COS 214 Project – Team 27: Anonymous

Members: Michael Geere – u20466570, Cameron Brammer – u20494654, Danie Schoeman – u15036058, Dian Lahouter – u20454342, Durandt Uys – u18286250, Matthew Verster – u16016239

GitHub: https://github.com/Jameson4/COS-214-Project

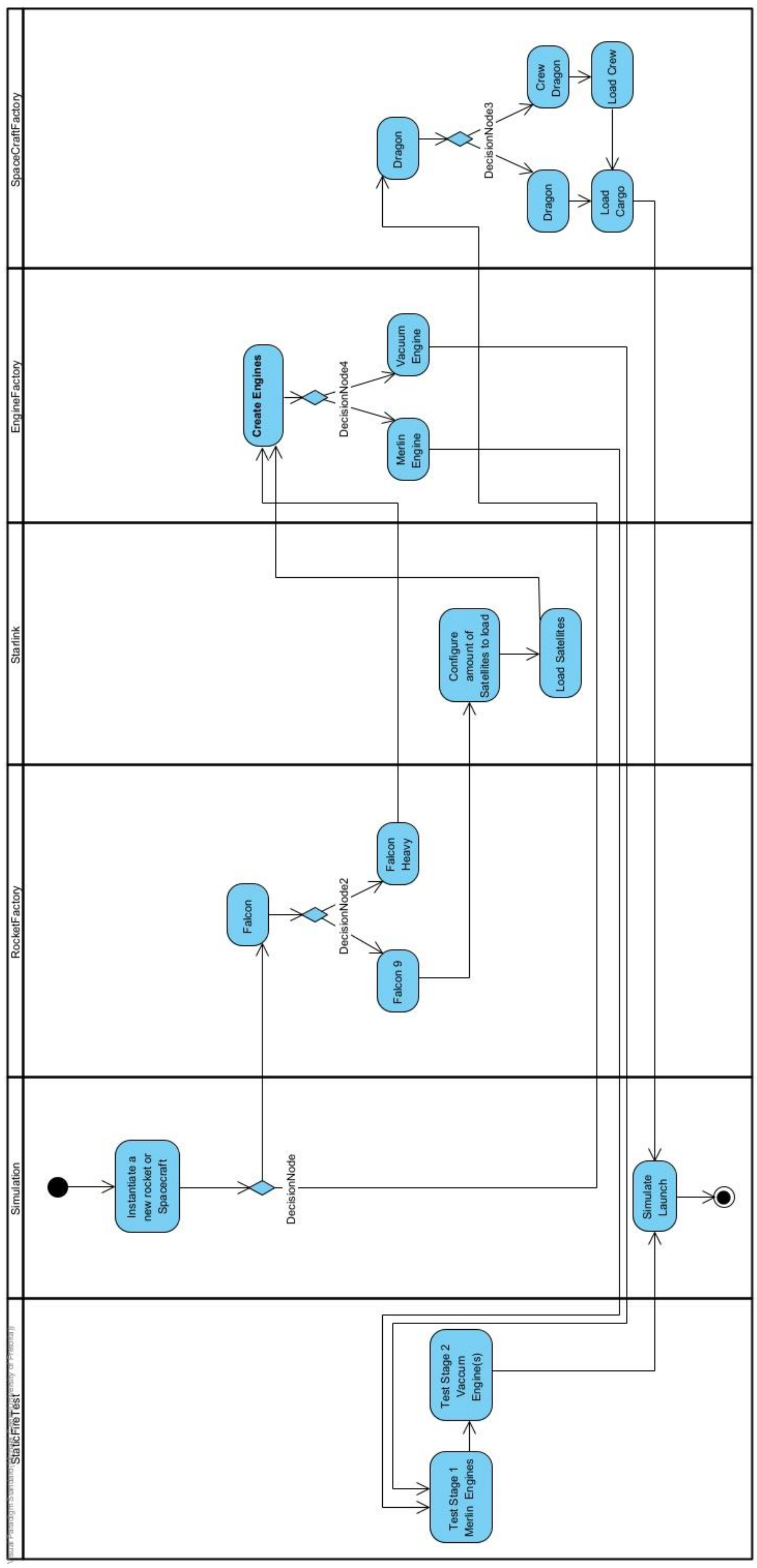
This document:
https://docs.google.com/document/d/1pURJH1W3fZ-iefLiaipNMhxc8e8vtEnxp6vnFfpZFyc/edit?usp=sharing

