to help each other and find a great balance in obtaining personal marks and at the same time prioritizing the group points. Whenever we need help, we will send out messages and hop in the voice channel and the team is on our back. I am more than grateful to have this team working together. Speaking into our teammates' work, Matthias is in charge of the drone pool, while Michael is working on the Dijkstra algorithm to find the shortest path, and Vivek working on the Package Observer. Matthias created a CSV reader class to parse the CSV file that contains the new drone types’ information. On the other hand, Michael’s priority was to implement the shortest path and implement the Dijkstra algorithm in our code base. Initially, he wrote an algorithm that looped through all the nodes which took 5 seconds and it is not ideal. Thus, he changed his algorithm to constant time that could allow the simulation to start instantly. We were all together helping him to fix a bug that occurred in the code until 3 am in the morning which is insane and exhausting but it was a great experience. As mentioned above, Vivek and I collaborated on both drone and package observers and decided to create an observer_manager() class that handles the adding and removing of observers from entities.

3.3.3 Design Tradeoffs

One of the trade-offs that we decided was to use the abstract observer EntityObserver over the concrete observer which is the WebSceneViewer. In addition, we also optimize our code base by creating a single observer_manager class instead of creating extra functions in the drone class. The removal of redundant code/functions(since the functionality is the same), has increased the high cohesion and low coupling of patterns in our code base.

