COS214 Project
Initial Design and Final Report

Group Name:

Coders Inc.

Group Members:

1. Alannah Abigail Sooruth (u19088133) – Team Leader
2. Mohamed Nizhar Aboobaker (u19001836)
3. Misbah Osman (u19028815)
4. Ponalo Notwane (u16115092)
5. Gift Monwa (u181196366)
6. Thangeni Faresa (u18312374)

Task 1 - Initial Design

Task 1.1

Function Requirements:

Task 4.1

- Create Rocket
- Create Engines
- Assign Engines
- Conduct static fire test on all engines
    - Switch Engines on
    - Test Engines
    - Refuel Engines
    - Switch Engines off
- Start Stage One
    - Switch Engines on
    - Test static fire test results for all engines
    - First launch
- Start Stage Two
    - Test Engines
    - Final launch into orbit

Task 4.2

- Create Spacecraft
- Create Cargo or Crew (or both)
- Assign haul to Spacecraft
    - Test Spacecraft's haul capacity
- Assign/set Spacecraft destination
- Send off (launch) Spacecraft
- Unload/retrieve cargo at Spacestation
- Return (launch again) Spacecraft

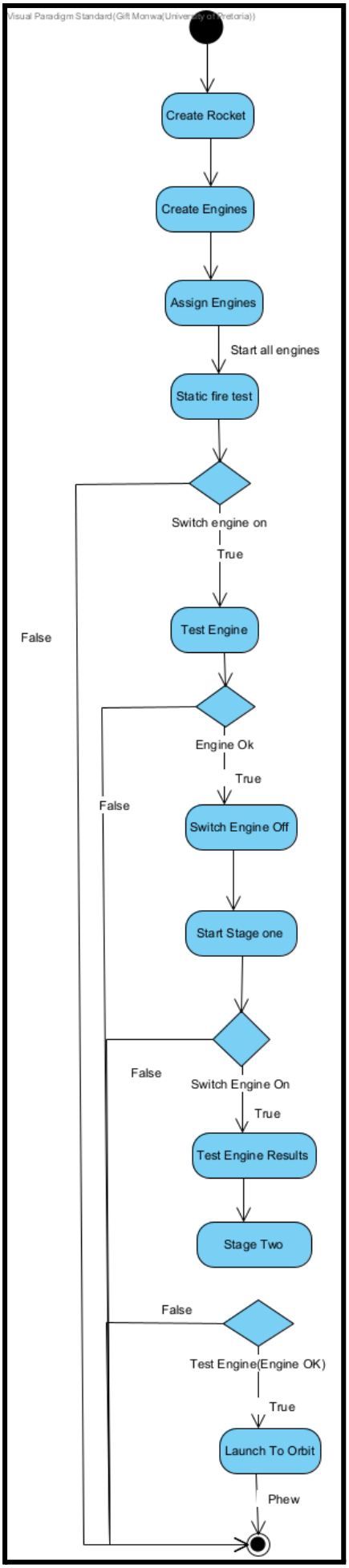
Task 4.3

- Create Satellites cluster
- Create Users
- Retrieve/Assign area coverage
- Launch Satellites
- Spread Satellites equally over designated area
    - Test distance between Satellites
- Satellites communicate with each other
- Satellites send signals to ground Users