Initial Design: Task 1.1-1.8

Task 1.1: Functional Requirements

1. Static Fire test to check if all engines (9 Merlin engines and a single Vacuum Merlin Engine for Falcon 9 and 27 Merlin engines and a single Vacuum Merlin Engine for Falcon Heavy) are functional before launch.
2. Configure the optimal cluster size of rockets to place onto a Falcon 9 rocket.
3. Separating the stages of the rocket.
4. Functionally landing stage 1 of both the Falcon 9 and the Falcon Heavy in order to reuse them.
5. Activate the single Vacuum Merlin Engine which will send the payload into orbit.
6. Equally distribute the amount of Starlink satellites that were placed on the Falcon 9 rocket.
7. Communicating between these satellites using lasers.
8. Communicating between these satellites and users on the ground using radio signals.
9. Simulate a launch in test mode which can be interrupted, tweaked and then allowed to continue.
10. Setup and run a simulation for a launch as if it were real.
11. Setup and run a simulation for a launch as if it were real in batches.
12. Store the data of the actual simulations.
13. Command rockets and spaceships to takeoff, separate, land, launch satellites etc.

Task 1.2: Process represented by Activity Diagrams



Task 1.3: Design Patterns

1. Abstract Factory:
    a. The abstract factory design pattern will be used to create a rocket or a Dragon Spacecraft. It will allow us to define attributes for the rocket such as the type of rocket (Falcon 9 or Falcon Heavy), number of Starlink satellites in the cargo and the number of Merlin engines. We can also define attributes for the Dragon Spacecraft such as type of spacecraft (Dragon or Crew Dragon), number of crew members and type of cargo.
2. State:
    a. The state design pattern will be used to monitor the behaviour of the rocket's engines and their stages in takeoff, separation and landing.
3. Prototype:
    a. The prototype design pattern will be used to clone the Starlink satellites that will be loaded onto the Falcon 9 rocket. These rockets are identical so therefore caching on an implementation level would increase efficiency.
4. Command:
    a. The command design pattern will be used to provide control of the rockets and spaceships to users on the ground. These users can command the rockets to takeoff, separate, land (stage 1) and release Starlink satellites.They can also command the spacecraft to takeoff and land.
5. Observer:
    a. The observer design pattern will be used in conjunction with the state design pattern to create an interface in which states can be easily changed and allow other classes to be notified of these changes. The observer will attach to various rockets and monitor their state and notify other classes involved with the various stages of the rockets.
6. Strategy:
    a. The strategy design pattern will be used in the 2 stages of each rocket. This will provide an interface for each algorithm dedicated to the different sections of each stage and can easily work in conjunction with the state design pattern and the observer design pattern.
7. Template:
    a. The template design pattern will be used with the rocket classes, the template method will invoke all the takeoff and landing procedures and thus will create a skeleton base algorithm for all methods a rocket will use.
8. Memento:

   a. The memento design pattern will be used to store the results of the simulation. In the result of a successful launch, the caretaker will restore the rocket to its original internal state before that launch.

9. Chain of responsibility:
   a. The chain of responsibility design pattern will be used to allow multiple objects' functions be called at the same time and in an efficient way. This design pattern will mainly be used with the falcon 9 cores and the engines connected to them.
10. Decorator:
    a. The decorator design pattern will be used to dynamically add extra functionality to the rocket and Spacecraft classes. Cargo and Passengers can be added to the Spacecraft classes.

Task 1.4: Classes

StaticFireTest

RocketState (State Design Pattern)

Launched (Concrete State)

Aborted (Concrete State)

Ready (Concrete State)

Landing (Concrete State)

Landed (Concrete State)

RocketFactory (Template Design Pattern, Abstract Factory)

Falcon9Factory (Concrete Factory)

Falcon9 (Concrete Product)

FalconHeavyFactory (Concrete Factory)

FalconHeavy (Concrete Product)

Falcon (Adapter Class)

Stages (Strategy Pattern)

rocketStage (Strategy)

StageOne (Concrete Strategy)

StageTwo (Concrete Strategy)

EngineFactory (Abstract Factory)

MerlinEngineFactory (Concrete Factory)

VacuumEngineFactory (Concrete Factory)

Engines (Abstract Object)

MerlinEngine (Factory Object)

VacuumEngine (Factory Object)

Payload (Decorator)

Falcon9Core (Chain of responsibility)

SpaceCraftFactory (Template Design Pattern, Abstract Factory)

CrewDragonFactory (Concrete Factory)

CrewDragon (Concrete Product)

DragonSpacecraftFactory (Concrete Factory)

DragonSpacecraft (Concrete Product)

