COS214 PROJECT

Group Name: Code Breakers

DESIGN

Approach:

We broke the Spacex and Starlink simulation into 4 subsystems, namely Falcon rockets, Dragon spacecraft, Starlink Satellites and Launch simulator. We combined the first three subsystems in the last subsystem "Launch Simulator". A breakdown of these subsystems is listed below.

Link To our Github repository:

- https://github.com/FaraiQC/COS_214_Project

Link To our Google docs:

- https://docs.google.com/document/d/1gs0NNO9YN0JF_Gwg1Z9vqOSuz4UBR-yM_8e0tlxygJ8/edit?usp=sharing

Table of Contents

The Falcon Rockets **3**
Task 1: Design 3
1.1 Functional Requirements 3
1.2 Activity Diagrams 5
1.3 Design Patterns 9
1.4 Classes and Design Patterns 9
1.6 Sequence and Communication Diagrams 11
1.7 State Diagrams 13
1.8 Object Diagram 15

The Dragon SpaceCraft **16**
Task 1: Design 16
1.1 Functional Requirements: 16
Crew Dragon 16
1.2 Activity Diagrams 17
1.3 Design Patterns 18
1.4 Classes And Design Patterns 18
Creator: ParachuteCreator 19
1.5 Class diagram 20
1.6 Sequence and Communication Diagram 21
1.6.1 Sequence Diagram 21
1.6.2 Communication diagram 22
1.7 State Diagram 23
1.8 Object Diagram 25

Starlink Satellites **26**
Task 1: Design 26
1.1 Functional requirements: 26
1.2 Activity Diagram 27
1.3 Design Patterns 28
1.4 Classes and their interrelationships 28
1.5 Class Diagram 29
1.6 Sequence and Communication Diagrams 29
1.8 Object Diagram 32

Launch Simulator **33**
Task 1: Design 33
1.1 Functional requirements: 33
1.2 Activity Diagram 33
1.3 Design Patterns 35
1.4 Classes and their interrelationships 35
1.5 Class Diagram 35
1.6 Sequence and communications Diagram 35
1.7 State Diagram 37

Link To our Github repository: **38**

Design patterns used: **38**

The use of the Design patterns in each subsystem **39**
Falcon Rockets: 39
Template design pattern: 39
Factory Design pattern: 40
State Design pattern: 40
Builder Design pattern: 41
Composite Design pattern: 42
Dragon Spacecrafts: 44
Observer Design pattern: 44
Factory Method Design Pattern: 44
Starlink Satellites: 47
Prototype design pattern : 47
Iterator design pattern : 47
Observer design pattern : 48
Mediator design pattern : 48
Launch Simulator: 49
Command Design pattern: 49

The Falcon Rockets

Task 1: Design

1.1 Functional Requirements

I. Initial Simulation - <u>TEST ("static-fire-test")</u>

A. Falcon 9

1. Individual Stage Testing
    a) 1st stage testing:
        - All Falcon 9 cores must be initiated(STATIC POWER)
        - All Falcon 9 Engines must be on "READY_STATE".
        - When All cores are at optimum, proceed to unit testing.
        - When an Engine fails, retest then proceed to unit testing
    b) 2nd stage testing:
        - Initiate the Vacuum Merlin Engine(ON - STATIC POWER)
        - When Vacuum Merlin Engine is at optimum, proceed to unit testing
        - When Engine fails, retest then proceed to unit testing
2. Unit Testing (NOT CMAKE)
    a) Engage All Falcon 9 cores.
    b) Engage the Merlin Vacuum Engine(ON - STATIC POWER).
    c) If all Engines are at optimum, Notify system, and -
    d) Proceed to launching.

B. Falcon Heavy

1. Individual Stage Testing
    a) 1st stage testing:
        - All three Falcon cores must be initiated(STATIC POWER)
        - All 27 Engines must be on "READY_STATE".
        - When All Engines are at optimum, proceed to unit testing.
        - When an Engine fails, retest then proceed to unit testing
    b) 2nd stage testing:
        - Initiate the Vacuum Merlin Engine(ON - STATIC POWER)
        - When Vacuum Merlin Engine is at optimum, proceed to unit testing
        - When Engine fails, retest then proceed to unit testing.

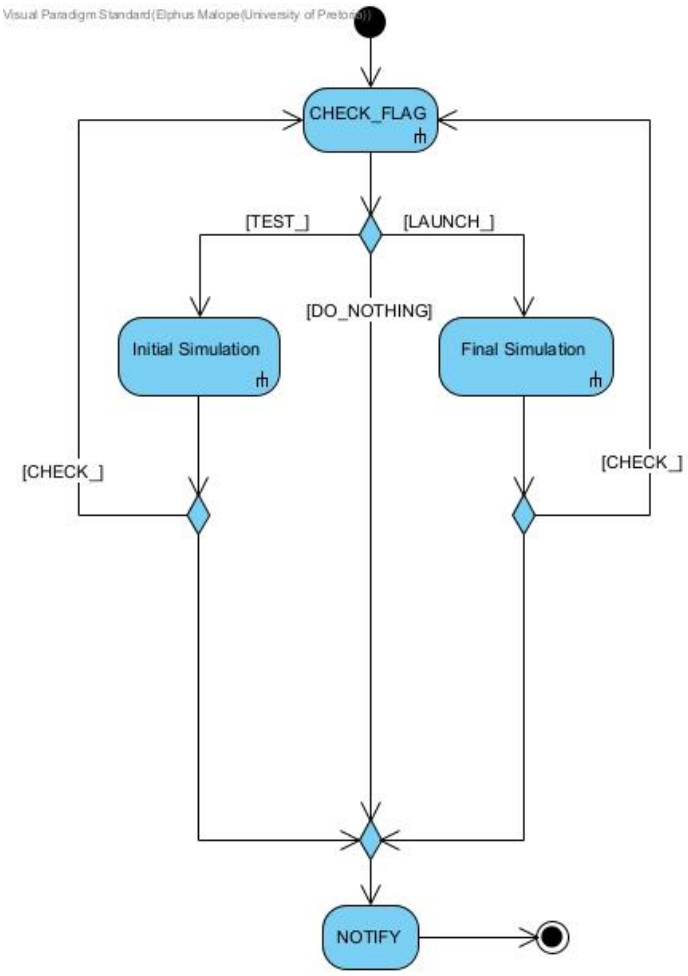
2. Unit Testing (NOT CMAKE)
    a) Engage All three Falcon 9 cores.
    b) Engage the Merlin Vacuum Engine(ON - STATIC POWER).
    c) If all Engines are at optimum, Notify system, and -
    d) Proceed to launching.

II. Final Simulation - **Actual Launch**

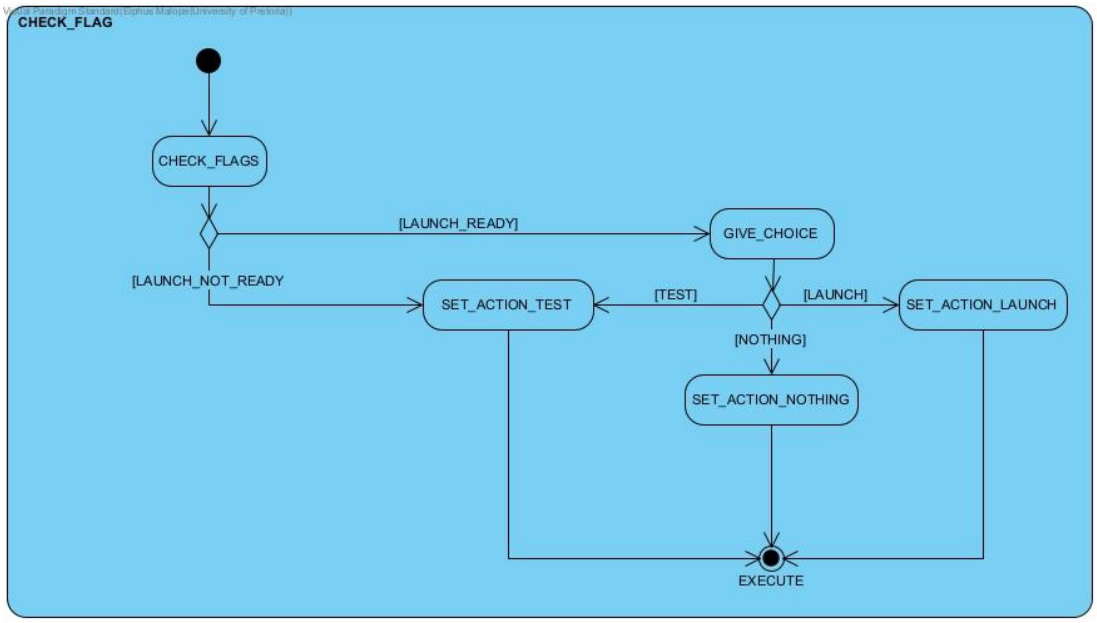
A. Falcon 9
    1. Launch
        a) Engage 1st Stage Engines
            - Initiate Engine_Start (Change state from OFF to ON)
            - Ignite the engines (STATIC POWER)
        b) Change Power_Intake from STATIC POWER to _VAR_POWER to initiate lift off
        c) Wait for signal from system to lift-off
        d) Lift off
    2. 1st Stage Separation
        a) Completely separate the stage from the whole unit
        b) Notify 2nd stage
        c) Execute return_instructions
        d) Initiate land_sequence & execute
        e) Notify system after successful landing procedure
    3. 2nd Stage Separation
        a) Completely separate the stage from the whole unit
        b) Notify Capsule

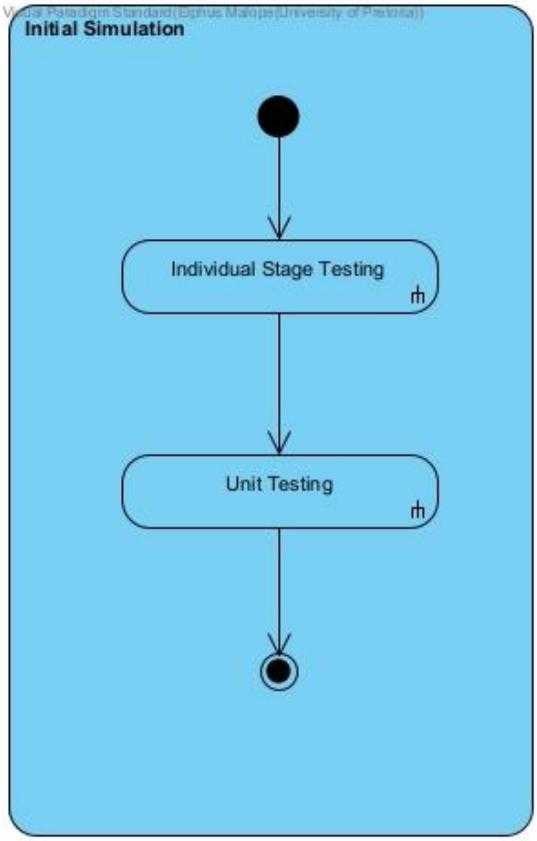
B. Falcon Heavy
    1. NOTE: Falcon Heavy stages have Falcon 9 core and uses the same instructions as Falcon 9 hence this reduces to having the same functional requirements as Falcon 9 for the Actual Launch

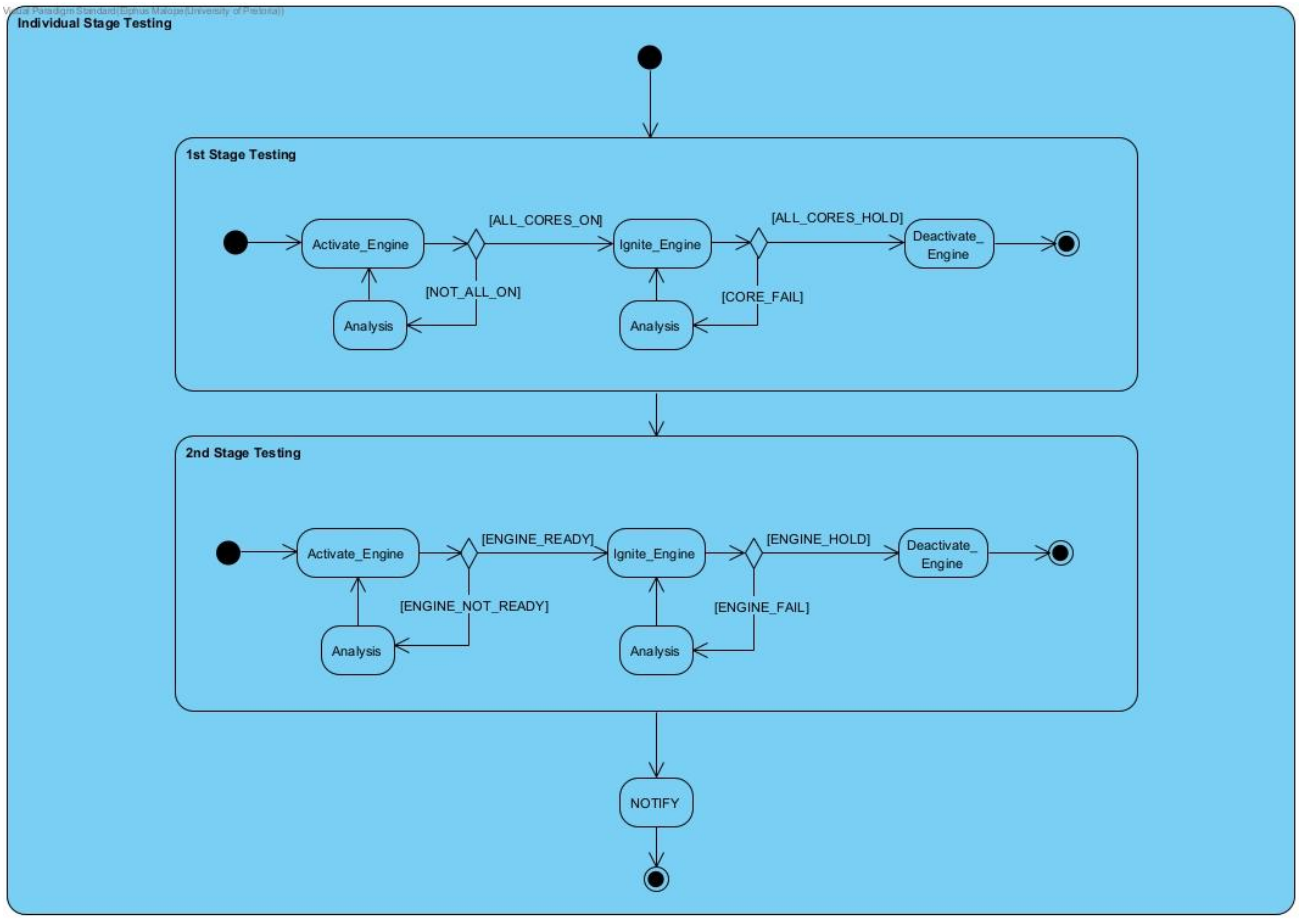
1.2 Activity Diagrams

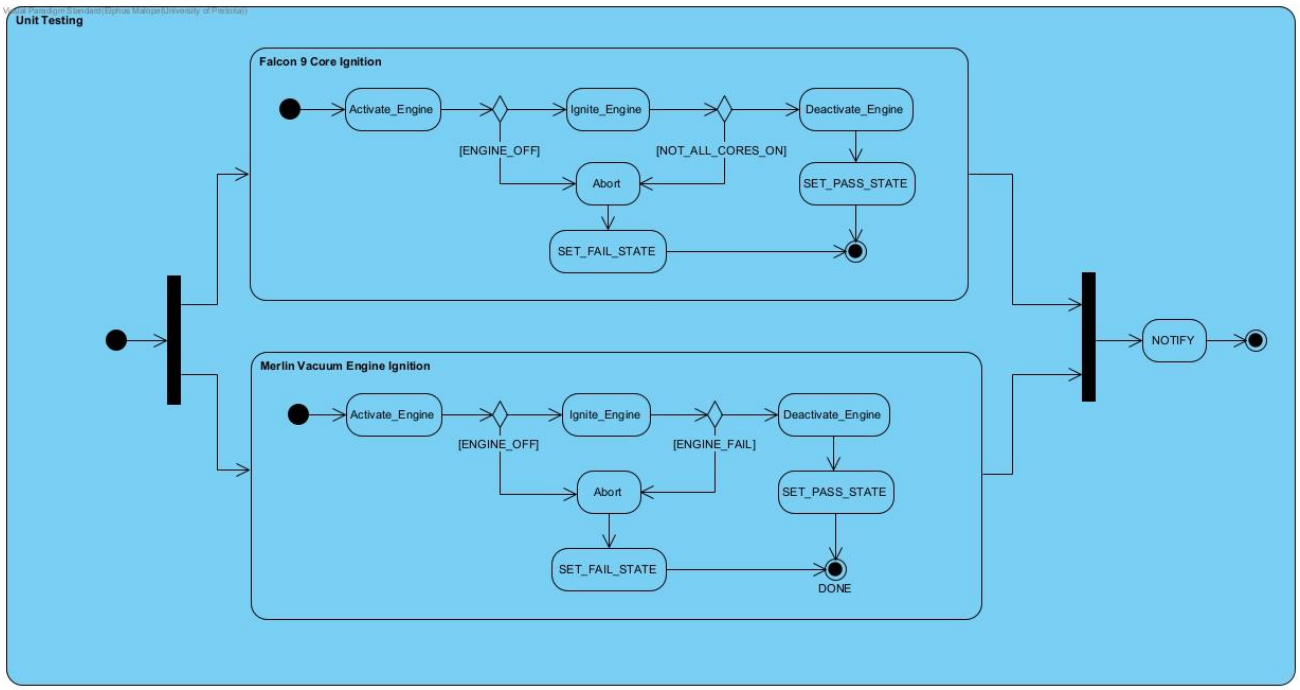


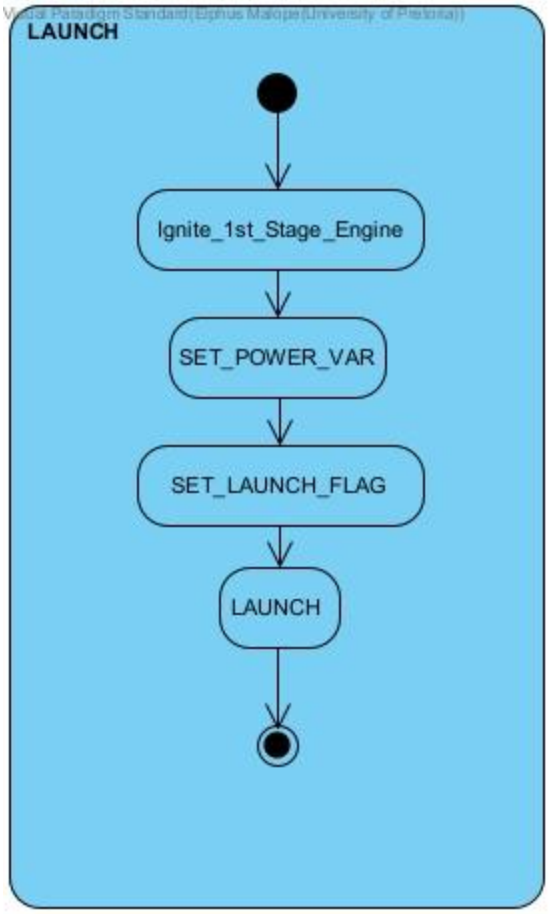
*****The image above is a composite of the activity diagrams below**