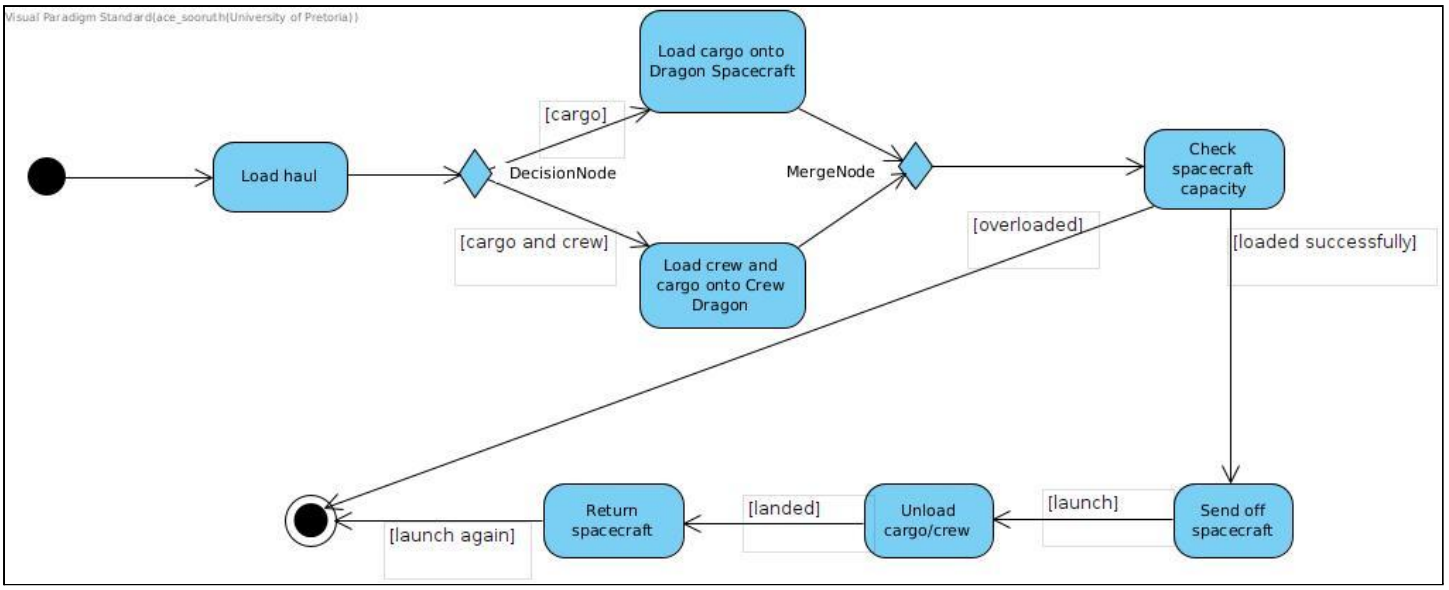
Task 4.4

- Test Engines
- Test Rockets
- Launch Rockets
- Load Cargo
- Test Spacecraft
- Launch Spacecraft
- Test Satellites
- Launch Satellites