Humans (Decorator)

Cargo (Decorator)

InternationalSpaceStation (Decorator)

Starlink (Observer Subject)

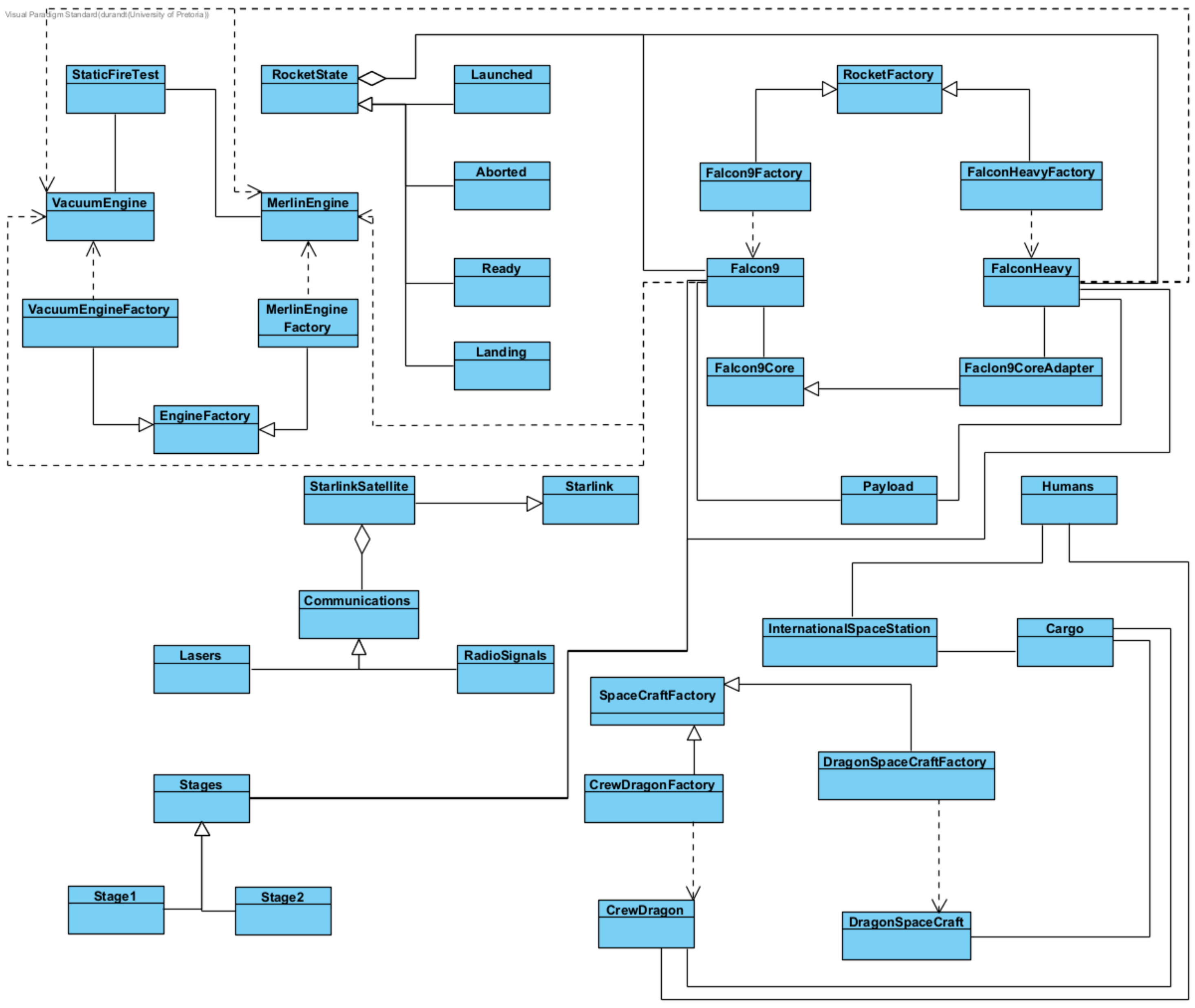
StarlinkSatellite (Observer)

Communications (ConcreteSubject)