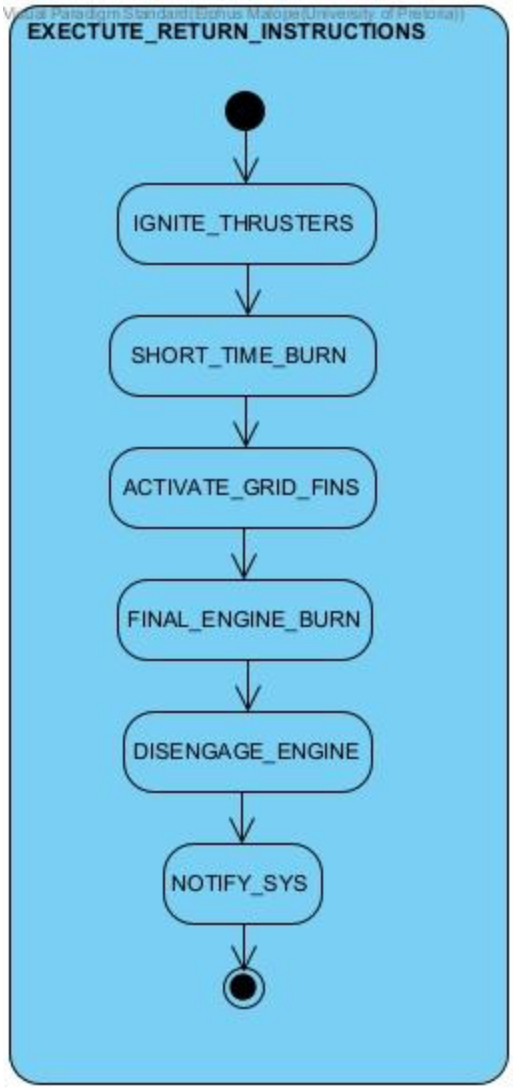


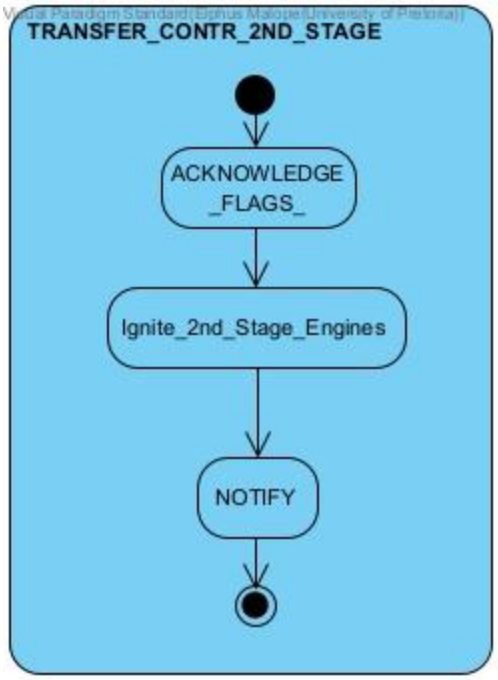


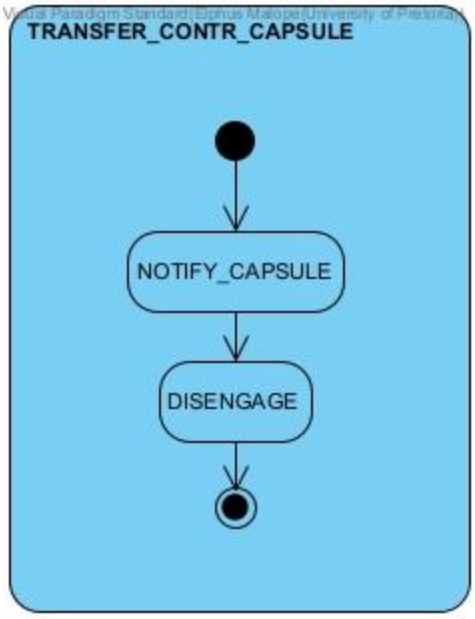

Individual Stage Testing
1st Stage Testing
Activate_Engine
[ALL_CORES_ON]
Ignite_Engine
[ALL_CORES_HOLD]
Deactivate_ Engine
[NOT_ALL_ON]
Analysis
[CORE_FAIL]
Analysis
2nd Stage Testing
Activate_Engine
[ENGINE_READY]
Ignite_Engine
[ENGINE_HOLD]
Deactivate_ Engine
[ENGINE_NOT_READY]
Analysis
[ENGINE_FAIL]
Analysis
NOTIFY


Unit Testing
Falcon 9 Core Ignition
Activate_Engine
Ignite_Engine
Deactivate_Engine
[ENGINE_OFF]
[NOT_ALL_CORES_ON]
Abort
SET_PASS_STATE
SET_FAIL_STATE
Merlin Vacuum Engine Ignition
Activate_Engine
Ignite_Engine
Deactivate_Engine
[ENGINE_OFF]
[ENGINE_FAIL]
Abort
SET_PASS_STATE
SET_FAIL_STATE
DONE
NOTIFY


LAUNCH
Ignite_1st_Stage_Engine
SET_POWER_VAR
SET_LAUNCH_FLAG
LAUNCH


EXECTUTE_RETURN_INSTRUCTIONS
IGNITE_THRUSTERS
SHORT_TIME_BURN
ACTIVATE_GRID_FINS
FINAL_ENGINE_BURN
DISENGAGE_ENGINE
NOTIFY_SYS


TRANSFER_CONTR_2ND_STAGE
ACKNOWLEDGE
FLAGS
Ignite_2nd_Stage_Engines
NOTIFY


TRANSFER_CONTR_CAPSULE
NOTIFY_CAPSULE
DISENGAGE

1.3 Design Patterns

Template Method
Factory Method
State Pattern
Builder Pattern
Composite Pattern

1.4 Classes and Design Patterns

Builder:

Director: RocketBuilderDirector
Builder: RocketBuilder
ConcreteBuilder: FalconNineBuilder, FalconHeavyBuilder

Template Method:

AbstractClass: Engine
ConcreteClass: MerlinEngine, VacuumEngine

Factory Method:

Creator: Engine, Stage, Rocket
ConcreteCreator: MerlinCreator, VacuumCreator, StageOneCreator, StageTwoCreator, InterStageCreator, FalconHeavyCreator, FalconNineCreator

Composite:

Component: Engine
Composite: RocketBooster
Leaf: MerlinEngine, VacuumEngine

State:

First

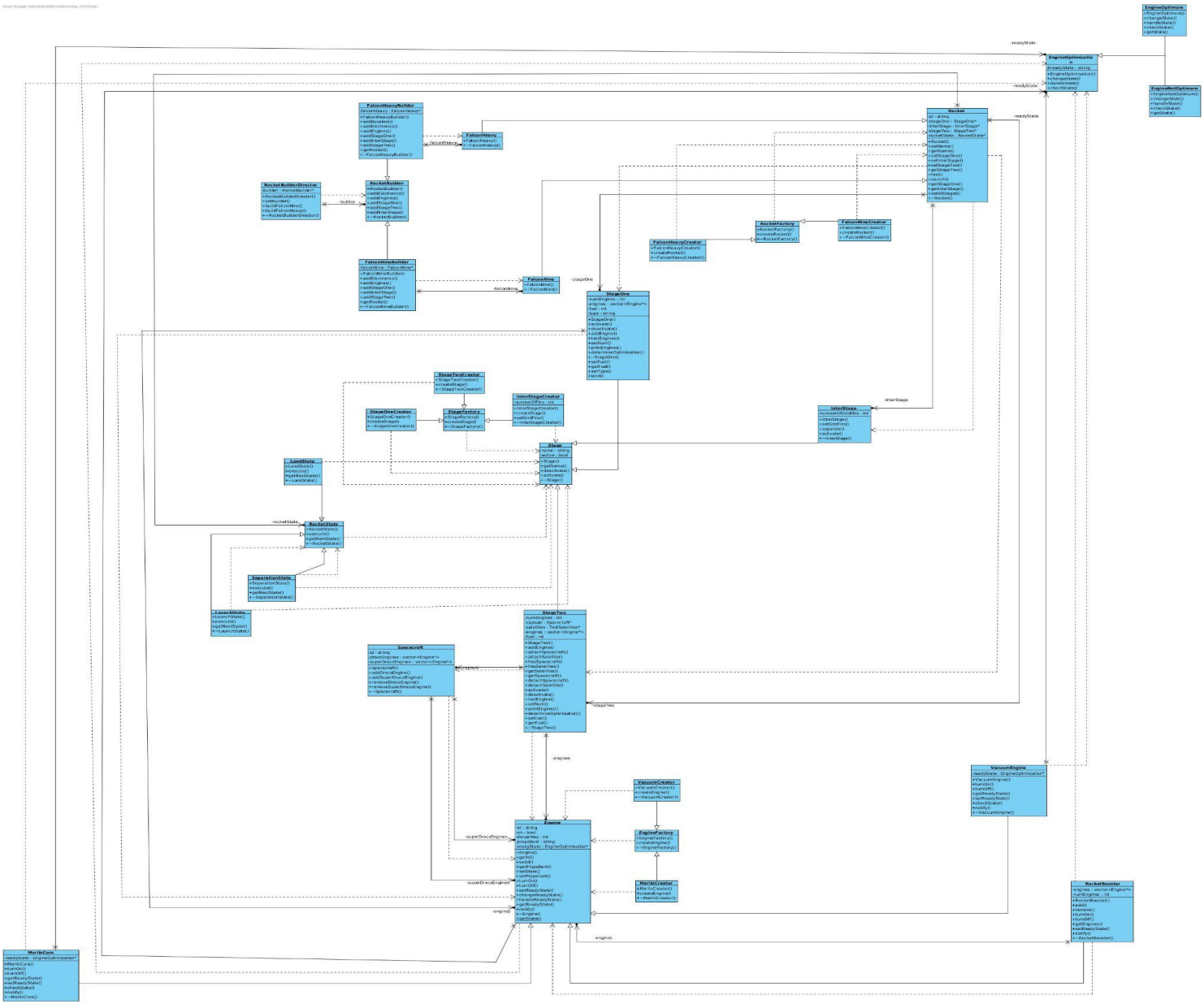
Context: Rocket
State: RocketState
ConcreteStates: LaunchState, SeparationState, LandingState

Second

Context: Engine
State: EngineOptimisation

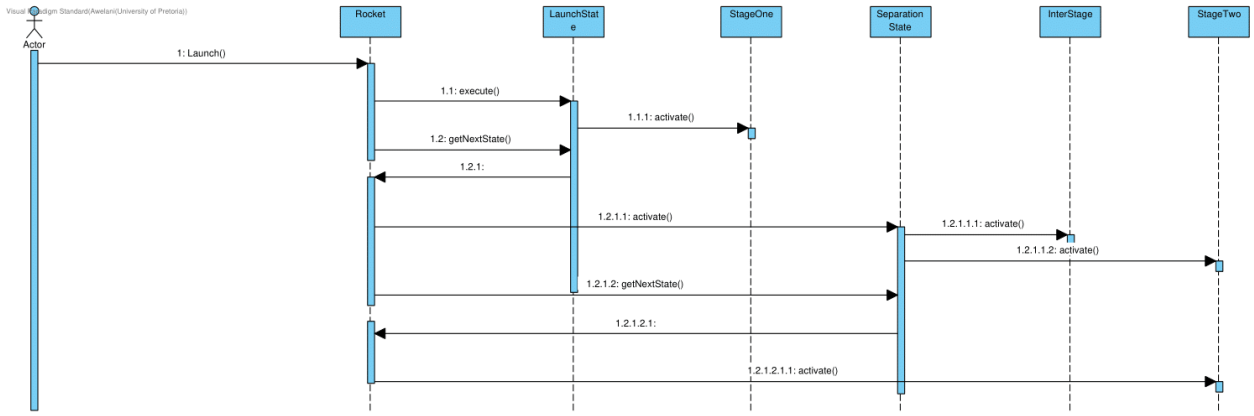
ConcreteStates: EngineOptimum, EngineNotOptimum