3.4 UML Diagram

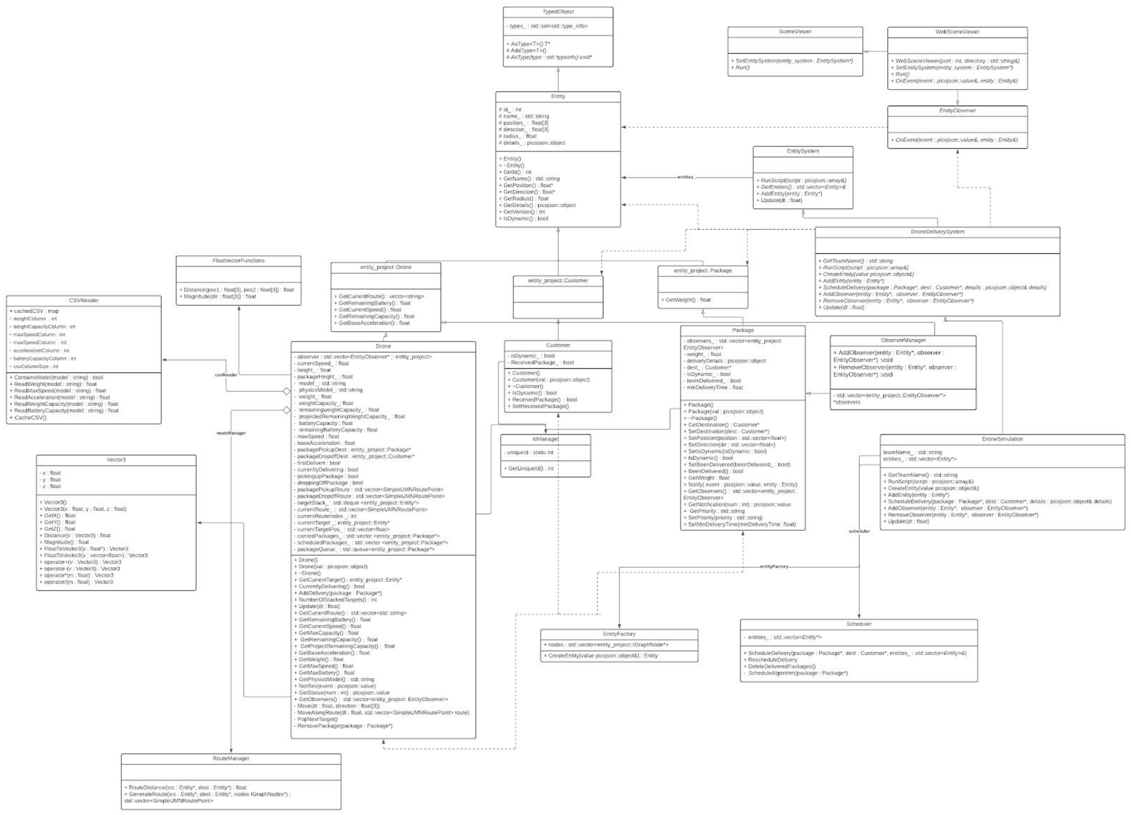
3.4.1 UML Diagram - Iteration 1


SceneViewer
+ SetEntitySystem(entity_system : EntitySystem*)
+ Run()
WebSceneViewer
+ WebSceneViewer(port : int, directory : std::string&)
+ SetEntitySystem(entity_system : EntitySystem*)
+ Run()
+ OnEvent(event : picojson::value&, entity : Entity&)
EntityObserver// object will alert the object
+ OnEvent(event : picojson::value&, entity : Entity&)
TypedObject
- types_ : std::set<std::type_info>
+ AsType<T>():T*
# AddType<T>()
# AsType(type : std::typeinfo):void*
<< Entity >>
# id : int
# name : std::string
+ position: float[3]
+ direction : float[3]
+ radius : float
- details : picojson::object
+ Entity()
+ GetId() : int
+ GetName() : std::string
+ GetPosition() : float*
+ GetDirection() : float*
+ GetRadius() : float
+ GetVersion() : int
+ IsDynamic() : bool
entities_
EntitySystem
+ RunScript(script : picojson::array&)
+ GetEntities() : std::vector<Entity>&
+ Update(dt : float)
DroneDeliverySystem
+ GetTeamName() : std::string
+ RunScript(script : picojson::array&)
+ CreateEntity(value picojson::object&)
+ AddEntity(entity : Entity*)
+ ScheduleDelivery(package : Package*, dest : Customer*, details : picojson::object& details)
+ AddObserver(entity : Entity*, observer : EntityObserver*)
+ RemoveObserver(entity : Entity*, observer : EntityObserver*)
+ Update(dt : float)
- isPackageValid() : boolean
+ StatusOfDelivery : int
Drone
+ packageload_ : Package*
+ route : std:vector<SimpleUMNRoutePoint>
+ destination : float
+ speed : float
+ magnitude : float
+ direction : float
+ Drone()
+ ~Drone()
+ Update(float: dt) : void
+ Move(direction : float, dt : float)
Customer
- firstName : std::string
- lastName : std::string
+ Customer()
+ GetName() : std::string
+ GetOrderNum() :std::string
Package
+ pounds : float
+ width : float
+ height : float
+ length : float
+ trackingnum : std::string
+ Package()
+ GetWeight(): float
+ GetSize(): float
+ GetTrackingNum () : std::string
+ IsDelivered() : bool
DroneSimulation
- teamName : std::string
- entities : std::vector<entity_project::Entity*>
+GetTeamName():std::string
+CreatEntity(entity : Entity)
+AddEntity(entity: Entity)
+ScheduleDelivery(package: Package)
+AddObserver(entity: Entity, observer: EntityObserver)
+RemoveObserver(entity:Entity,observer:EntityObserver)
+GetEntities()::std::vector<entity_project::Entity>&
+Update(dt:float)
DroneFactory
+ CreateEntity(value picojson::object&) : Drone*
CustomerFactory
+ CreateEntity(value picojson::object&) : Customer*
PackageFactory
+ CreateEntity(value picojson::object&) : Package*
EntityFactory
+ CreateEntity(value picojson::object&) :Entity*

*zoom in for better visualization

(revised on 12 November 2020)

3.4.2 UML Diagram - Iteration 2



*zoom in for better visualization

(revised on 20 November 2020)

4 Conclusion

4.1 Summary

4.1.1 Iteration 1 Summary

Currently, we are at Iteration 1 which is the prototyping stage, we have functional code that allows the drone to deliver packages along with some unit testing to test the functionality of the code base. In addition, we are creating detailed documentation on Doxygen for the code and design documents about the entire process for future use. The basic structure of the drone simulation in the previous discussion has proven that the code base has flexibility in preparation to move forward for more advanced and complex design patterns. With forward design thinking, we might create drone or package observers and develop multiple drones or multiple packages to increase the efficiency of the drone delivery system.

4.1.2 Iteration 2 Summary

At this stage, our group is able to implement all 3 priorities and 4 features, thus, we named our team - The Most since we did all of them. A two weeks of heavy teamwork creates an agile work environment which is an extremely helpful skill gained in preparation for corporate environments. Throughout the process, we are able to find a balance between marks distribution and work contribution which drive us toward success. To sum up, it was a positive experience to collaborate with Mattias, Vivek and Michael, and we are looking forward to working together for iteration 3.

4.2 Technology Forecast

In the future, we might expect more customers and might need to create more classes for further use. Thus, code reusability is the main thing in our design pattern.

4.3 Constraints

Due to the petite size of a drone, packages that over 60lb might not be able to deliver by a single drone. We might consider using more than one in the future.

5 Citation

1. Iteration1Requirements.pdf, src-link: https://github.umn.edu/umn-csci-3081-f20/project-portal/blob/master/Iteration1Requirements.pdf
2. Iteration2Requirements.pdf, src-link: https://github.umn.edu/umn-csci-3081-f20/project-portal/blob/master/Iteration2Requirements.pdf
3. Google Images