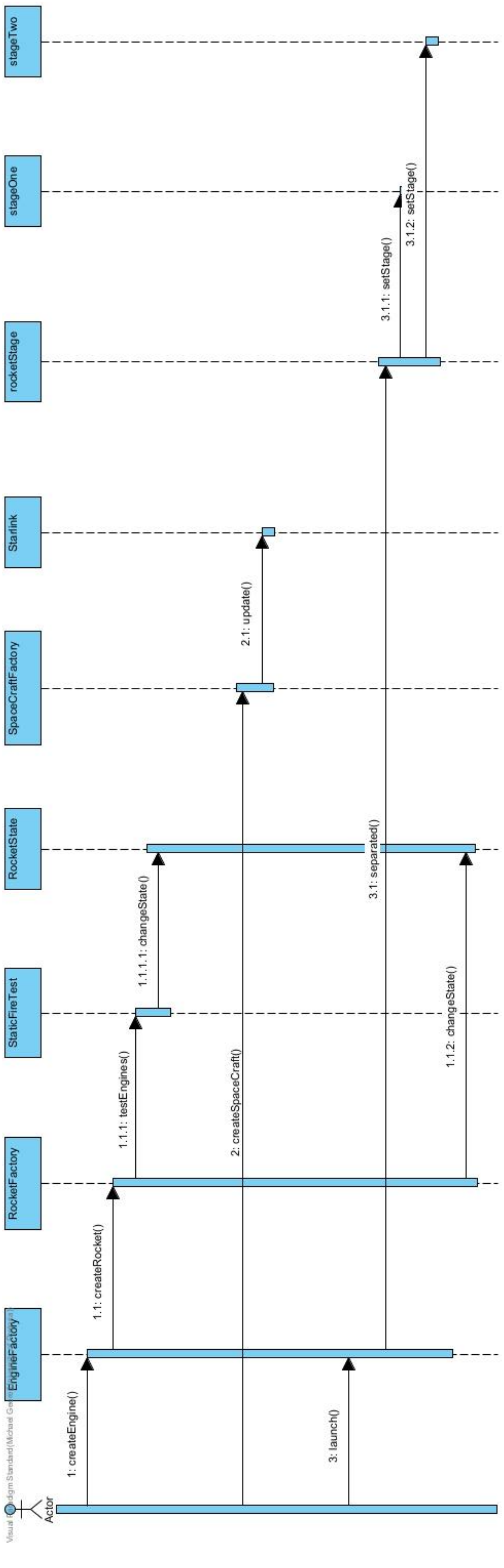
Satelite (Concrete Observer)

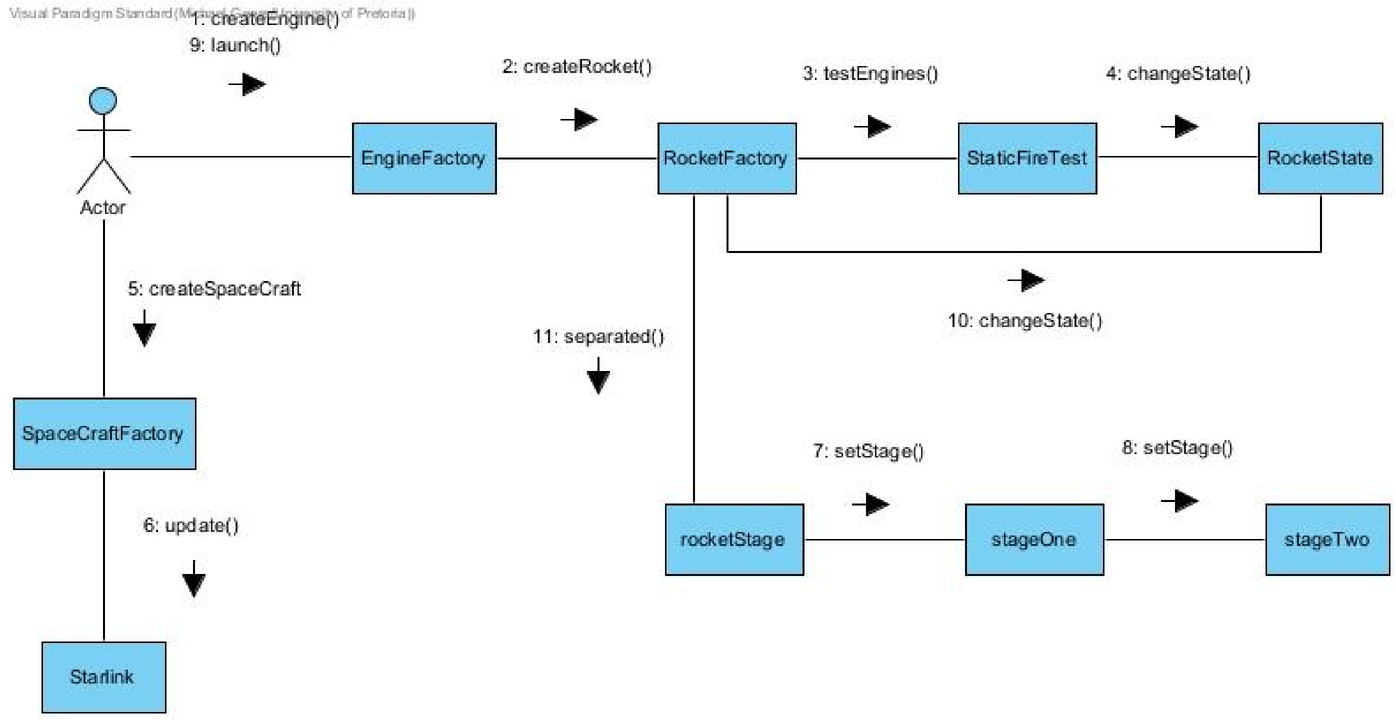
Simulation (Memento)