1.5 Class Diagram

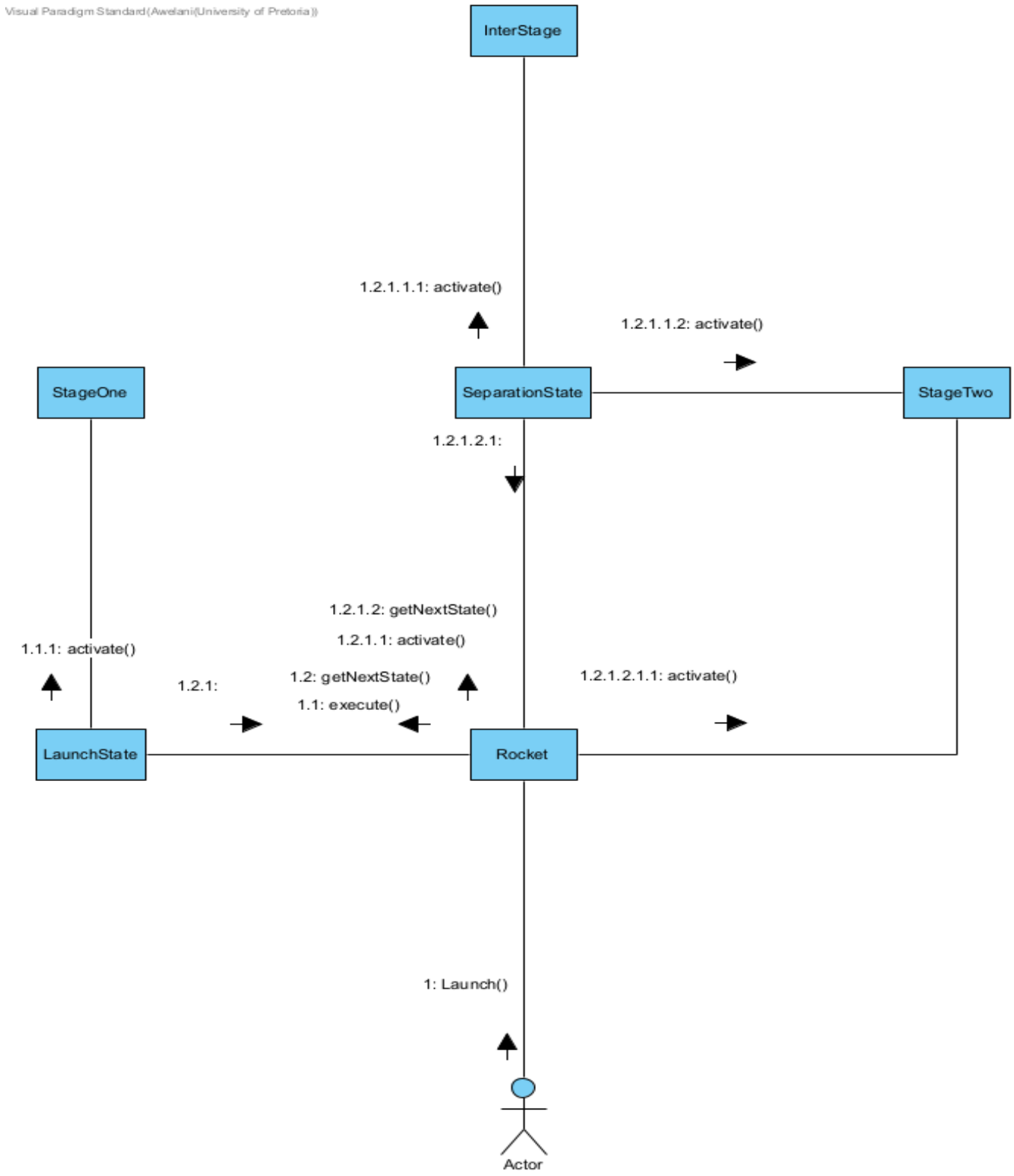


1.6 Sequence and Communication Diagrams

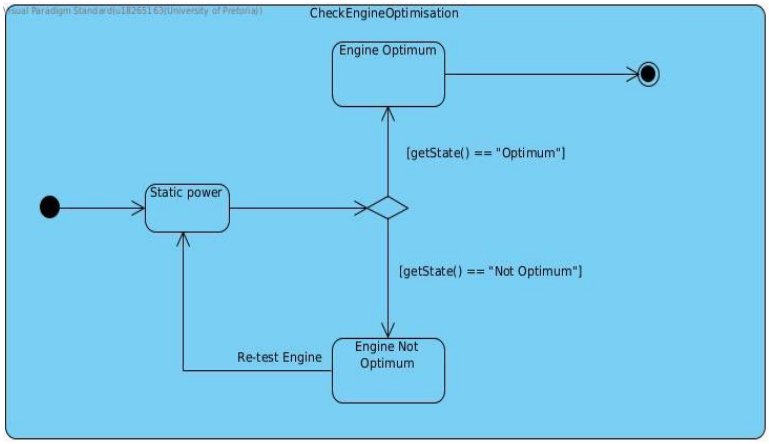
Sequence diagram

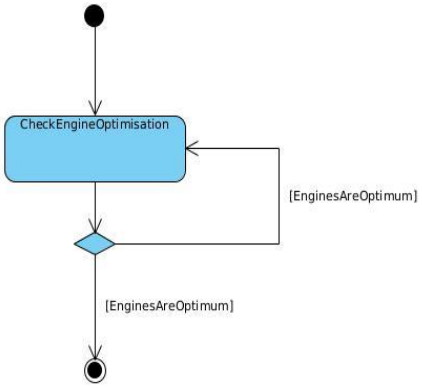


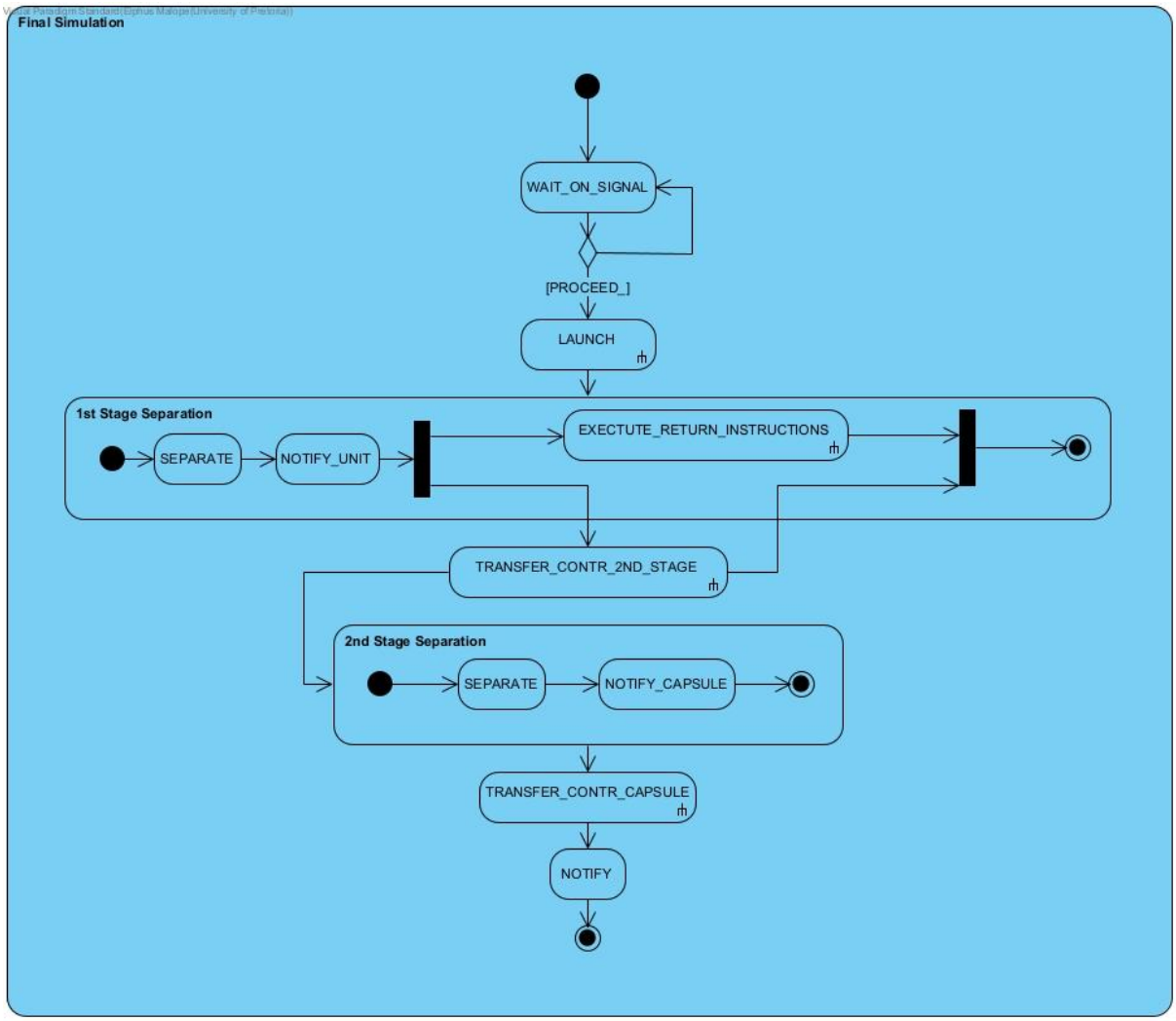
Communication diagram


Visual Paradigm Standard(Awelani(University of Pretoria))
InterStage
1.2.1.1.1: activate()
1.2.1.1.2: activate()
StageOne
SeparationState
StageTwo
1.2.1.2.1:
1.2.1.2: getNextState()
1.2.1.1: activate()
1.1.1: activate()
1.2.1:
1.2: getNextState()
1.1: execute()
1.2.1.2.1.1: activate()
LaunchState
Rocket
1: Launch()
Actor

1.7 State Diagrams

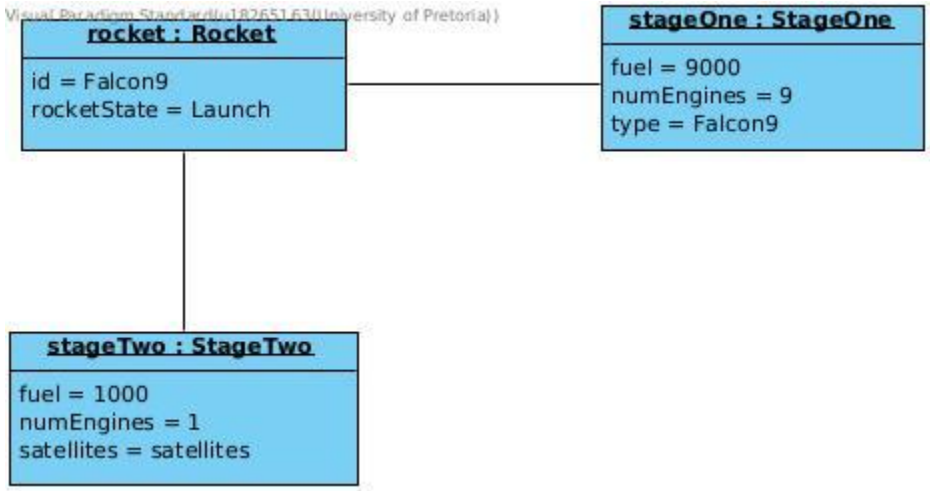




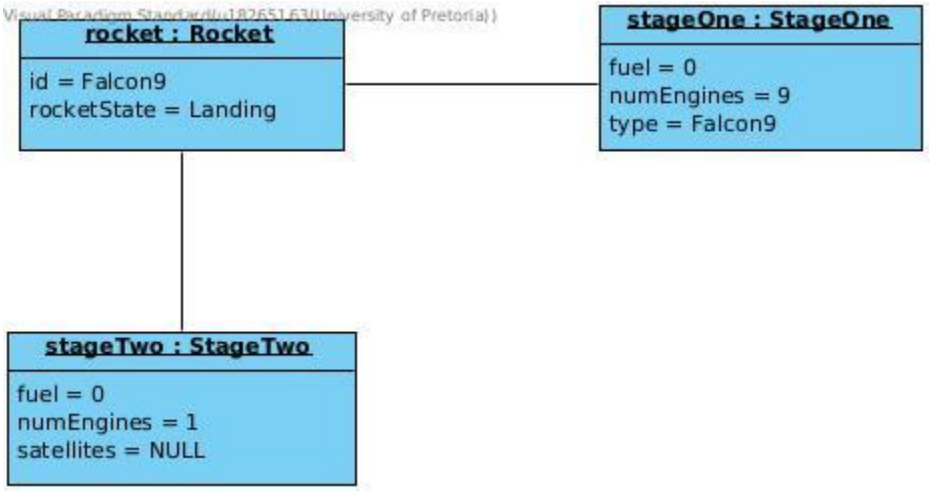


1.8 Object Diagram

Before Launch:



When Docked:



The Dragon SpaceCraft

Task 1: Design

1.1 Functional Requirements:

Crew Dragon
Thrusters:

create 1 thruster
make clones

Parachutes:

same as Thruster

Simulation:

instantiate the crew dragon spacecraft
attach parachutes
attach thrusters
dock passengers
load cargo
save the spacecraft

Pass Check:

check weight of passengers and cargo, if over 6000 kg when going to orbit simulation fails.
check parachute number if # less than 4, simulation fails.
If Mark2 Parachute is used check if the total weight carried is greater than 5510 kg, if true simulation fails.
If draco thrusters is used check if the total weight carried is greater than 5510 kg, if true simulation fails.
If Crew dragon is used check if passengers on board are greater than 7 ,if true Simulation failed.
Times reused have to be less than 5 or else the spacecraft fails.
More than 1440 pounds-force to be produced by overall thrusters attached to the spacecraft.

Mission:

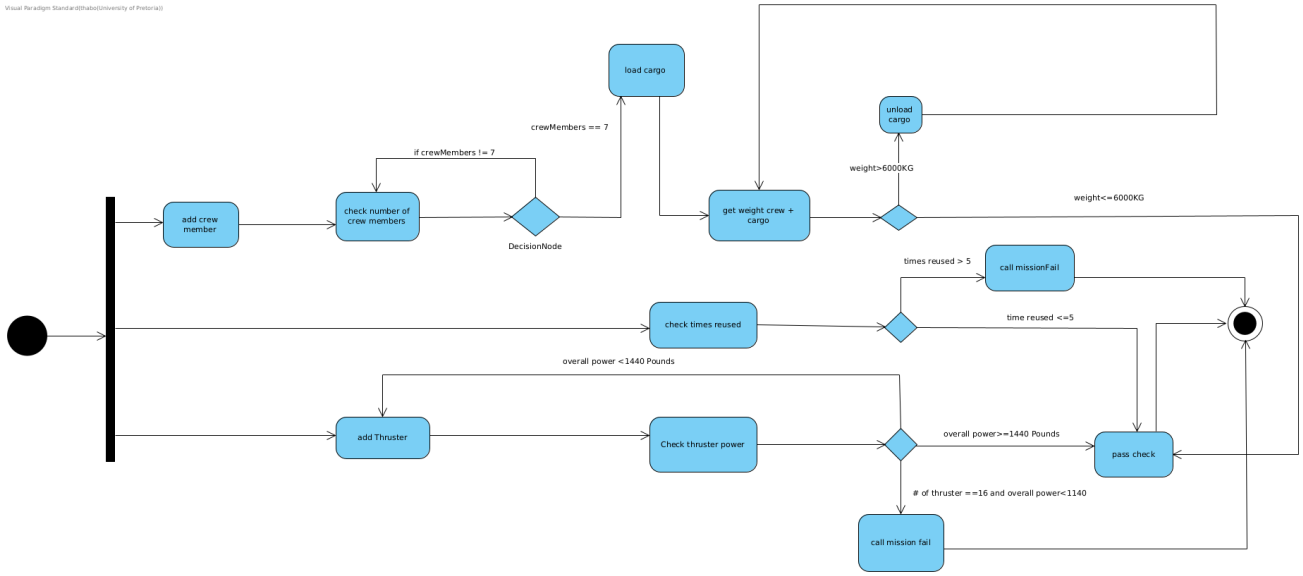
set destination(ISS)
dock spacecraft to ISS(International Space Station)
unload cargo/ passengers
setDestination(earth)

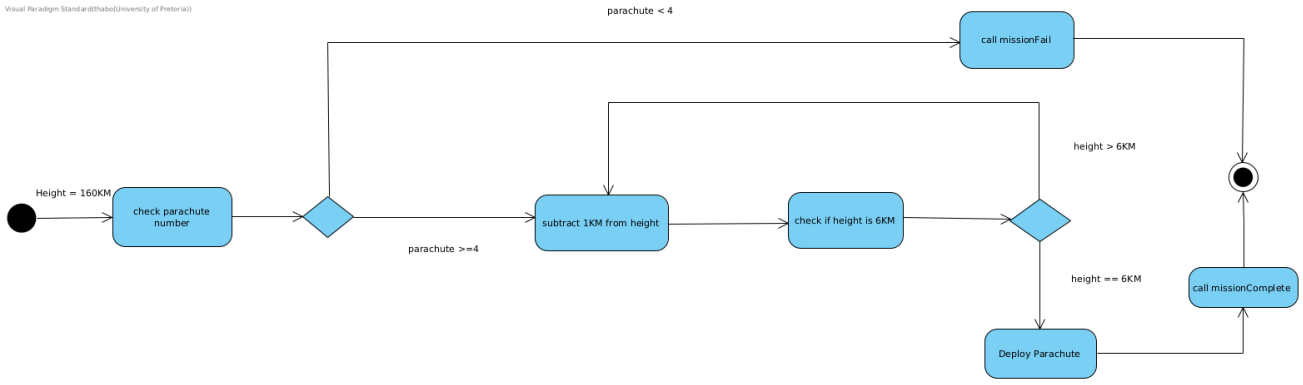
undock spacecraft from the ISS
monitor height(once reach earth monitor from 160KM)
deploy parachutes at approximately height of 6KM
retuned safely MISSION COMPLETE (simulation Passed)
Decrement number of times spacecraft can be used

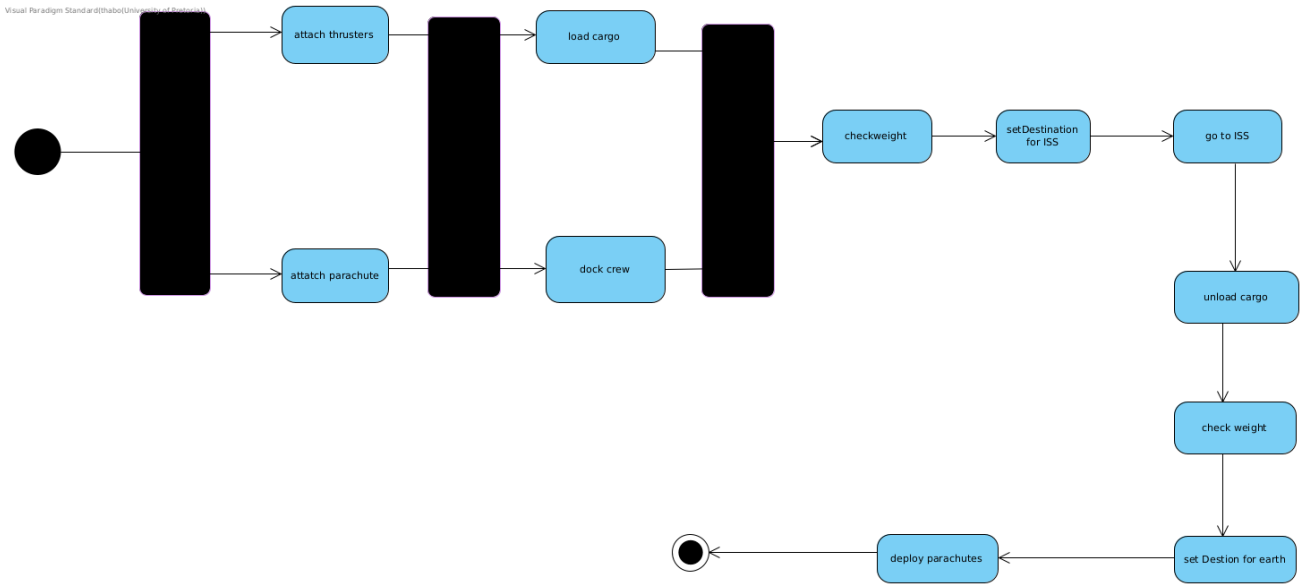
Dragon Spacecraft

same procedure as Crew Dragon with exception of the following:
No seats(no stationing of passengers)

1.2 Activity Diagrams







1.3 Design Patterns

1.3.1 Observer Pattern:

Observing the height of the Crew Dragon / Dragon for parachute deployment.

1.3.2 Factory Method:

Producing the thrusters and the parachutes

1.4 Classes And Design Patterns

Observer

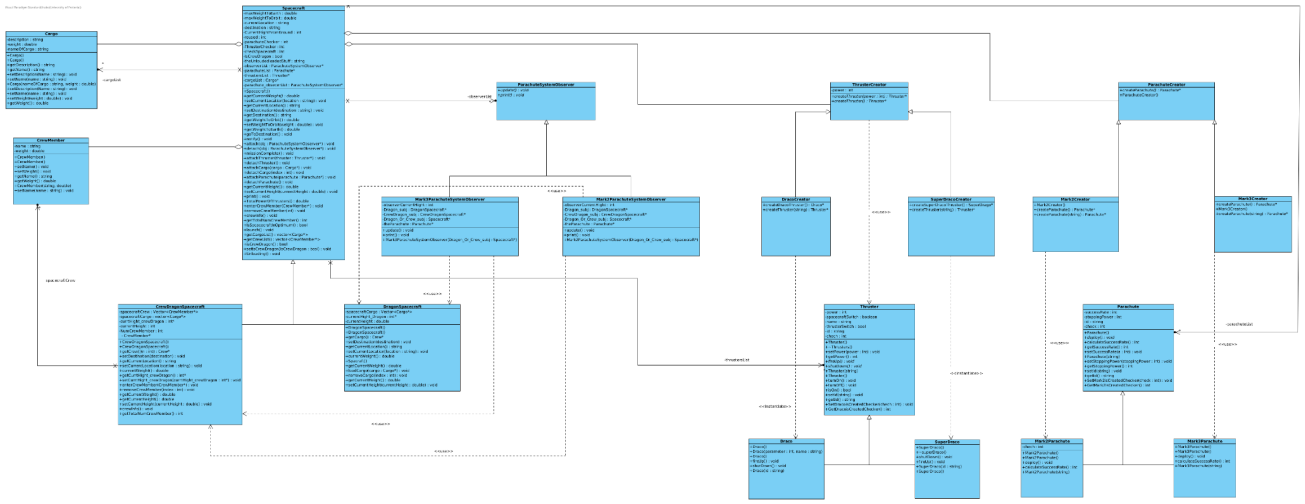
Subject: spaceCraft
concreteSubject: crewDragon/Dragon spacecraft
Observer: ParachuteSystemObserver
concreteObserver:Mark2ParachuteSystemObserver
concreteObserver:Mark3ParachuteSystemObserver

Factory Method

creator: ThrusterCreator
concreteCreator: DracoCreator
concreteCreator:SuperDracoCreator
concreteProduct:DracoThruster
concreteProduct:SuperDracoThrusters

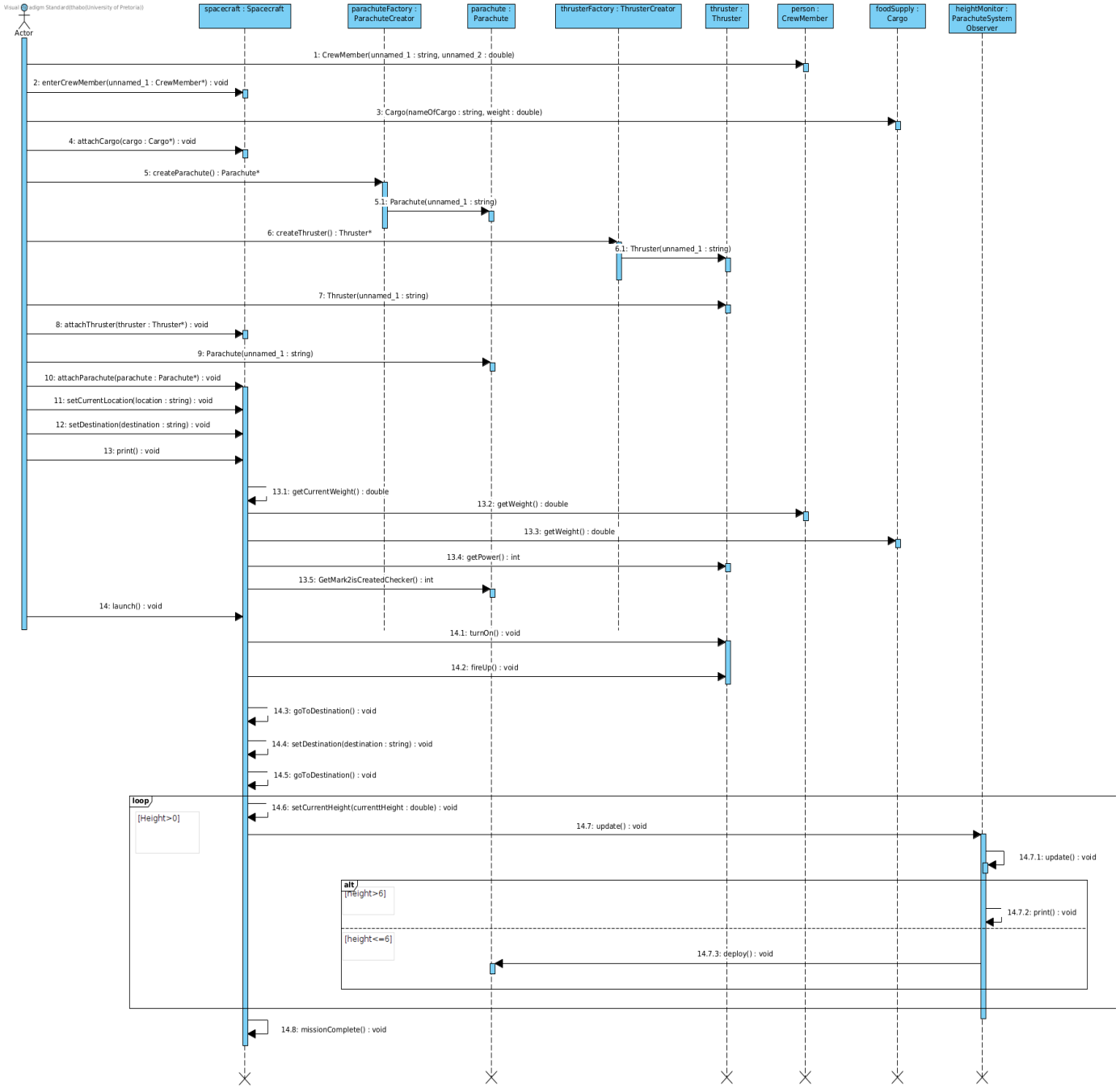
Creator: ParachuteCreator
concreteCreator: Mark2Creator
concreteCreator: Mark3Creator
concreteProduct: Mark2Parachute
concreteProduct: Mark3Parachute

