COS214 Project Report

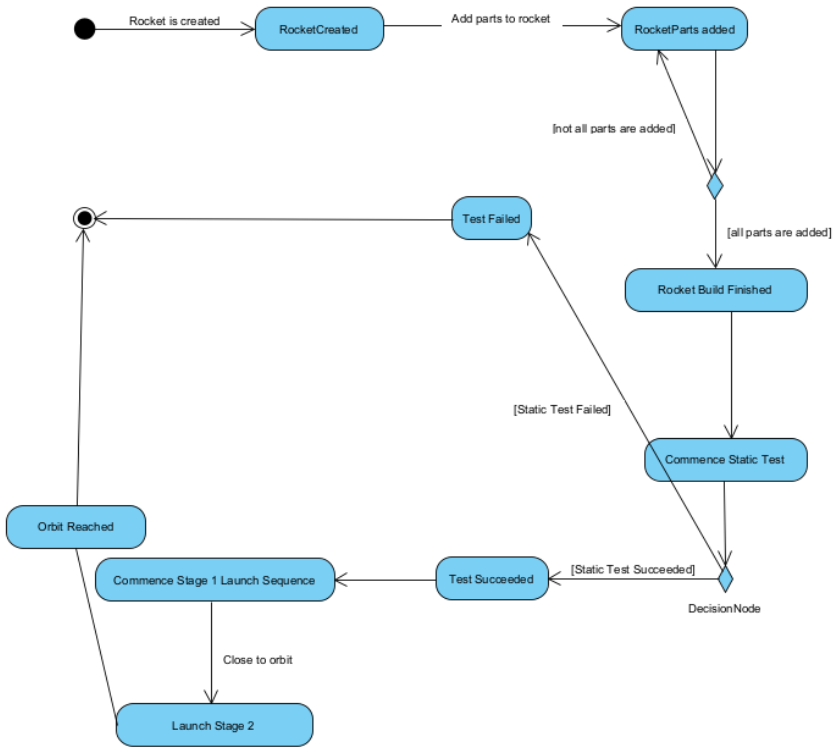
Falcon:

We needed a way to monitor the state of engines in the static test as well as in the actual flight. For this task we decided to use the Observer Design Pattern. We could have also used the State Design Pattern, but the notify() method of the Observer DP had the functionality we wanted, so we opted for that DP. The notify is necessary to ensure that no part of the rockets get damaged, or purely that nothing goes wrong with the engines during launch (or static test).

The rockets obviously consist of multiple parts, which need to be assembled and form one rocket. It went without speaking that we would use the Composite Design Pattern to accomplish this. Composite DP makes this possible to combine many subclasses into one – which is why we use this DP multiple times throughout the program.

As mentioned above, rockets consist of multiple parts. These parts will be recreated thousands of times, and the design of these parts will almost definitely change in the future. For these reasons we used the Abstract Factory Design Pattern to fulfil these requirements. This allows us to make multiple clones of an abstract class. We decided against using the Factory Method here, since Abstract Factory DP works well with composition when creating clones.

Rockets have certain functions which they need to do. This can easily be accomplished by implementing the Command Design Pattern. By using receivers, invokers and command, we can make sure that each respective rocket knows what they should do in the bigger scope of the simulation and launch. It will also make the coding and understanding of the client a lot easier since we can give each rocket the same command, but they will know how to interpret and execute that command correct according to the receiver which handles the individual rockets.



Classes are:

Rocket, RocketPart, Falcon 9 Core, Merlin Engine, Vacuum Merlin Engine, EngineObserver, Observer, RocketPartFactory, Falcon9CoreFactory, MerlinEngineFactory, VacuumMerlinEngineFactory.

Observer design pattern:

Static testing engines.

Subject – RocketPart

ConcreteSubject – Falcon 9 Core, Merlin Engine, Vacuum Merlin Engine.

Observer – Observer.

Concrete Observer - EngineObserver

Composite design pattern:

Component – RocketPart

Composite – Rocket

Leaf –Falcon 9 Core, Merlin Engine, Vacuum Merlin Engine.

(the rockets will be assembled in the main according to the spec.)

Abstract Factory Design Pattern:

Abstract Factory: RocketPartFactory

Concrete Factory: Falcon9CoreFactory, MerlinEngineFactory, **VacuumMerlinEngineFactory:**

Product: RocketPart

ConcreteProduct: Falcon 9 Core, Merlin Engine, Vacuum Merlin Engine.

Command Design Pattern:

Receiver – Rocket

Command – Command

ConcreteCommand – CommenceStaticTest, PrepareForLaunch, Launch, CommenceROLLProgram, CommenceStage2 (Stage 1 lands in the ocean), Release.