Task 1.5: Class Diagram (include multiplicity)



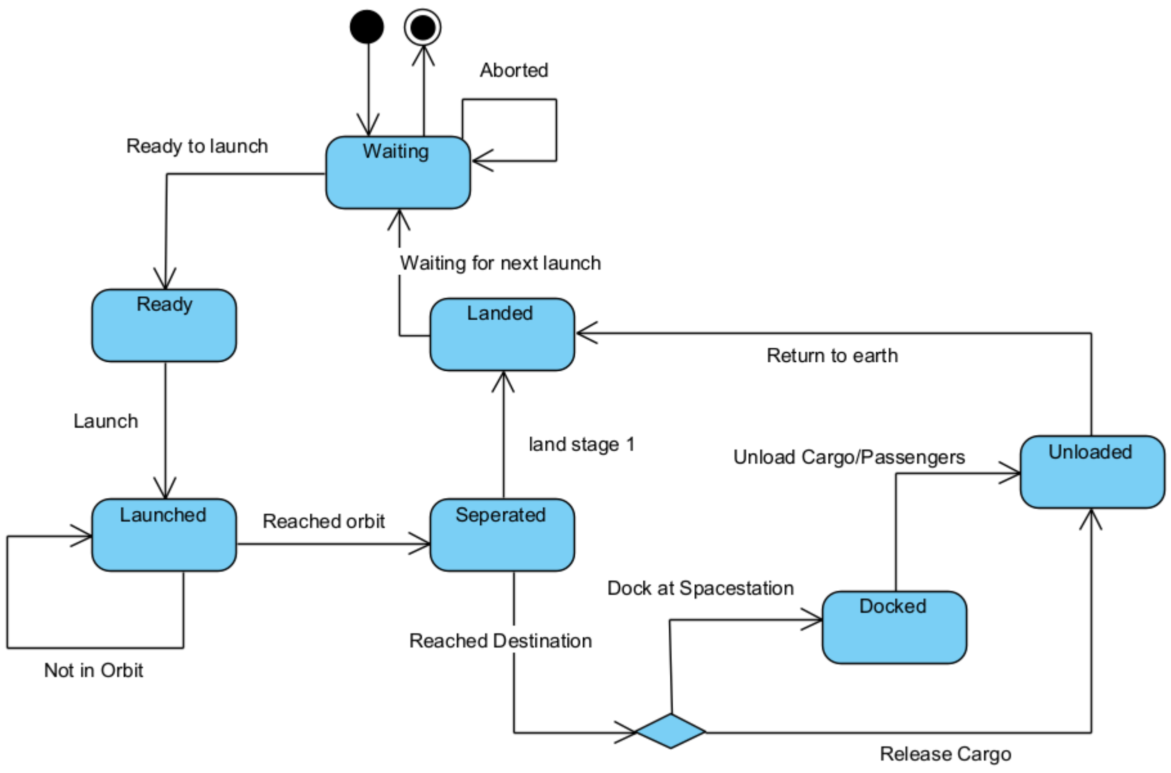
Task 1.6: Sequence and Communication diagrams