1.5 Class diagram

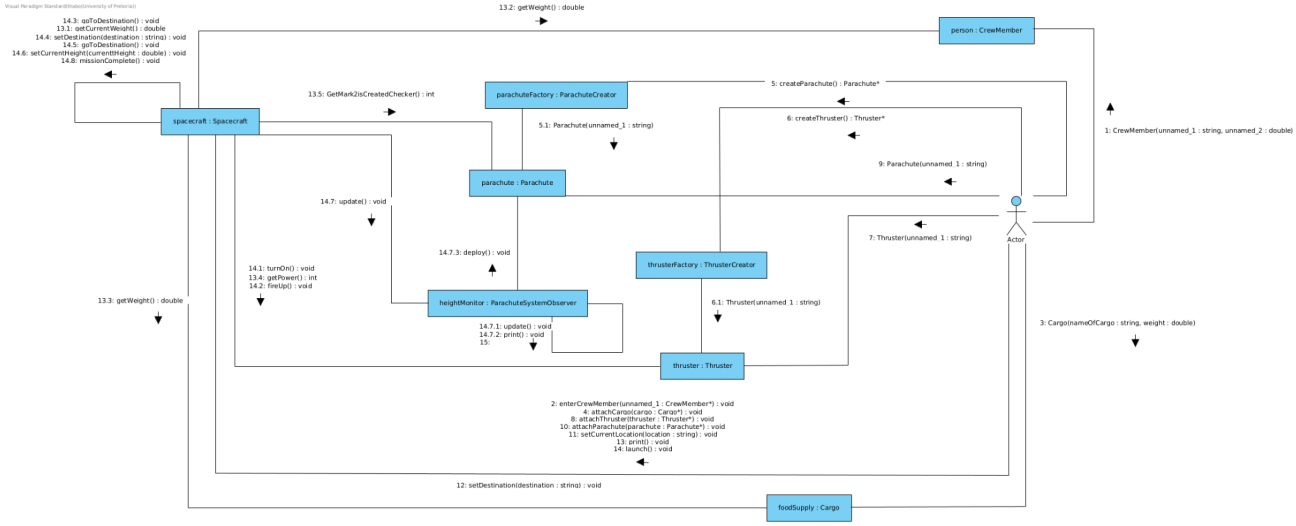


1.6 Sequence and Communication Diagram

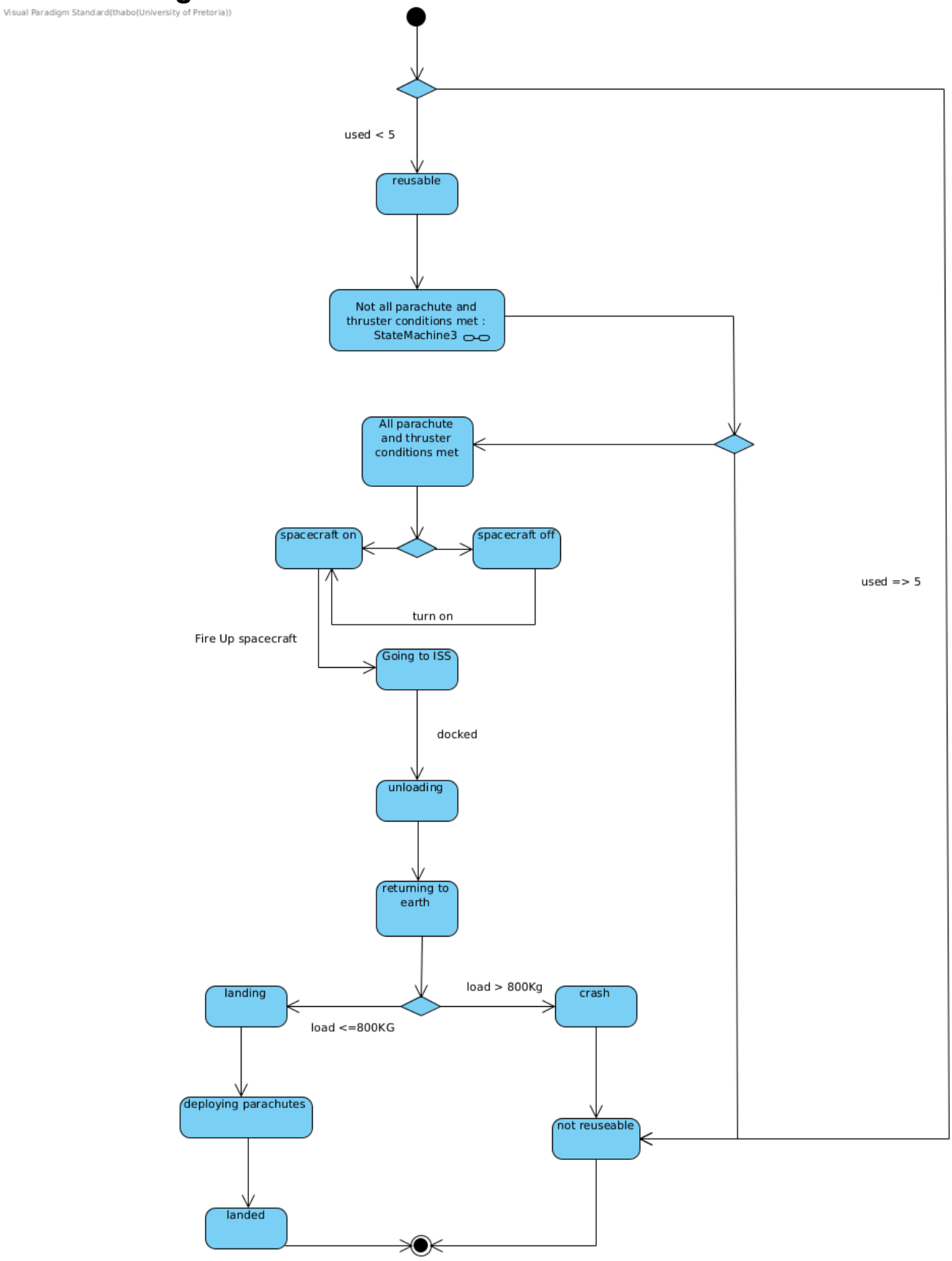
1.6.1 Sequence Diagram

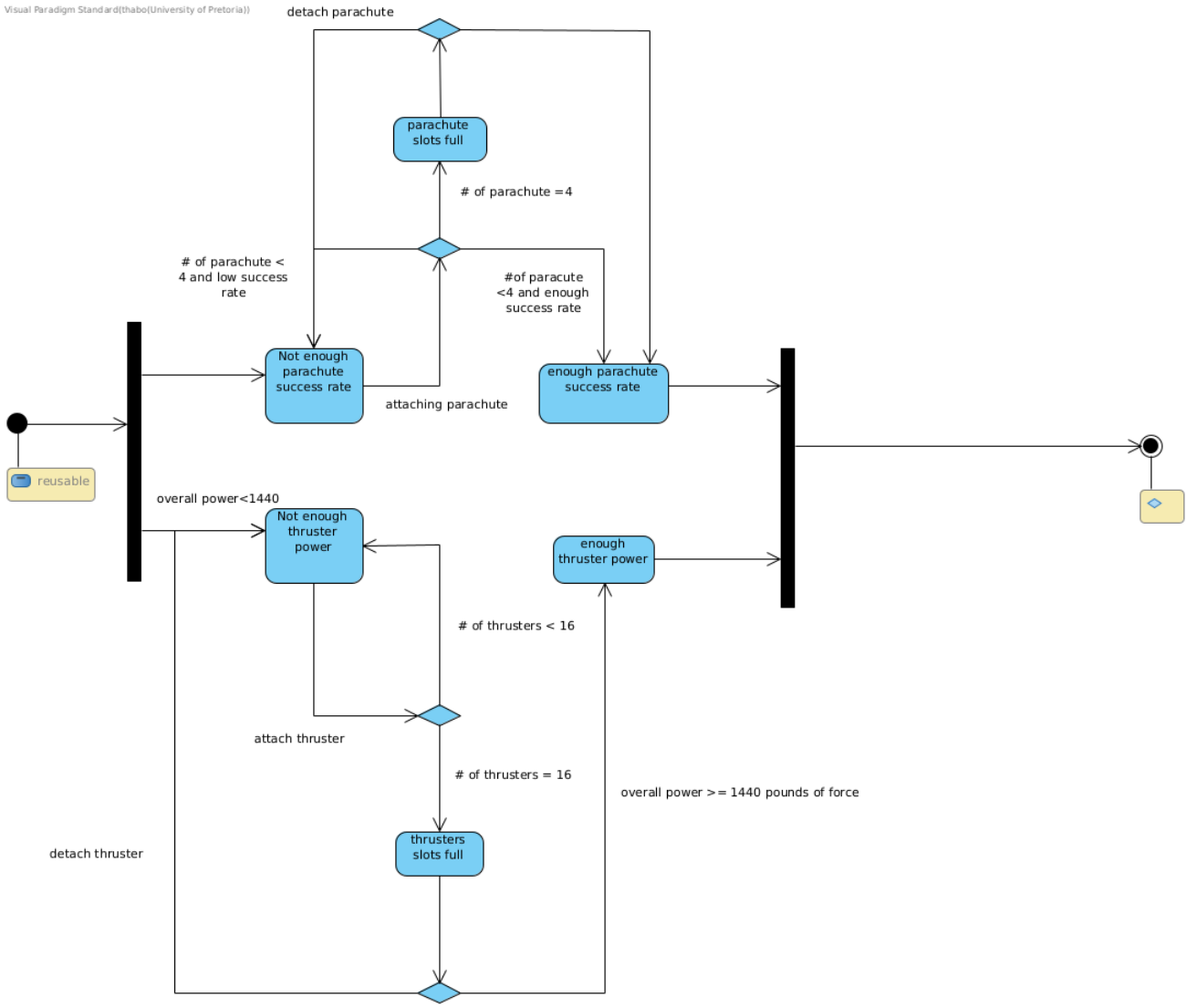


1.6.2 Communication diagram



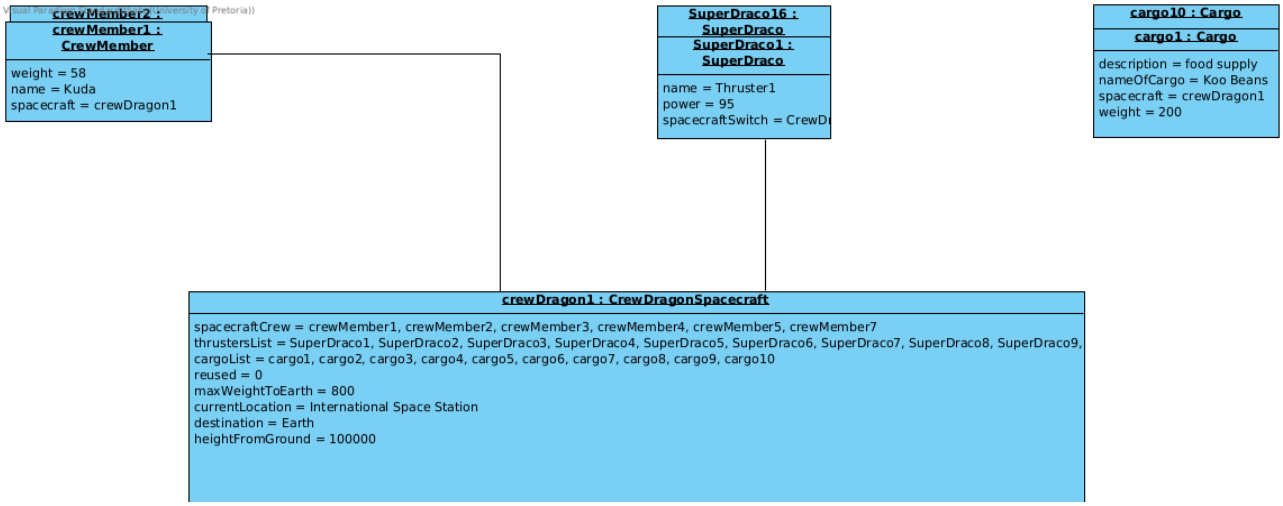
1.7 State Diagram



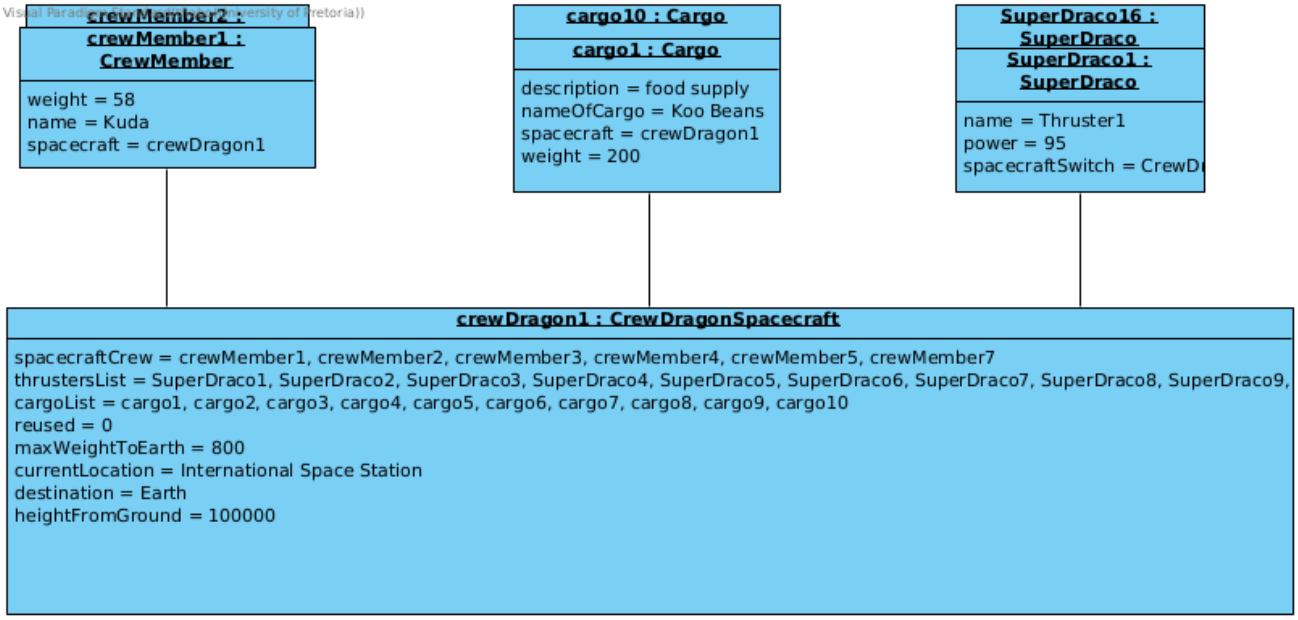

Visual Paradigm Standard(thabo(University of Pretoria))
detach parachute
parachute slots full
# of parachute =4
# of parachute < 4 and low success rate
#of paracute <4 and enough success rate
Not enough parachute success rate
attaching parachute
enough parachute success rate
reusable
overall power<1440
Not enough thruster power
enough thruster power
# of thrusters < 16
attach thruster
# of thrusters = 16
overall power >= 1440 pounds of force
detach thruster
thrusters slots full

1.8 Object Diagram

On earth



On the ISS



Starlink Satellites

Task 1: Design

1.1 Functional requirements:

StarLink Satellites

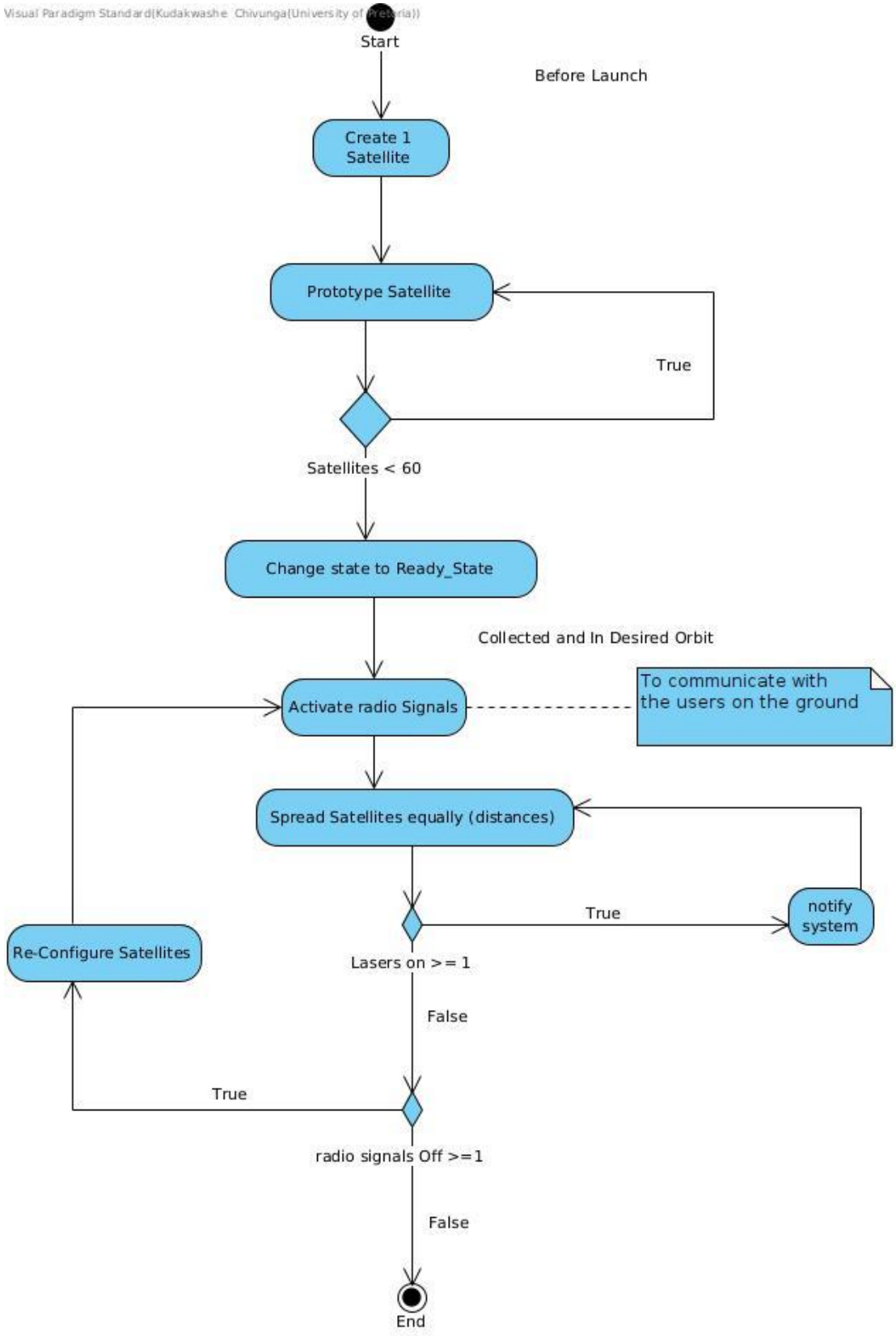
A. Before Launch
   1. Creation of StarLink Satellites
      - Creation of 1 Satellite
      - Create 60 clones of the 1 Satellite.
      - All 60 Satellites' radio signals and lasers used for communication should not be activated / turned to default value “Off”.
      - When all Satellites have been created, set state to Ready_State

   2. Capsule Engines
      - Subtract Satellites from the storage facilities.
      - Load them in Clusters of up to **60** Satellites at a time in a Falcon 9 Rocket
      - When all Satellites have been launched, notify System/ “loaders”

B. In Orbit - After Launch
   1. Registration of Satellites
      a. Once in orbit, all 60 Satellites should cover the entire orbit equally by:
         - Keeping the same distance of 0.1 km from all its other neighbouring satellites.
         - Making use of its laser states (On/Off) to notify each other of any movements from the original position
         - If atleast one laser is On/detected, All Satellites should be notified and re-adjusted.
         - OR failure to function correctly

      b. All 60 Satellites should enable communication with the Antenna (users on the ground)

1.2 Activity Diagram



1.3 Design Patterns

- **Prototype** - for “Duplicating” 1 Satellites to 60.
- **Iterator** - Iterates through the list of Satellites to enable changes in Satellites
- **Mediator** - Allows communication between Satellites, using the On/Off states of lasers.
- **Observer** - Allows Antena (users from the ground) read and act upon radio signals sent from the satellites.