Invoker - Button

<u>Dragon Spacecraft:</u>

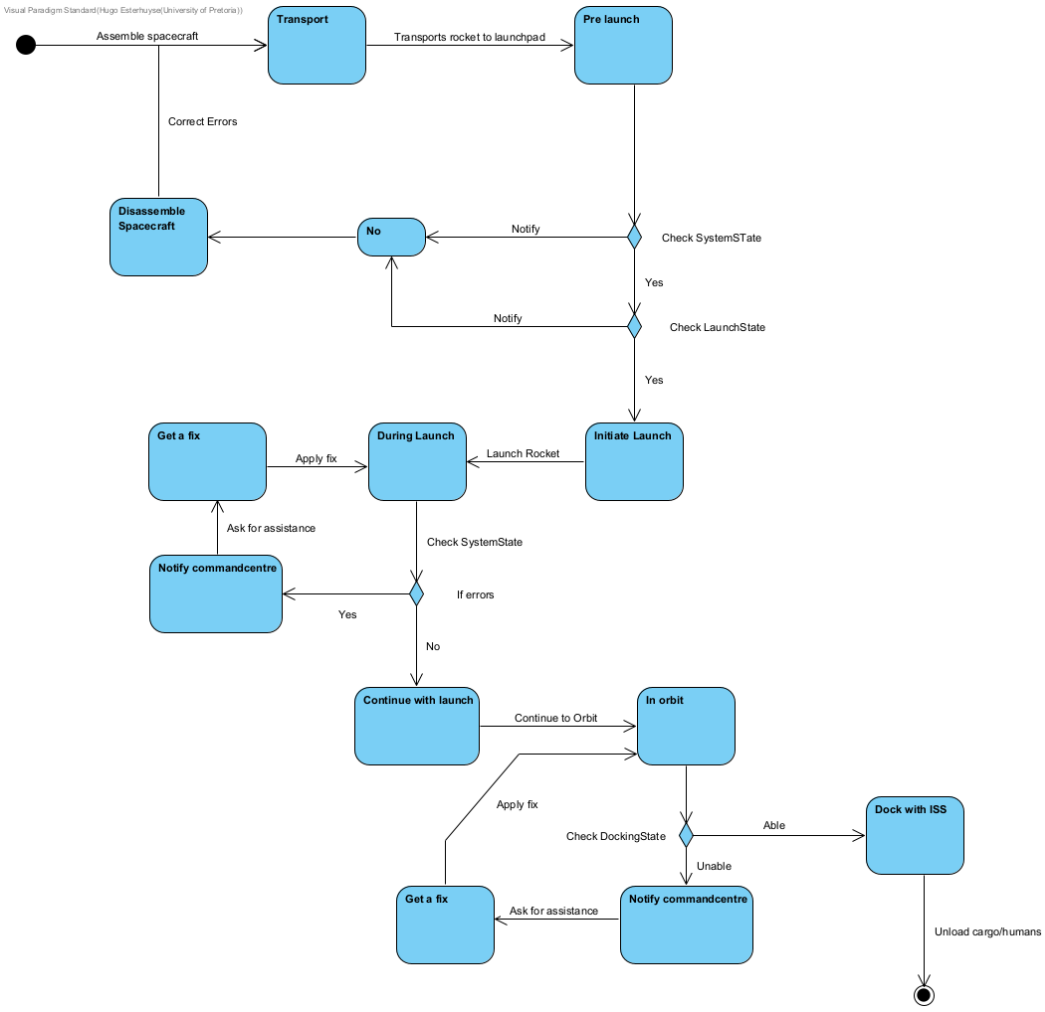
For the Dragon Spacecraft part of the project, several design components were utilised. They can be summarised with explanations as follows:

1. Abstract Factory: To create and generate new dragon spacecrafts we used the factory and abstract factory method to produce a spacecraft whenever it is necessary because the actual Dragon craft needs to be attached to the falcon rockets, so it makes sense to be able to produce a craft whenever required. It is also necessary to make use of concrete factories for the two different types of crafts that need to be built.
2. Decorator: The decorator design pattern is used to add several important parts to the different spacecrafts depending on their required specifications. These concrete decorators represent important aspects of the Dragon Spacecraft that get used and that also importantly need to be monitored for a safe and successful simulation. This design pattern allows us to add multiple different types of external or internal components to the dragon crafts after the actual craft has been built.
3. State: This pattern was used to monitor the state of all the added decorations and the spacecrafts themselves to make sure everything is working pre, during and post launch (docking and deployment included). It

is important that state be used to monitor the dragon spacecrafts so that if any errors should arise, they can be attended to such that a successful simulation occurs.