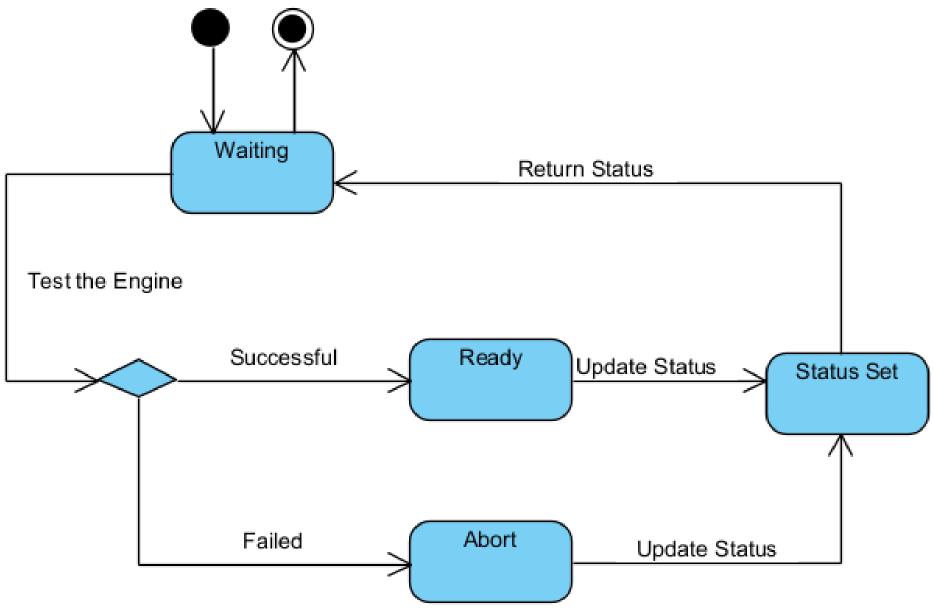

1: createEngine()
9: launch()
2: createRocket()
3: testEngines()
4: changeState()
Actor
EngineFactory
RocketFactory
StaticFireTest
RocketState
10: changeState()
5: createSpaceCraft
11: separated()
SpaceCraftFactory
7: setStage()
8: setStage()
6: update()
rocketStage
stageOne
stageTwo
Starlink

Task 1.7: State diagrams

Complete Spacecraft state diagram for the whole launch:

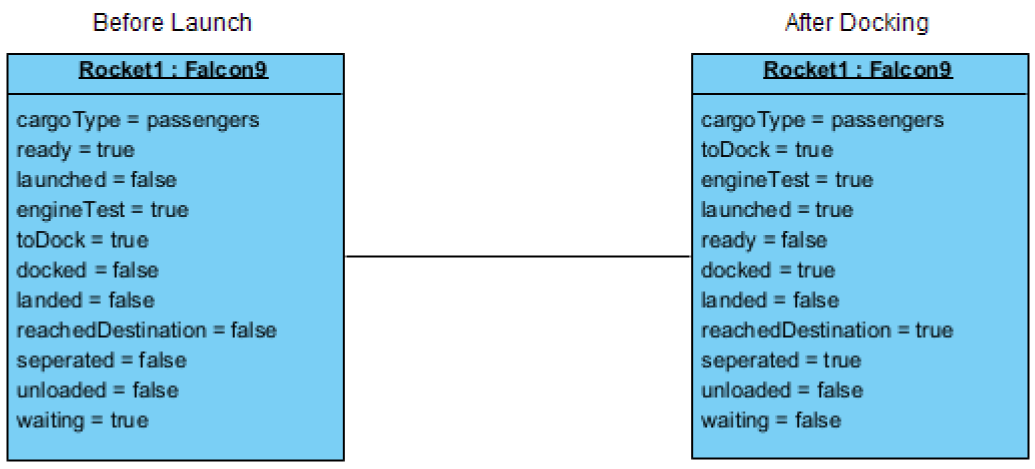


State diagram for the engine test:

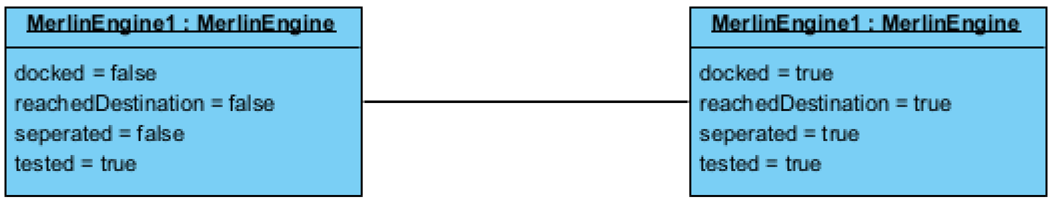


Task 1.8: Object diagrams

Complete Falcon9 spacecraft object diagram, before launch and after docking.