1.4 Classes and their interrelationships

Prototype:

Prototype - Satellite
concretePrototype - ConcreteSatellite
Client - SatellitesMaker

Iterator

Iterator - Satellite
Aggregate - SatelliteMaker
concreteAggregate - N/A
concreteIterator - concreteSatellite

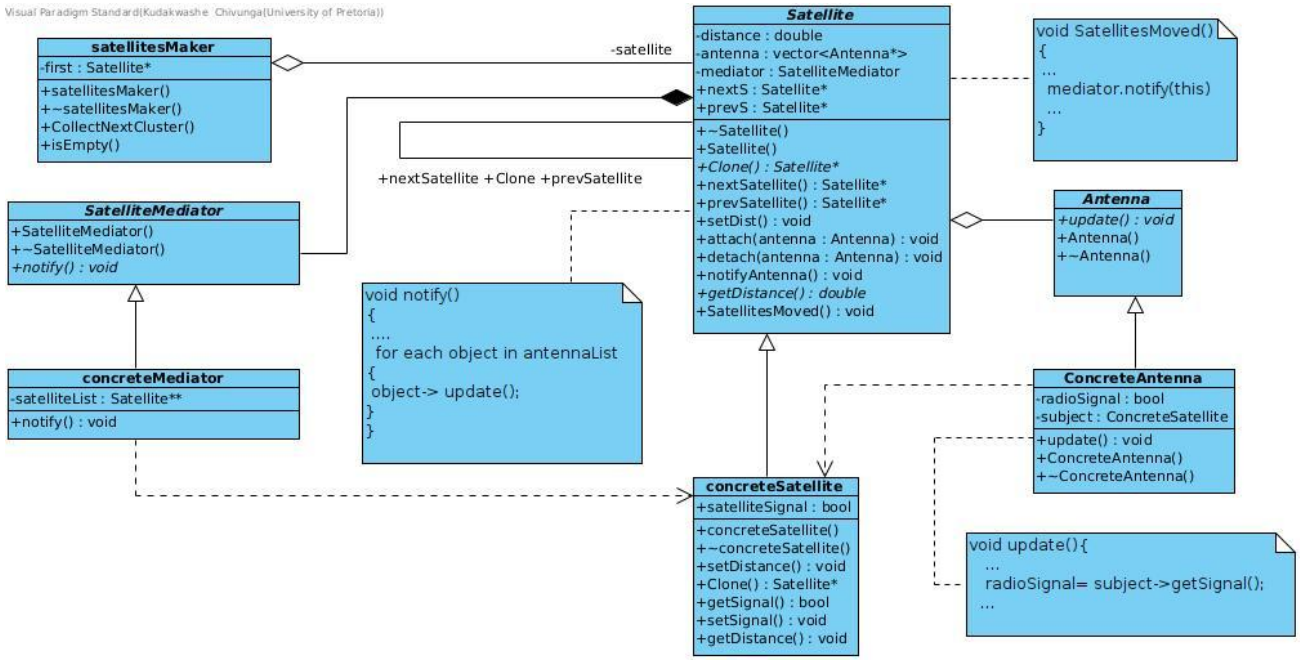
Mediator:

Mediator - SatelliteMediator
Colleague - Satellite
ConcreteColleague - ConcreteSatellite
ConcreteMediator - ConcreteSateliteMediator

Observer:

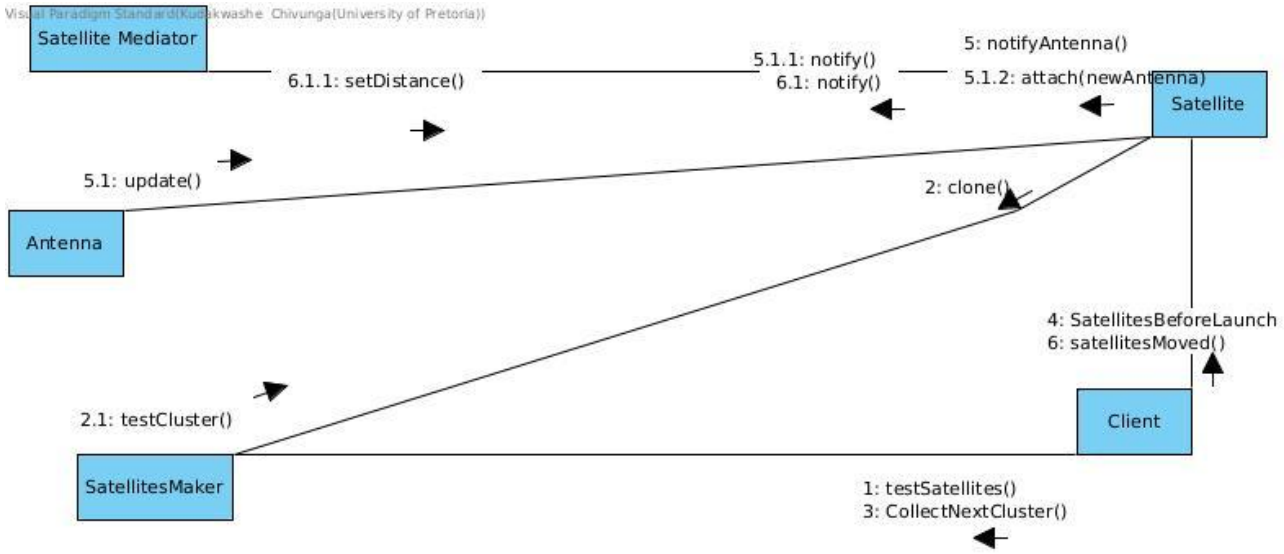
Observer - Antenna
ConcreteObserver - ConcreteAntenna
Subject - Satellite
ConcreteSubject - ConcreteSatellite

1.5 Class Diagram

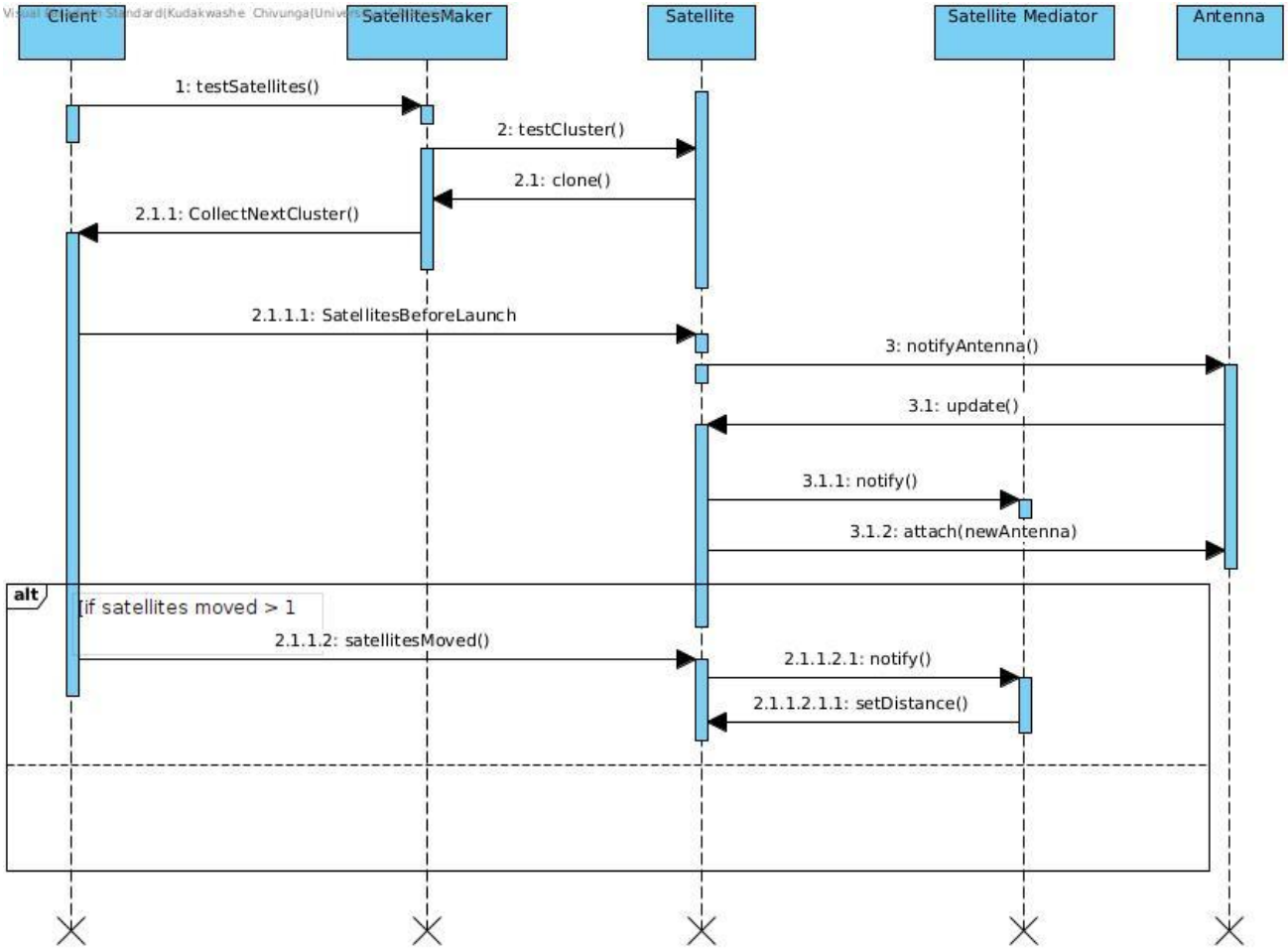


1.6 Sequence and Communication Diagrams

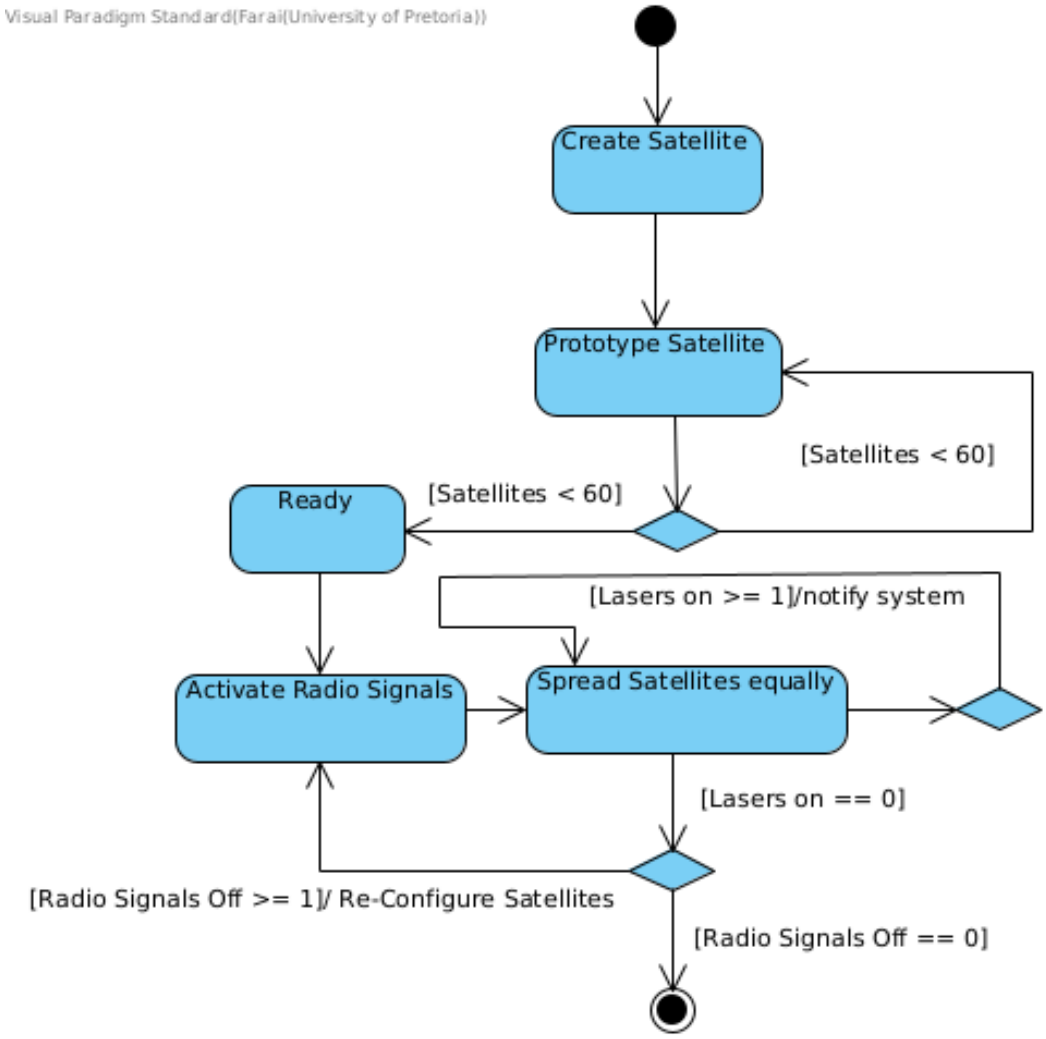
Communication diagram



Sequence diagram

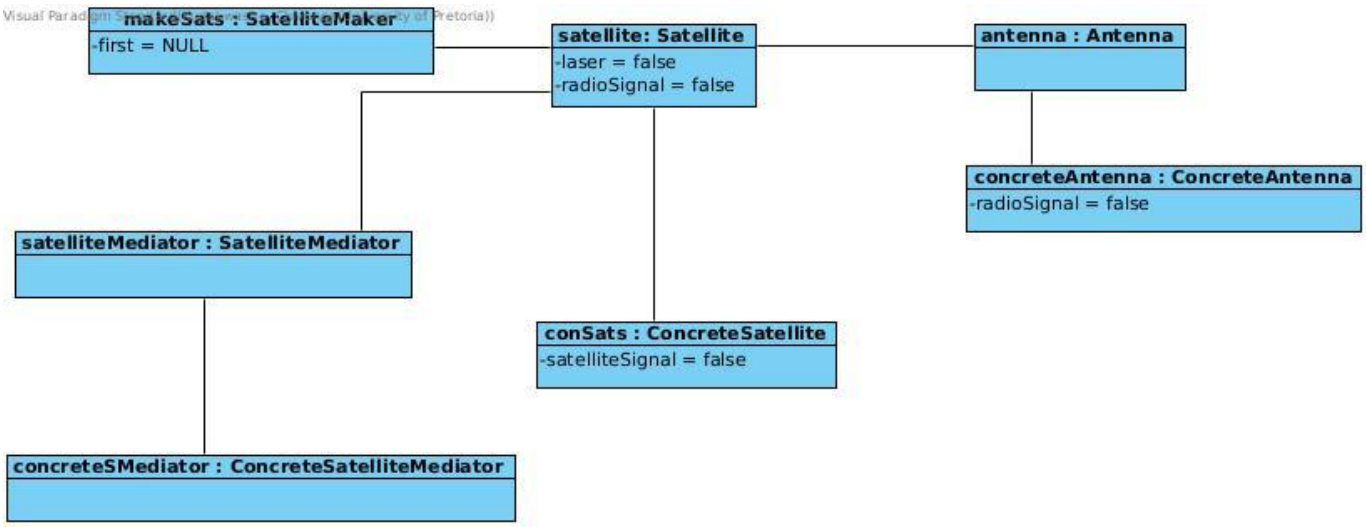


1.7 State Diagram

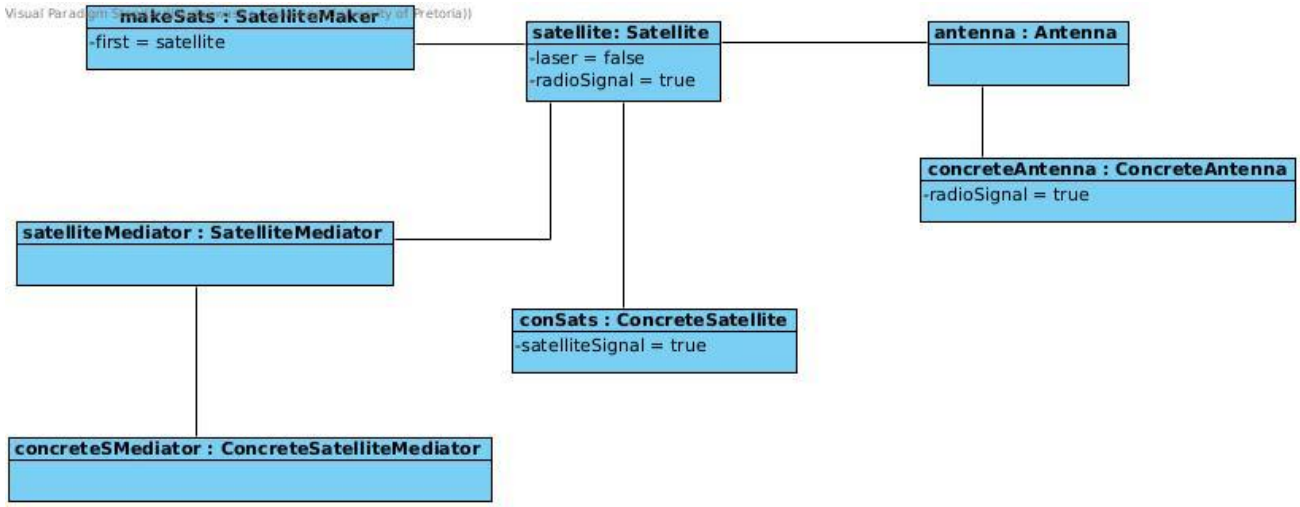

Visual Paradigm Standard(Farai(University of Pretoria))
Create Satellite
Prototype Satellite
[Satellites < 60]
Ready
[Satellites < 60]
[Lasers on >= 1]/notify system
Activate Radio Signals
Spread Satellites equally
[Lasers on == 0]
[Radio Signals Off >= 1]/ Re-Configure Satellites
[Radio Signals Off == 0]

1.8 Object Diagram

- **Before Launch**



- ***In desired orbit***



Launch Simulator

Task 1: Design

1.1 Functional requirements:

I. **Two Choices of simulation**

A. Test Mode

a. Tweaking the amount of fuel needed to reach desired destinations. Arrive at desired destination (km) , using bare min fuel.

b. Create combinations of Starlink Satellites, cargo and humans, making sure we don't pass the Rocket's weight threshold.