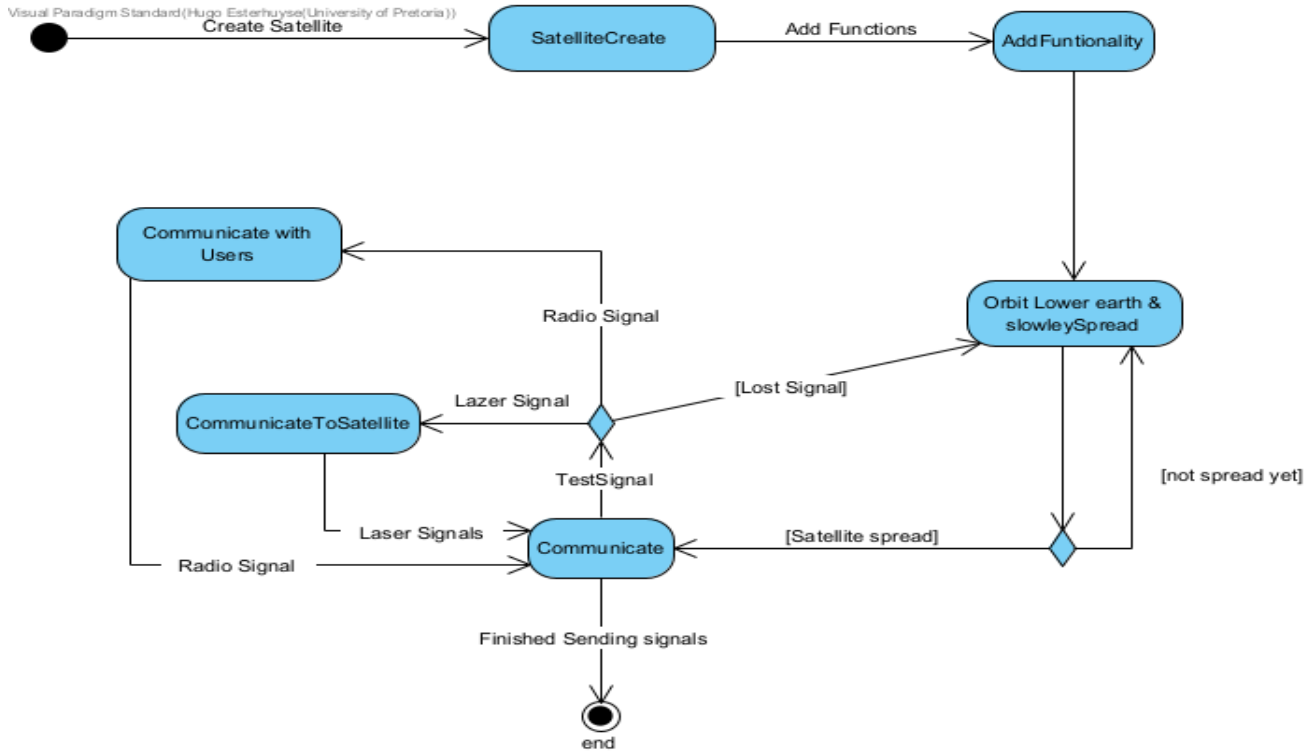
4. Observer: Observer was used to monitor certain important parts of the dragon crafts that can experience failures during important steps of a rocket launch and cargo deployment. Observer can monitor these possible failure states, update and notify the user and then eventually “fix” the failures. This design pattern was done in such a way to allow multiple observers to be added to one craft as there could be multiple systems or deploy observers for example on one craft and this pattern makes it very simple to literally observe and fix problems that may arise during simulation.

Using all these design components together we have built a Dragon Spacecraft that is correctly always monitored to ensure correct functioning throughout the entire simulation. Whilst there are obviously more design patterns that can be used to provide other functionality, we felt these would be the best and most simplistic patterns to give us what we wanted for what we felt necessary to include.

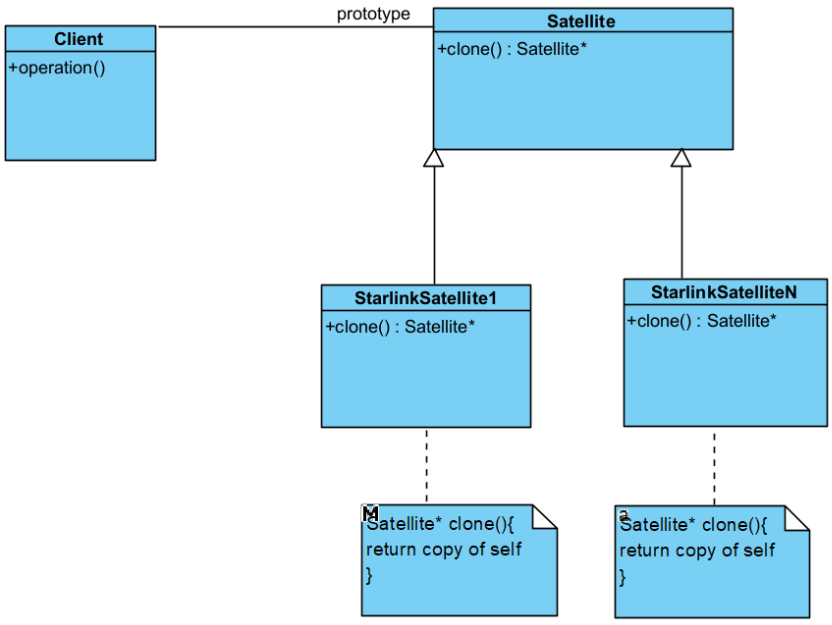


Starlink Satellites:

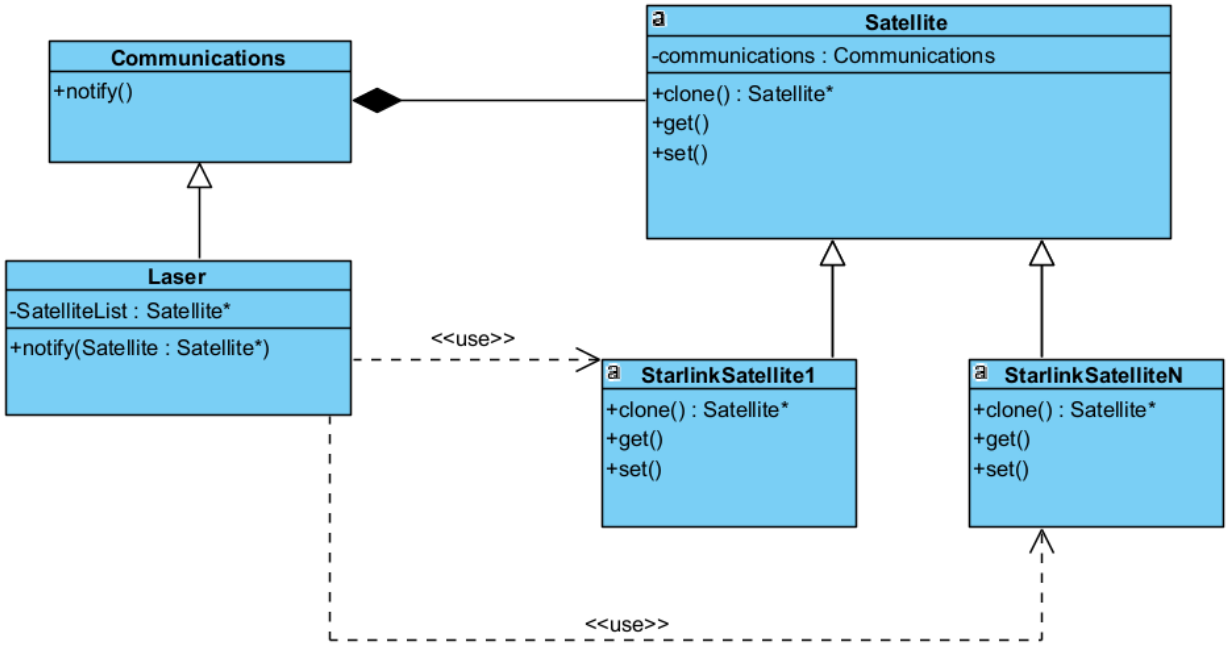
We needed a way to create a Starlink Satellite and the factory method suited our needs well. We only require one type of satellite so the system does not need to be expanded upon in the future, therefore the factory method was used instead of the abstract factory method.



The Prototype design pattern was used to create the multiple satellites that will orbit in space and communicate with one another as well as the users on the ground. The prototype design pattern's clone function made it easy to create multiple satellites at a time, and the original satellite that was created using the factory method was used to clone, that is why we chose it.



The Starlink Satellites needed a way to communicate with each other as well as with the clients on the ground, thus the Mediator design pattern was used. It gives an interface for the satellites to communicate with each other while not referring to each other explicitly. Thus the Mediator design pattern suited our needs perfectly.



1.4

- Factory method:
  - Product: Satellite
  - ConcreteProduct: StarlinkSatellite
  - Creator: SatelliteFactory
  - ConcreteCreator: StarlinkFactory
- Prototype:
  - Prototype: Satellite
  - ConcretePrototpye: StarlinkSatellite
- Mediator:
  - Mediator: Communications
  - ConcreteMediator: Laser
  - Colleague: Satellite
  - ConcreteColleague: StarlinkSatellite

Simulator:

The simulator was written in the main.cpp file and allows the user to configure the rocket and launch details in any way they want.

Link to google docs version:

https://docs.google.com/document/d/12AeBli5M9MZXyTTcI5xTEf6HfAIKvq1V6lZ3iqAaQJ8/edit?usp=sharing

Github repository link:

https://github.com/RainbowPeanut69/The_Factory_Methheads---COS214_Project