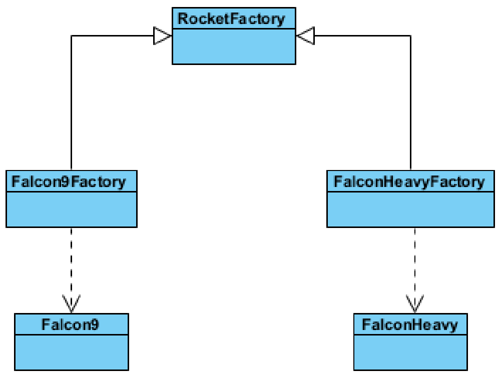
Single Merlin Engine object diagram, before launch and after docking.


MerlinEngine1 : MerlinEngine
docked = false
reachedDestination = false
seperated = false
tested = true
MerlinEngine1 : MerlinEngine
docked = true
reachedDestination = true
seperated = true
tested = true

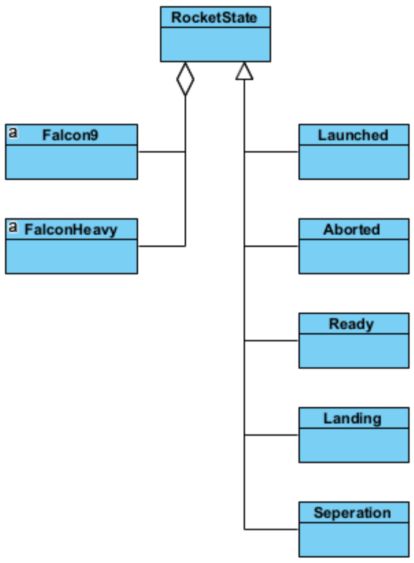
Report: Task 3

Design Patterns used and their use case

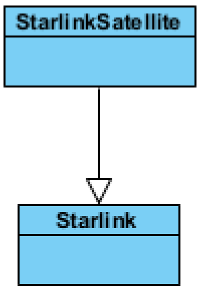
1. Abstract Factory:
    a. The abstract factory design pattern was selected for the purpose of creating the various rockets. We needed a way of creating various types of rockets. This design pattern was chosen as all of the rockets are related but have different functions that can easily be created with this design pattern through abstract interfaces. This is the main design pattern in this project as this is the basis where we can create and implement our rockets for use and testing.



2. State:
    a. We also need to monitor the state and behaviour of the different rocket engines on the rockets. The best fit pattern for this is the State design pattern because the rocket's engines will change according to a state, if they are running or if separation has occurred and they need to start up or if any error occurs. This pattern allows us to monitor the state of the engines but also alter the behaviour according to the various states they are in without having to try and accommodate all states the engines might find themselves in.

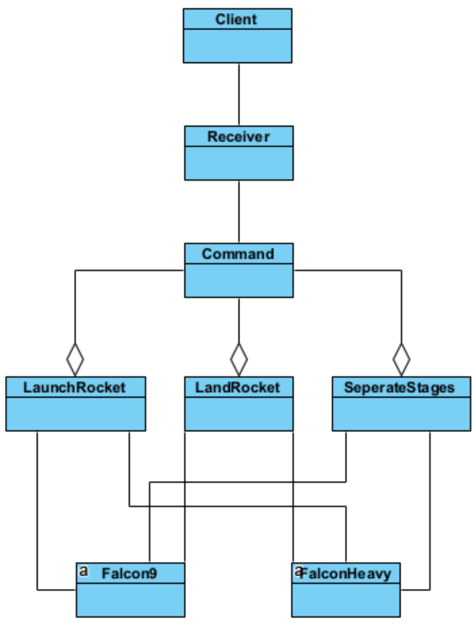


3. Prototype:
    a. One of the main rocket payloads will be Starlink Satellites. These satellites are identical and it would be a burden to have to create each one separately. This is why we chose to use the Prototype design pattern as this allows us to clone a certain instance of an object to create a large amount of said object. This means we will be able to create a lot of satellites quickly and efficiently for use in the rockets.



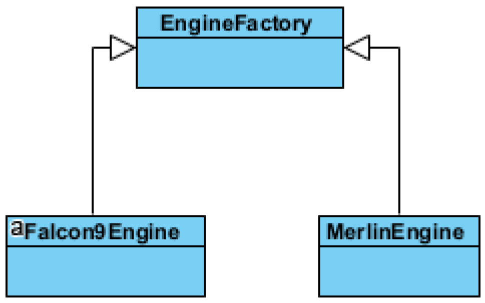
4. Command:
    a. Our next requirement is that users on the ground should be able to launch land, separate and release satellites. The Command pattern was selected for this purpose as this allows us to send one command to the pattern and that will call and execute the relevant execute command, be it to launch or any other needed function. We can send through an object which lets us execute a command according to the object that calls it as in a landing or launching object. This mitigates the need to implement every type of request the user

can do and making sure the request goes to the correct classes. This in turn also reduces overhead.