- Tweaking the number of elements in the spacecraft

c. Make sure thruster power of the rocket is enough to push the Spacecraft in motion(the overall power of thruster has to be over 1440 Pounds of power)
Attach thrusters with different power(SuperDraco has more power than the Draco)

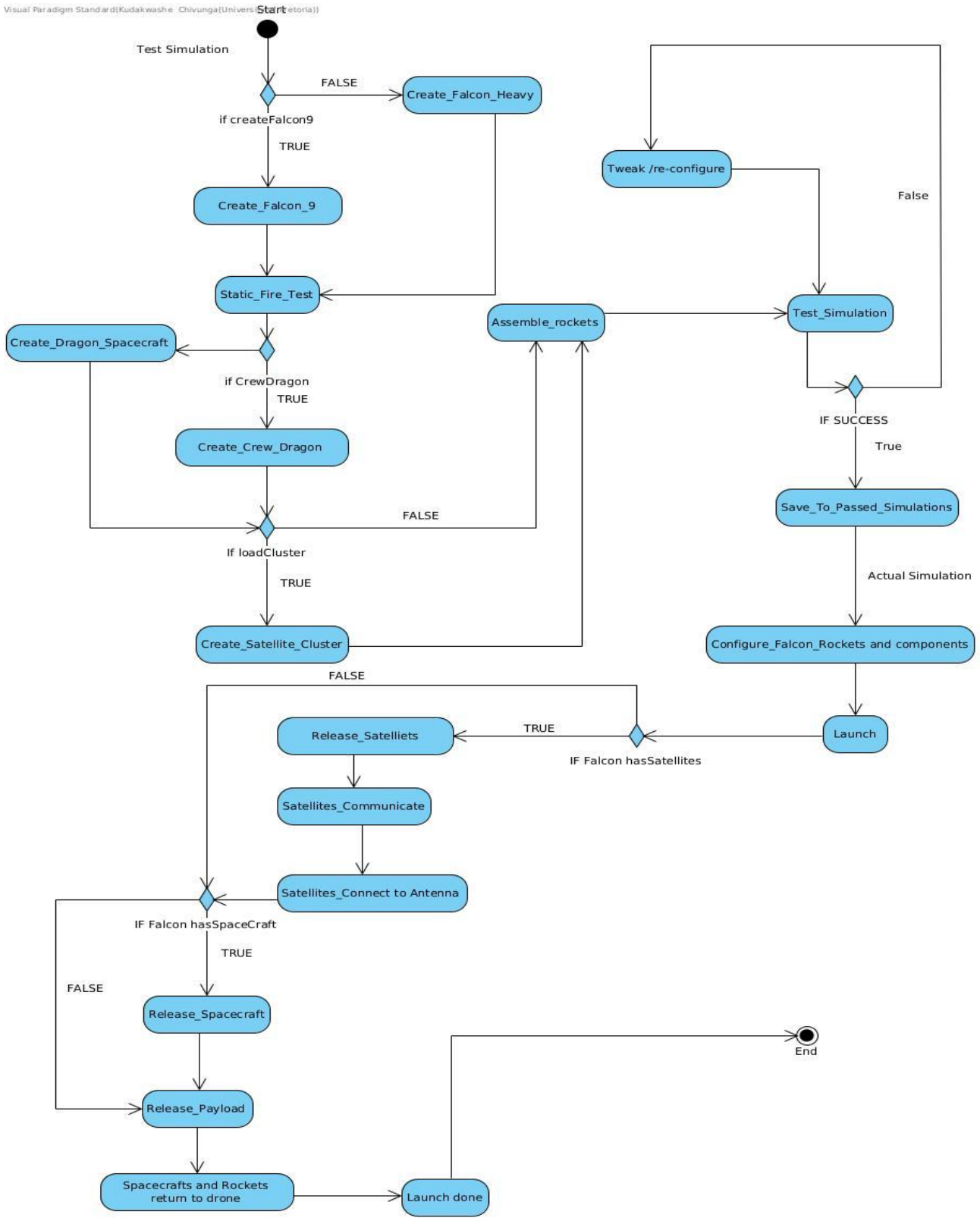
d. If the simulation is optimized /'passed', save it to the actual simulator list.

B. Real/ Actual Simulation

a) Ensure that the rocket in use is at an optimum state and is ready for launch.(Falcon9/Falcon Heavy is optimum or not)

b) Run the simulations that passed "Test Mode".

1.2 Activity Diagram


Start
Test Simulation
FALSE
Create_Falcon_Heavy
if createFalcon9
TRUE
Create_Falcon_9
Static_Fire_Test
Create_Dragon_Spacecraft
if CrewDragon
TRUE
Create_Crew_Dragon
FALSE
If loadCluster
TRUE
Create_Satellite_Cluster
FALSE
Tweak /re-configure
False
Assemble_rockets
Test_Simulation
IF SUCCESS
True
Save_To_Passed_Simulations
Actual Simulation
Configure_Falcon_Rockets and components
Launch
TRUE
IF Falcon hasSatellites
Release_Satelliets
Satellites_Communicate
Satellites_Connect to Antenna
IF Falcon hasSpaceCraft
TRUE
FALSE
Release_Spacecraft
Release_Payload
Spacecrafts and Rockets return to drone
Launch done
End

1.3 Design Patterns

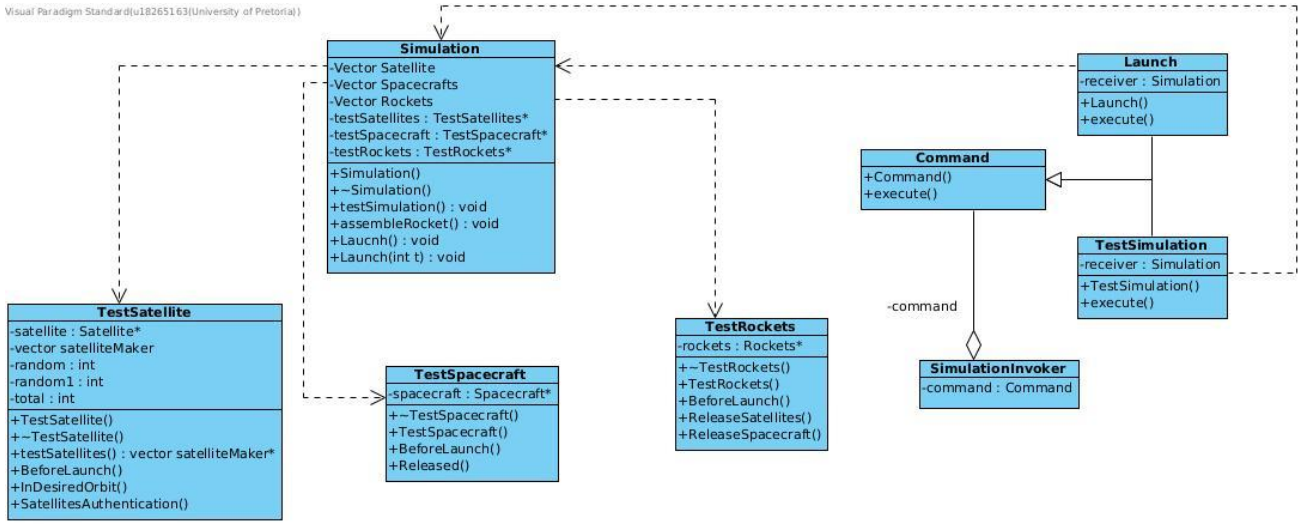
Command - Used as an interface to perform the actions defined by simulation.

1.4 Classes and their interrelationships

Command

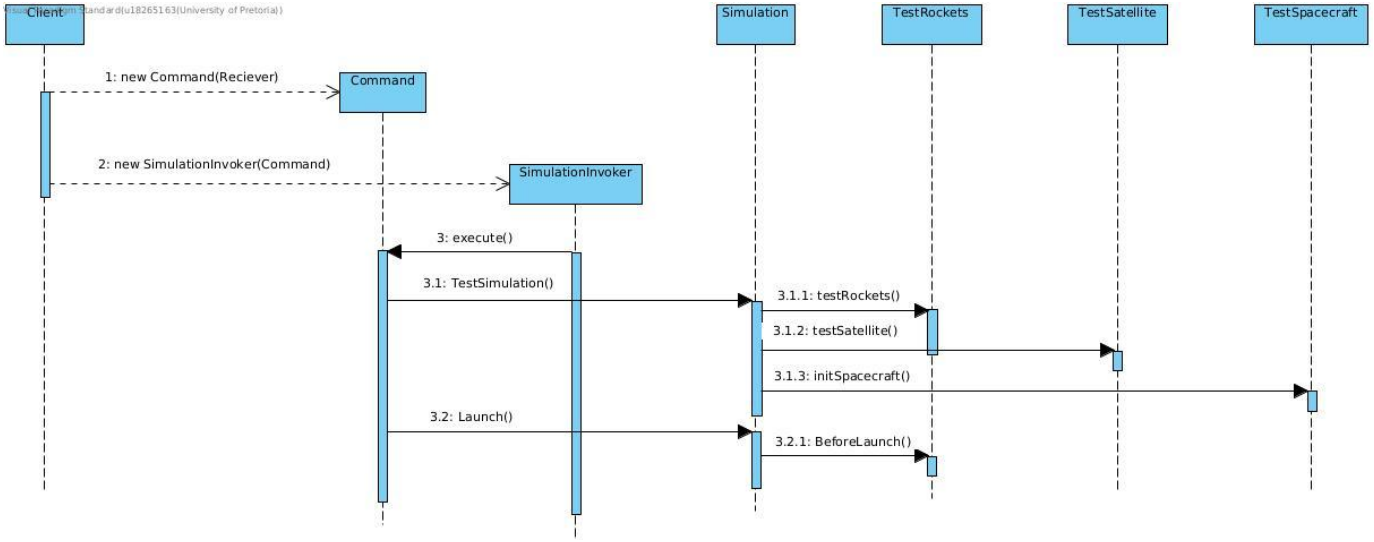
Command - Command
ConcreteCommand - TestSimulation
ConcreteCommand - Launch
Invoker - SimulationInvoker
Receiver - Simulation

1.5 Class Diagram

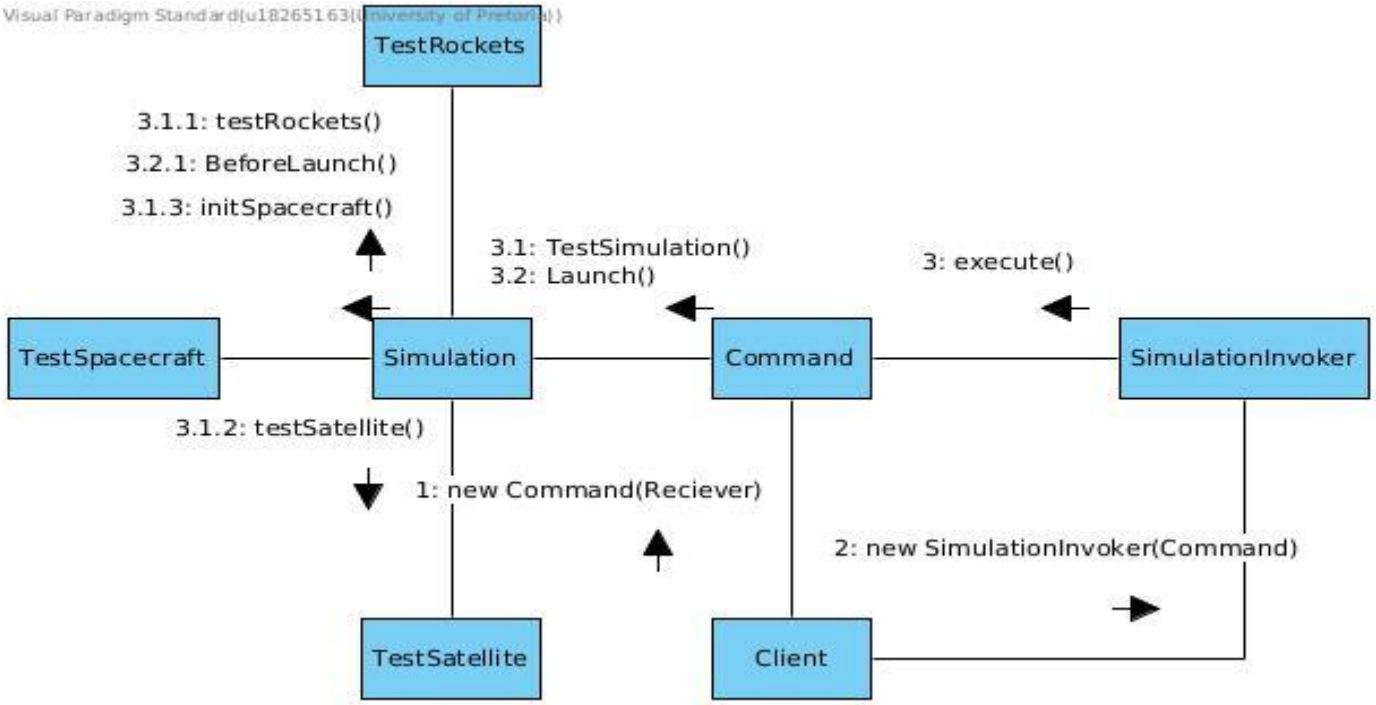


1.6 Sequence and communications Diagram

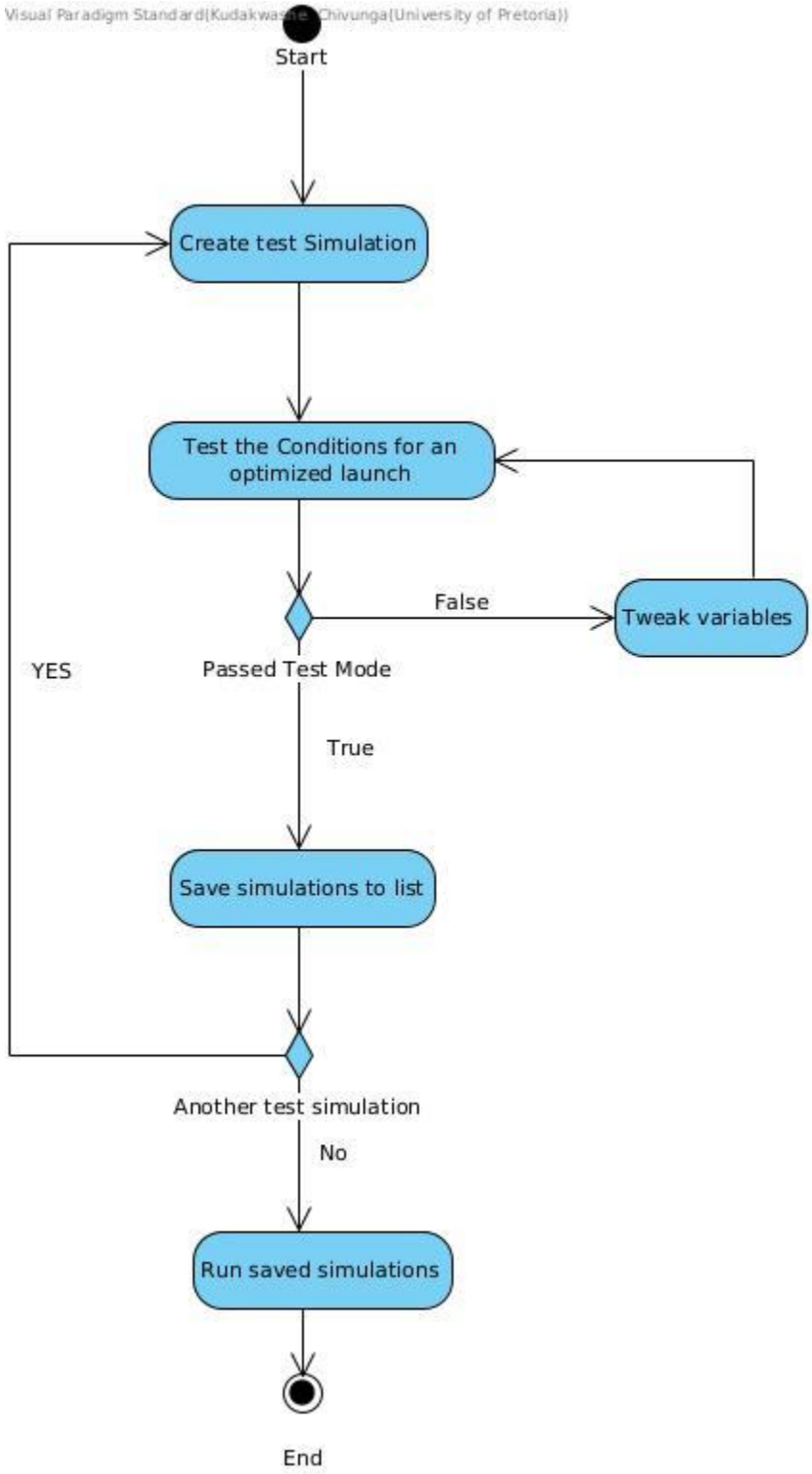
- **Sequence diagram:**



- **Communication diagram:**



1.7 State Diagram



Report

This project is meant to strengthen our knowledge of design patterns. On completion of this project, we should be able to say with conviction that we as a group have completed a relatively large project and have experience of working in a relatively large design and development team. We broke the Spacex and Starlink simulation into 4 subsystems, namely Falcon rockets, Dragon spacecraft, Starlink Satellites and Launch simulator. We combined the first three subsystems in the last subsystem “Launch Simulator”.

Design patterns used:

1. Template
2. Factory
3. State
4. Observer
5. Mediator
6. Prototype
7. Builder
8. Iterator
9. Composite
10. Command

The use of the Design patterns in each subsystem

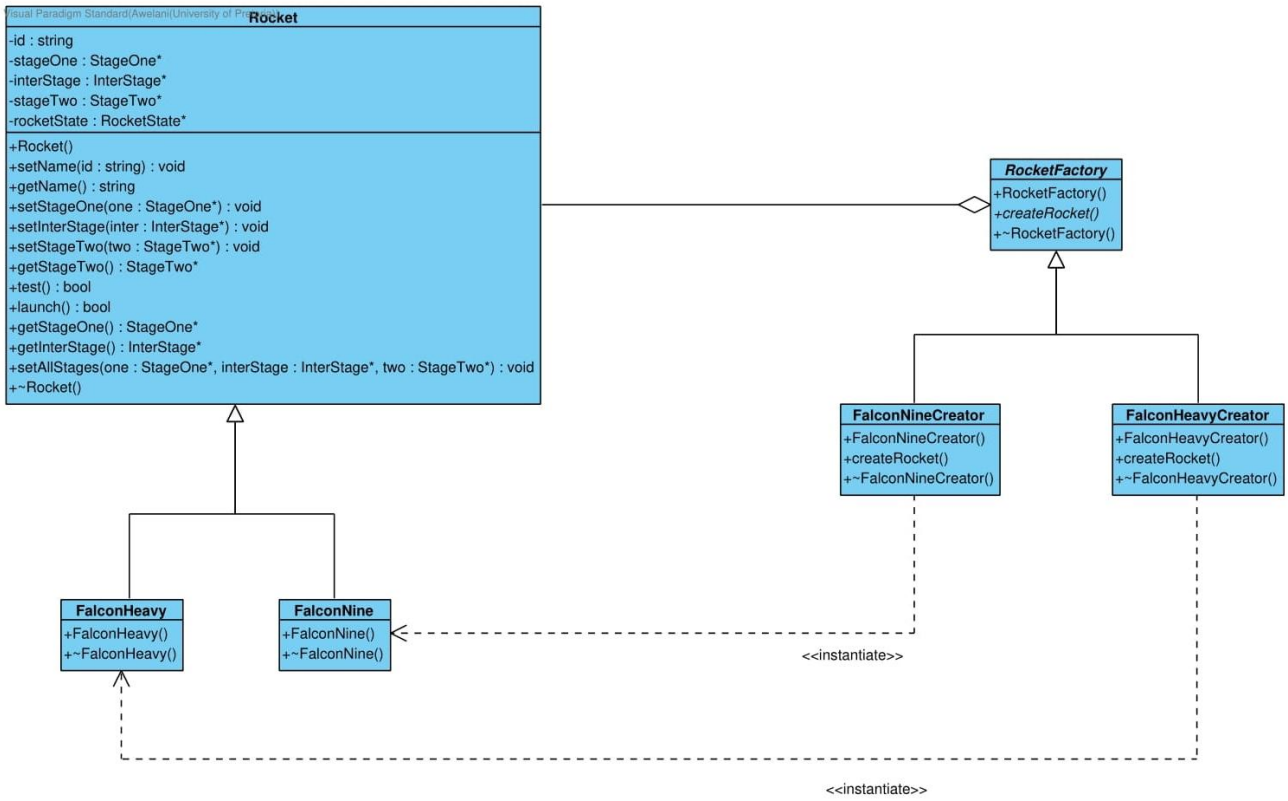
Falcon Rockets:

Template design pattern:

Template method was used for the engines. Since the rockets have different kinds of engines which work slightly in different ways it makes sense to have sub classes that implement the same interface. Primitive methods are turnOn() and turnOff

Factory Design pattern:

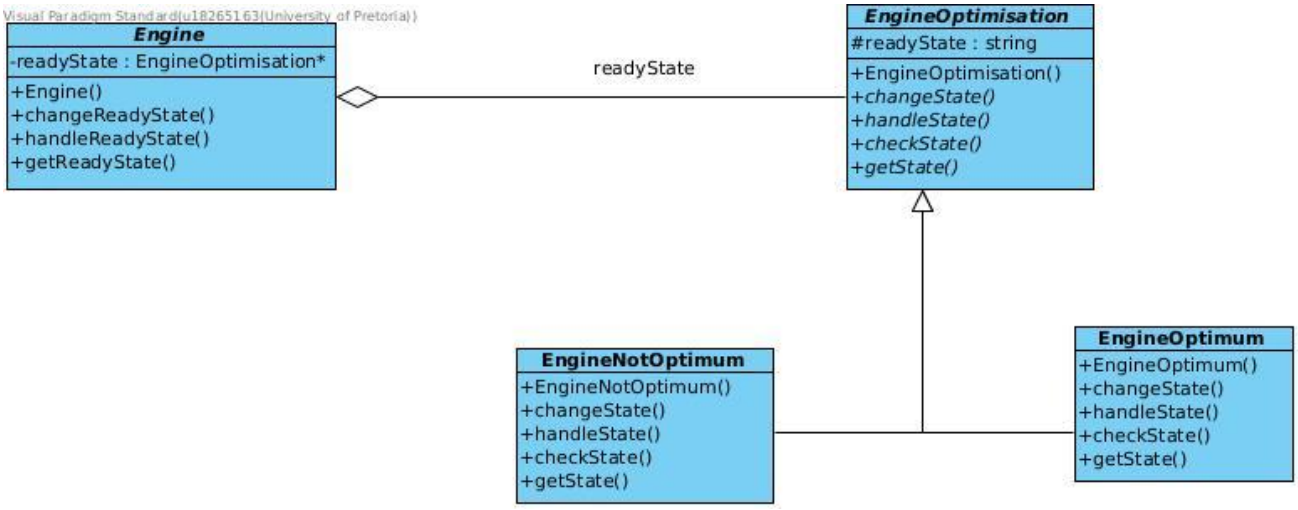
The factory was used in conjunction with the builder, we have multiple factory methods for creating parts of the rocket (i.e Engines, Stages, Rocket Containers/Shells). The builder communicates with the factories then “assembles” the parts step by step.



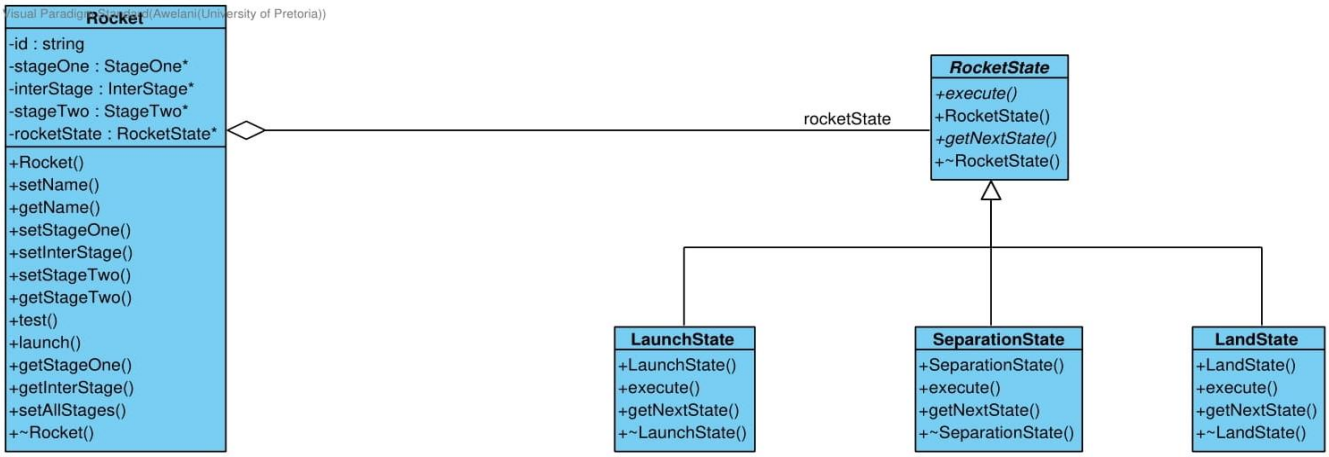
State Design pattern:

We used the State Design pattern twice: For the Engine state and the Rocket state

- **Engine State:**
    - The engine state can be one of two states, “EngineOptimum” and “EngineNotOptimum”. The state design pattern allows a change in state for the engine during testing and launching and it makes the state changing become less extensive.

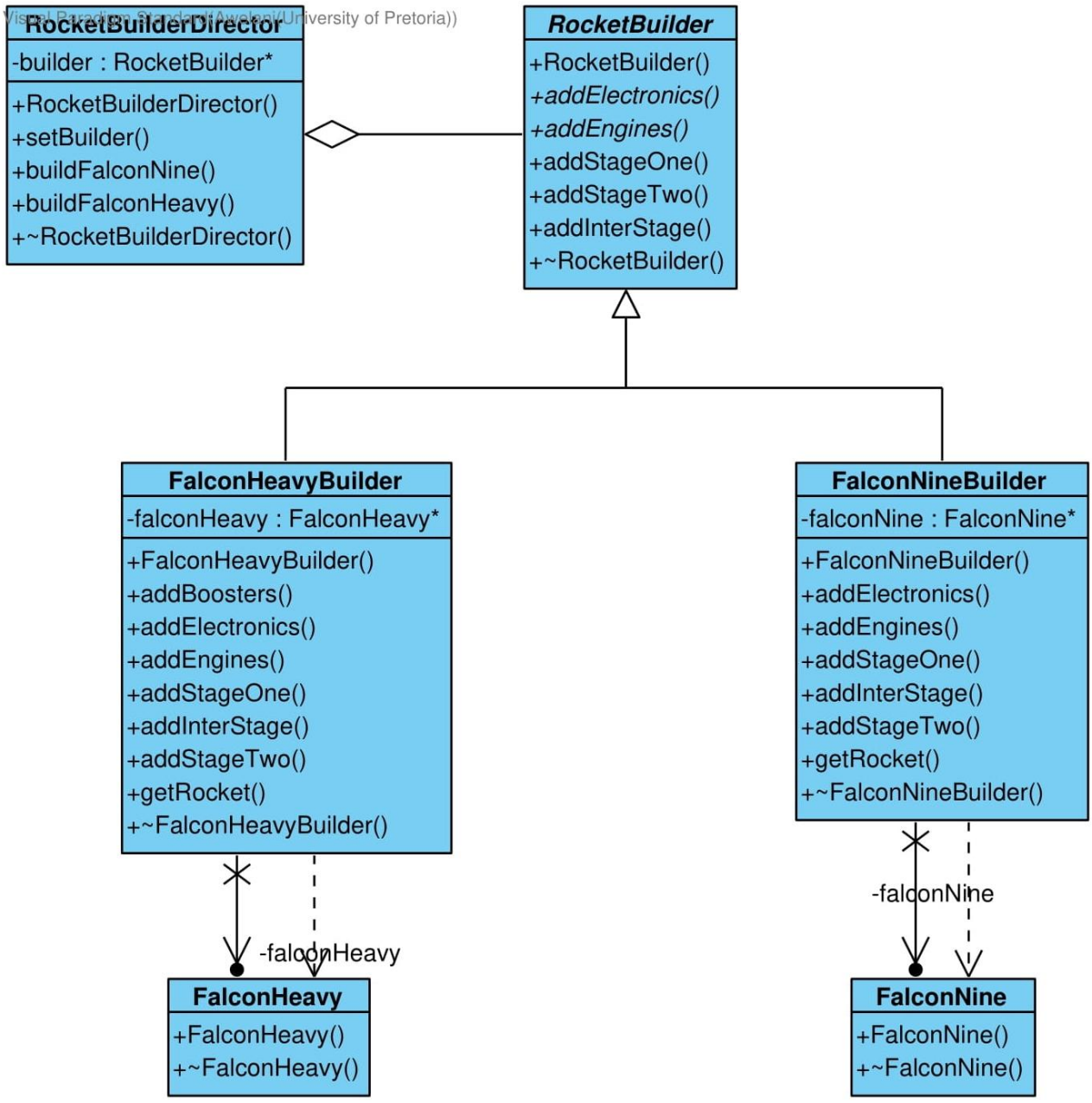


- Rocket State:
    - The State design pattern allows a change in the Rocket state, whether it's in Launch state, Separation state or land state the rocket can change states non extensively.



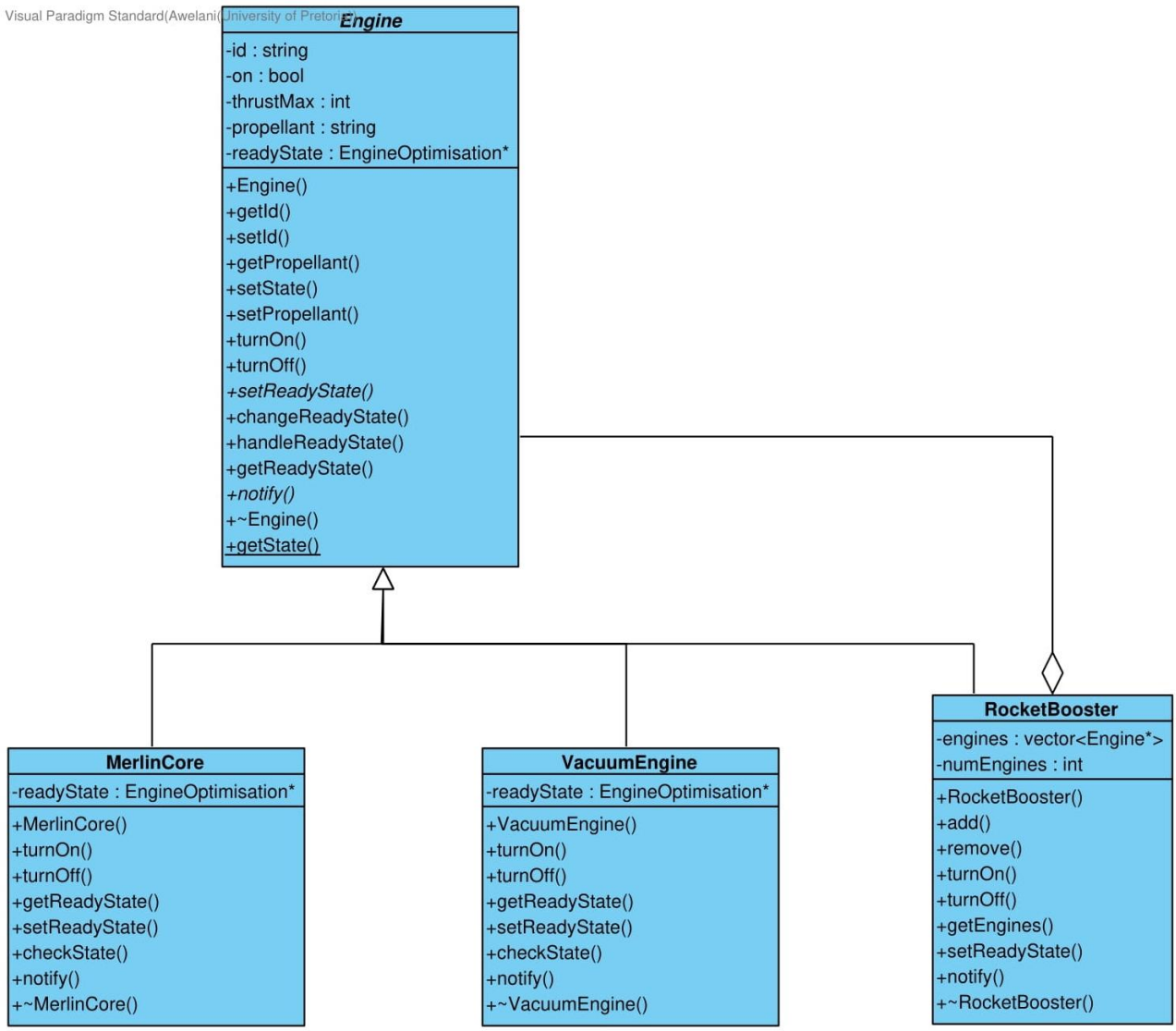
Builder Design pattern:

We decided to use the builder pattern along with the factory method pattern because the builder allows much more flexibility compared to other creational patterns, since rockets are complex objects and they vary. It makes more sense to use a pattern that would allow "building" the rockets step by step.



Composite Design pattern:

Since the falcon nine and the falcon heavy share the same types engines but instead the falcon heavy uses three times the amount of merlin core engines(two boosters with 9 engines each), we used Composite to allow use to treat the booster as a normal engine because it is composed of multiple merline core engines.

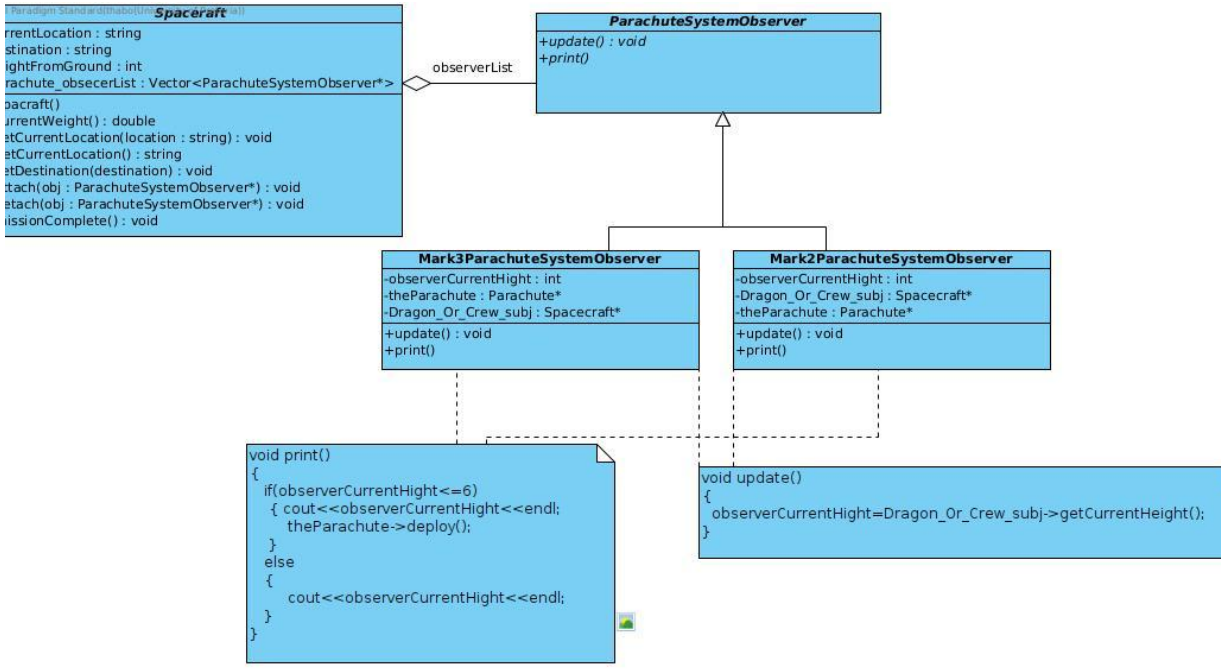

Visual Paradigm Standard(Awelani(University of Pretoria))
Engine
-id : string
-on : bool
-thrustMax : int
-propellant : string
-readyState : EngineOptimisation*
+Engine()
+getId()
+setId()
+getPropellant()
+setState()
+setPropellant()
+turnOn()
+turnOff()
+setReadyState()
+changeReadyState()
+handleReadyState()
+getReadyState()
+notify()
+~Engine()
+getState()
MerlinCore
-readyState : EngineOptimisation*
+MerlinCore()
+turnOn()
+turnOff()
+getReadyState()
+setReadyState()
+checkState()
+notify()
+~MerlinCore()
VacuumEngine
-readyState : EngineOptimisation*
+VacuumEngine()
+turnOn()
+turnOff()
+getReadyState()
+setReadyState()
+checkState()
+notify()
+~VacuumEngine()
RocketBooster
-engines : vector<Engine*>
-numEngines : int
+RocketBooster()
+add()
+remove()
+turnOn()
+turnOff()
+getEngines()
+setReadyState()
+notify()
+~RocketBooster()

Dragon Spacecrafts:

The spacecraft is divided into 2 types: the dragon spacecraft and the crew dragon spacecraft. The spacecraft takes cargo or in another instance crew members to and from earth and outer space. The spacecraft has to be able to go to outer space with cargo to come back and since it has to be reusable, parachutes are attached for it to land successfully and be able to be used again on another mission to outer space.