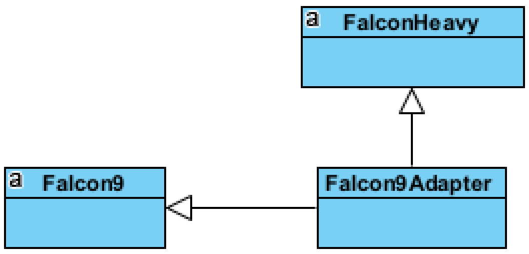


5. Observer:
    a. The observer is needed to work in conjunction with the State pattern, we will attach observer objects to the various parts of the rocket. As states change we need to be able to notify other classes of the changes. If a rocket separates we need to be able to notify the next stage to start executing. Using this with the state pattern also allows us to monitor and change the state in a simple way for testing mid flight of the rocket.
6. Strategy:
    a. The strategy design pattern allows you to have various algorithms and interchange them according to the class that it's being used in. This is advantageous when the user or the program needs to interact with the different parts of the rocket, mainly engines and payload delivery systems. This allows the algorithms to be implemented in one place and without the need of large conditional systems to select which algorithm is to be used. This allows easy communication between the parts of the rocket during stage changes. This works well with the observer and state pattern as we can easily get the current state of an engine and notify the strategy and run our algorithms accordingly.
7. Template:

   a. All the rockets will have similar methods for takeoffs and landings and possibly other actions. It’s easier to implement a skeleton class that contains the various algorithms and methods that are shared between the rockets. So all the rockets can easily implement the same algorithm that’s needed for takeoff and landing. But also that each rocket can easily implement rocket specific methods from the template method if needed. In this scenario we are using the EngineFactory as the Template class also. This means that the engines it produces ie. Falcon9 and Merlin can implement the Template class methods.

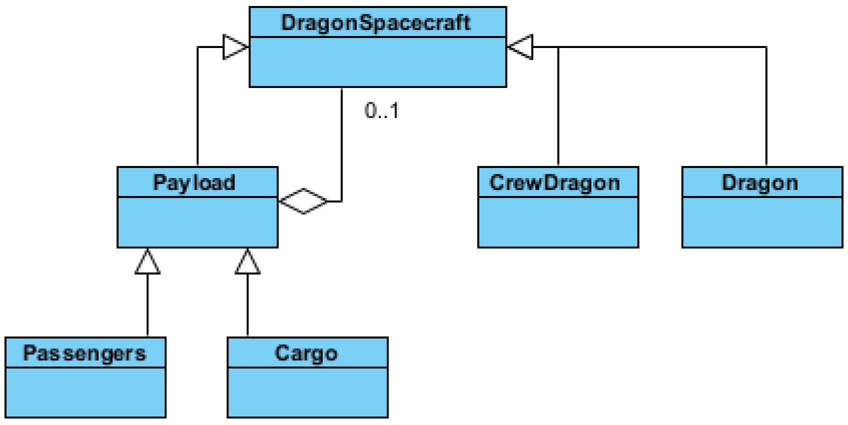


8. Memento:
   a. We need a way of checking the results and states of the various rocket components at certain times. This is the main purpose of running a simulation, to check whether anything has failed or was successful during the launch simulation. We also need to get results of the simulation launch for research. The Momento does exactly this, it allows us to capture states of parts if the rocket. The other benefit is that it also allows us to reset the rocket to the original state and rerun simulations. It also allows the user to set a certain part of the rocket to a previous state for a new simulation.
9. Adapter:
   a. There are many parts in a rocket and at some point some parts will have incompatible interfaces. The Adapter pattern will be used as a more general pattern that allows it to adapt interfaces to work together, one place where this will be needed is for the firing of the different rockets. Below you can see that we use the Falcon9 interface as our base and we adapt FalconHeavy to use the class interface of Falcon9.



10. Decorator:

   a. The decorator pattern will also be used as a more general use pattern. This pattern allows us to dynamically add functionality to rockets and spacecraft, we will be able to add cargo and passengers to the spacecraft. But it also allows us to add any future functionality to the rockets to be used in the simulations ie. if a new type of cargo is developed or an engine gets increased power or capability. We can add these easily to the simulation without rebuilding the whole system. Below you can see the payload is used as the decorator to add passengers and cargo to the spacecraft.


DragonSpacecraft
0..1
Payload
CrewDragon
Dragon
Passengers
Cargo