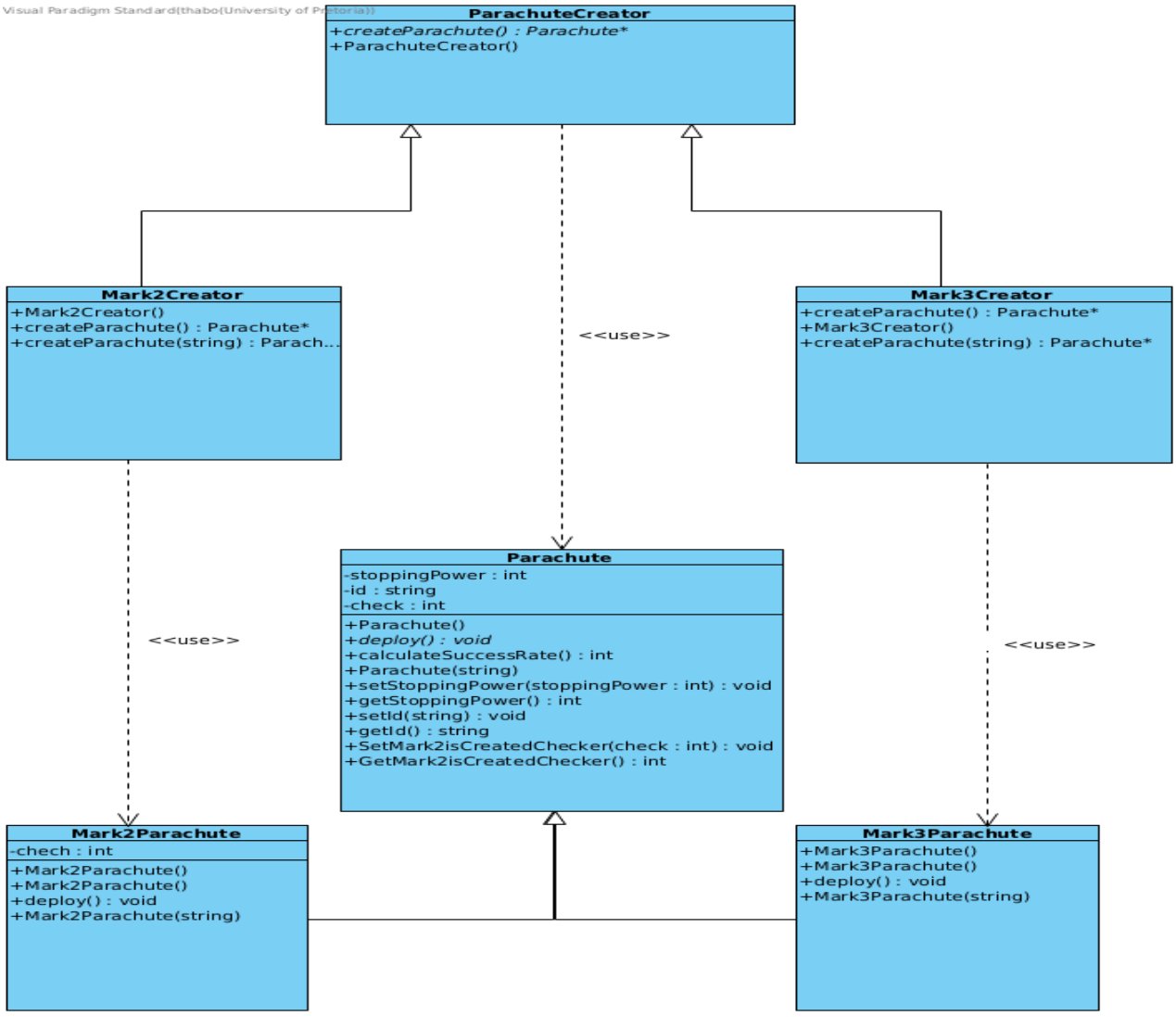
Observer Design pattern:

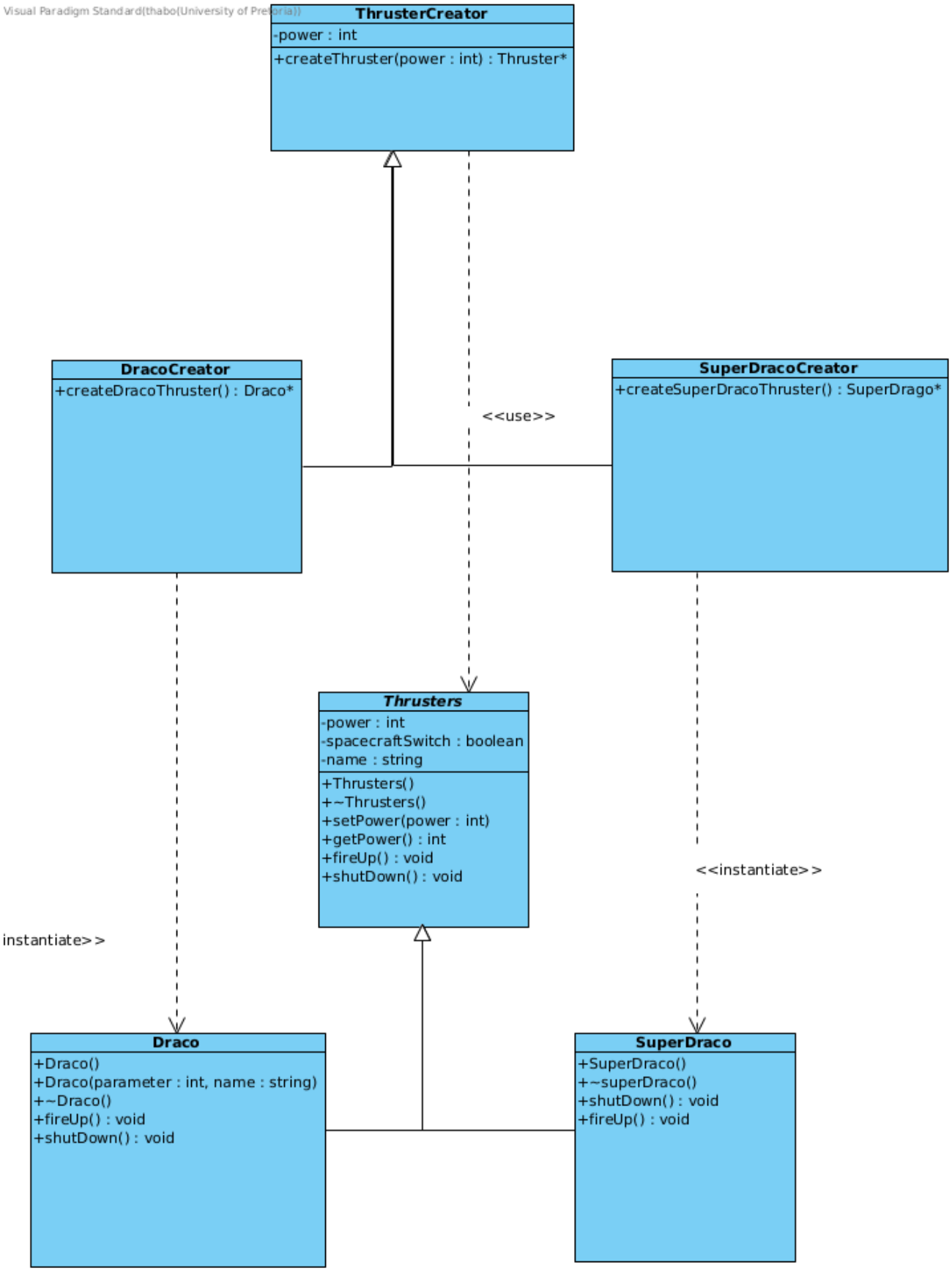
We used the Observer Design pattern in order for the spacecraft to be able to deploy the parachutes at an optimum height (6 kilometres from ground level).We achieved this by observing the current height of the spacecraft.When the spacecraft reaches earth's orbit (160 kilometres height from ground level). We initialised the current height of the spacecraft to 160 km, While we kept on decrementing the current height by 19.25. The ParachuteSystemObserver observed the current height of the spacecraft if the current height was not in optimum (6 kilometres from ground level) we displayed the current height else we deployed the parachutes.



Factory Method Design Pattern:

We used the Factory method to produce both our 16 thrusters of type Draco and SuperDraco, and also our 4 parachutes of type Mark2 and Mark3 for our spacecraft.


Visual Paradigm Standard(thabo(University of Pretoria))
ParachuteCreator
+createParachute() : Parachute*
+ParachuteCreator()
Mark2Creator
+Mark2Creator()
+createParachute() : Parachute*
+createParachute(string) : Parach...
<<use>>
Mark3Creator
+createParachute() : Parachute*
+Mark3Creator()
+createParachute(string) : Parachute*
Parachute
-stoppingPower : int
-id : string
-check : int
+Parachute()
+deploy() : void
+calculateSuccessRate() : int
+Parachute(string)
+setStoppingPower(stoppingPower : int) : void
+getStoppingPower() : int
+setId(string) : void
+getId() : string
+SetMark2isCreatedChecker(check : int) : void
+GetMark2isCreatedChecker() : int
<<use>>
<<use>>
Mark2Parachute
-chech : int
+Mark2Parachute()
+Mark2Parachute()
+deploy() : void
+Mark2Parachute(string)
Mark3Parachute
+Mark3Parachute()
+Mark3Parachute()
+deploy() : void
+Mark3Parachute(string)

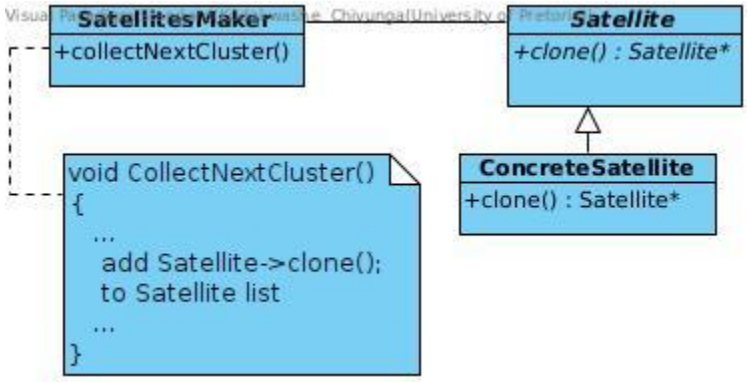

Visual Paradigm Standard(thabo(University of Pretoria))
ThrusterCreator
-power : int
+createThruster(power : int) : Thruster*
DracoCreator
+createDracoThruster() : Draco*
SuperDracoCreator
+createSuperDracoThruster() : SuperDrago*
<<use>>
Thrusters
-power : int
-spacecraftSwitch : boolean
-name : string
+Thrusters()
+~Thrusters()
+setPower(power : int)
+getPower() : int
+fireUp() : void
+shutDown() : void
<<instantiate>>
instantiate>>
Draco
+Draco()
+Draco(parameter : int, name : string)
+~Draco()
+fireUp() : void
+shutDown() : void
SuperDraco
+SuperDraco()
+~superDraco()
+shutDown() : void
+fireUp() : void

Starlink Satellites:

This subsystem creates and manages the Satellites that need to be launched. This subsystem allows Falcon 9 rockets to collect only 60 functional satellites at a time. This subsystem also makes sure all the satellites launched remain active and continue working accordingly.

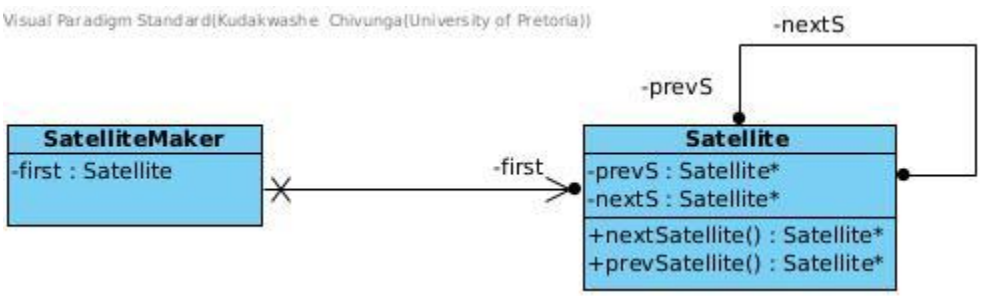
Prototype design pattern :

There was a need to provide a cluster of satellites that consisted of 60 functional satellites to the Falcon 9 rockets as per request. We created 1 prototypical instance of a Satellite object and used the prototype design pattern to clone 59 more from it, thus making a cluster of fully functional satellites.



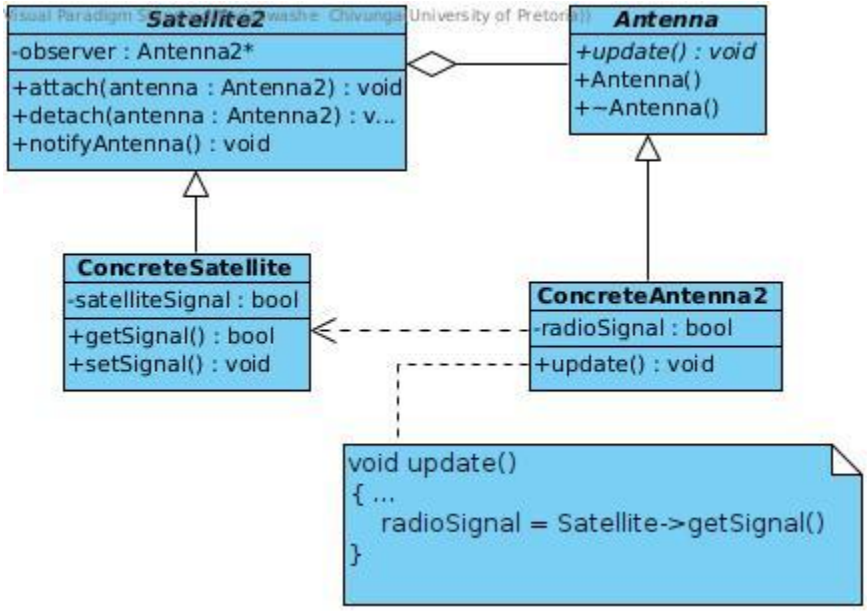
Iterator design pattern :

There was a need to provide access to the Satellites in the Satellites list individually. An iterator design pattern is then used to iterate through the Satellites list, setting the necessary parameters(distance between itself and its neighbouring satellites) of a Satellite to the desired values. We also used the iterator to link the cluster of satellites and to register all its neighbouring satellites using the pointers nextS (next satellite), prevS (previous satellite).



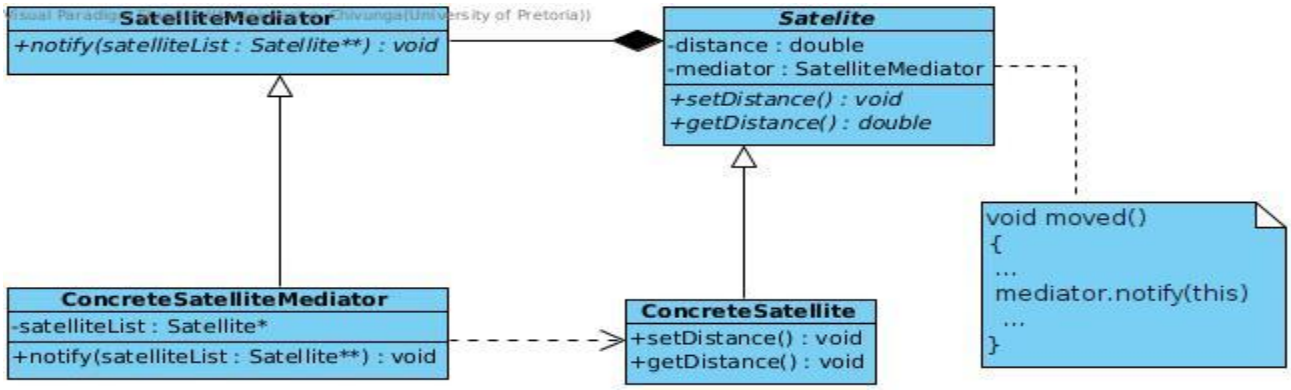
Observer design pattern :

We used the Observer design pattern in order for the Antennas on the ground to be able to communicate with the Satellites in orbit based on the change of state by the Satellite's radio signals. We achieved this by setting all of the radio signals initially set to false. The Antenna will then observe the Satellites radio signal states. If ALL Satellites radio signals are turned On , then the Satellites would be working. If ATLEAST one Satellite's radio signal is Off, the Antennas should notify the system.



Mediator design pattern :

We used the Mediator design pattern to enable communication between Starlink Satellites using. Messages were sent through the distance. In order for all Satellites to spread equally across the orbit, all of them should be able to keep a distance of 0.1km away from each other. If a Satellite moves out of position its laser goes on, notifying all the system to re-configure all Satellites positions. Satellites and then all of the Satellites will be repositioned in such a way that they are 0.1km away from each other again.



Launch Simulator:

This is where we brought all the subsystems together. This subsystems consists of

Command Design pattern:

The command design pattern is used to make an execution of the simulation class action without the need of keeping track of the Simulation class methods. The command hierarchy will execute the “TestSimulation” method and “Launch” method while using an invoker to invoke the current Command execution.