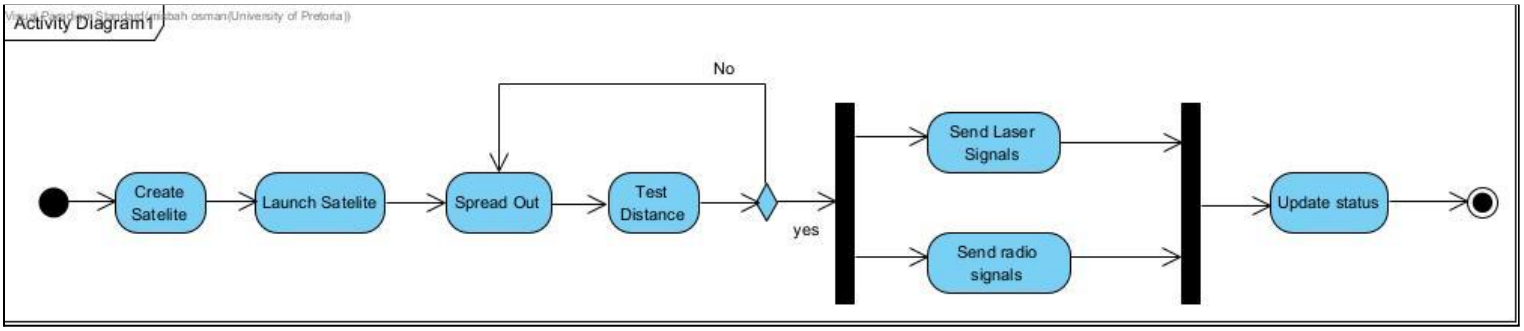
Task 1.2



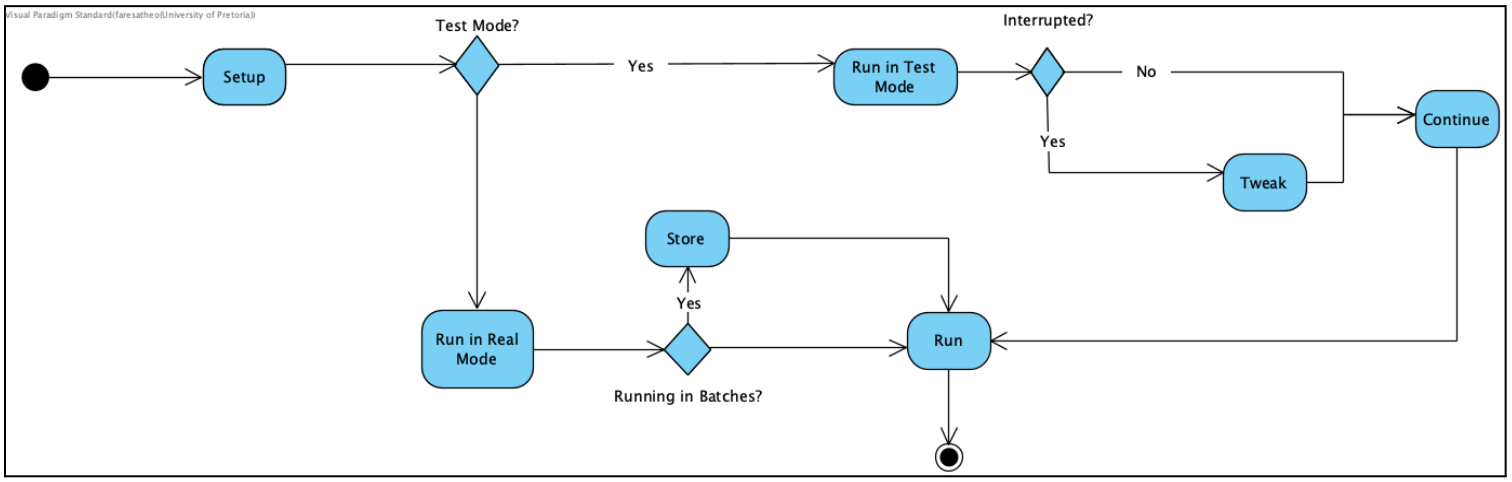
ANNEXURE A – TASK 4.1 ACTIVITY DIAGRAM



ANNEXURE B – TASK 4.2 ACTIVITY DIAGRAM



ANNEXURE C – TASK 4.3 ACTIVITY DIAGRAM



ANNEXURE D – TASK 4.4 ACTIVITY DIAGRAM

Task 1.3

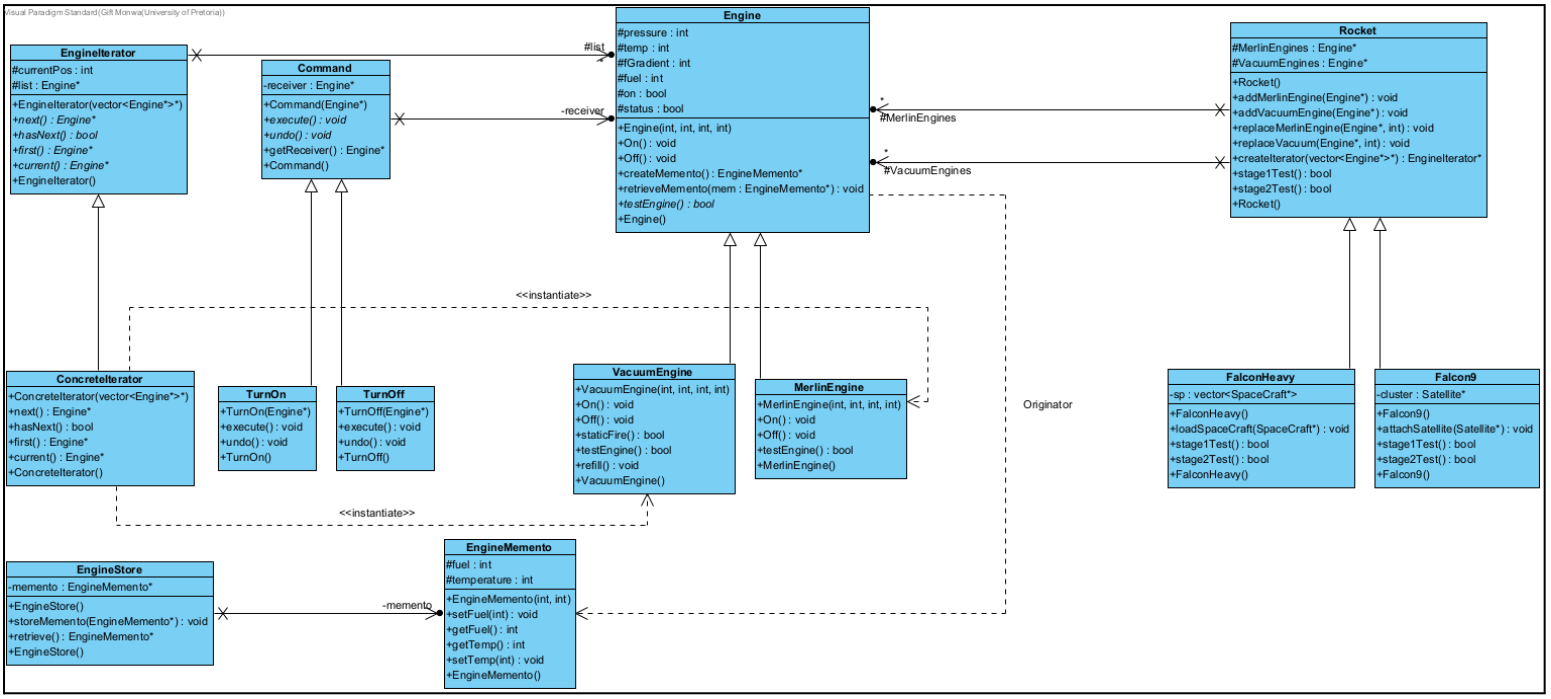
Design Pattern	Reason Chosen	Task Implemented In
Command	The Command design pattern encapsulates a request as an object. The Rocket class needs to “command” the Engine class to be ignited for launch. Thus the need to encapsulate the request	4.1
Iterator	The Rocket class is an aggregate class that contains a vector of engine objects,The iterator class provides a convenient way of iterating through the engines in the Rocket	4.1
Memento	Due to the tests done on the Fuel attribute from the Engine Class, said attribute’s value/state will fluctuate throughout the program. This calls for a need to capture and externalise an the state so that the Engine can be restored to original state	4.1
Factory Method	The Spacecraft objects are created based off the type of cargo they carry, thus a Factory Method would define an interface for creating an object, but letting subclasses decide which class to instantiate	4.2
Strategy	Strategy allows us to encapsulate each Load/Cargo algorithm, and make them interchangeable to be used for loading.	● 4.1 ● 4.2
Observer	Observer pattern defines a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically. Thus allowing all users on the ground to be notified when an issue occurs with one satellite	4.3

Mediator	Mediator patterns defines an object that encapsulates how a set of objects interact. Thus allowing all satellites in a cluster to communicate with each other without interacting with the users on the ground	4.3
Prototype	A cluster of 60 identical satellites require the specification types of on object to create using a prototypical instance, and create new objects (satellites) by copying this prototype	4.3
Template	Defines the skeleton of an algorithm in an operation, deferring some steps to subclasses; allowing us to derive versions of Rockets and Engines	4.1
Chain of Responsibility	Chains the receiving objects and pass the request along the chain until an object handles it, allowing to handle each simulation for each spacecraft	4.4

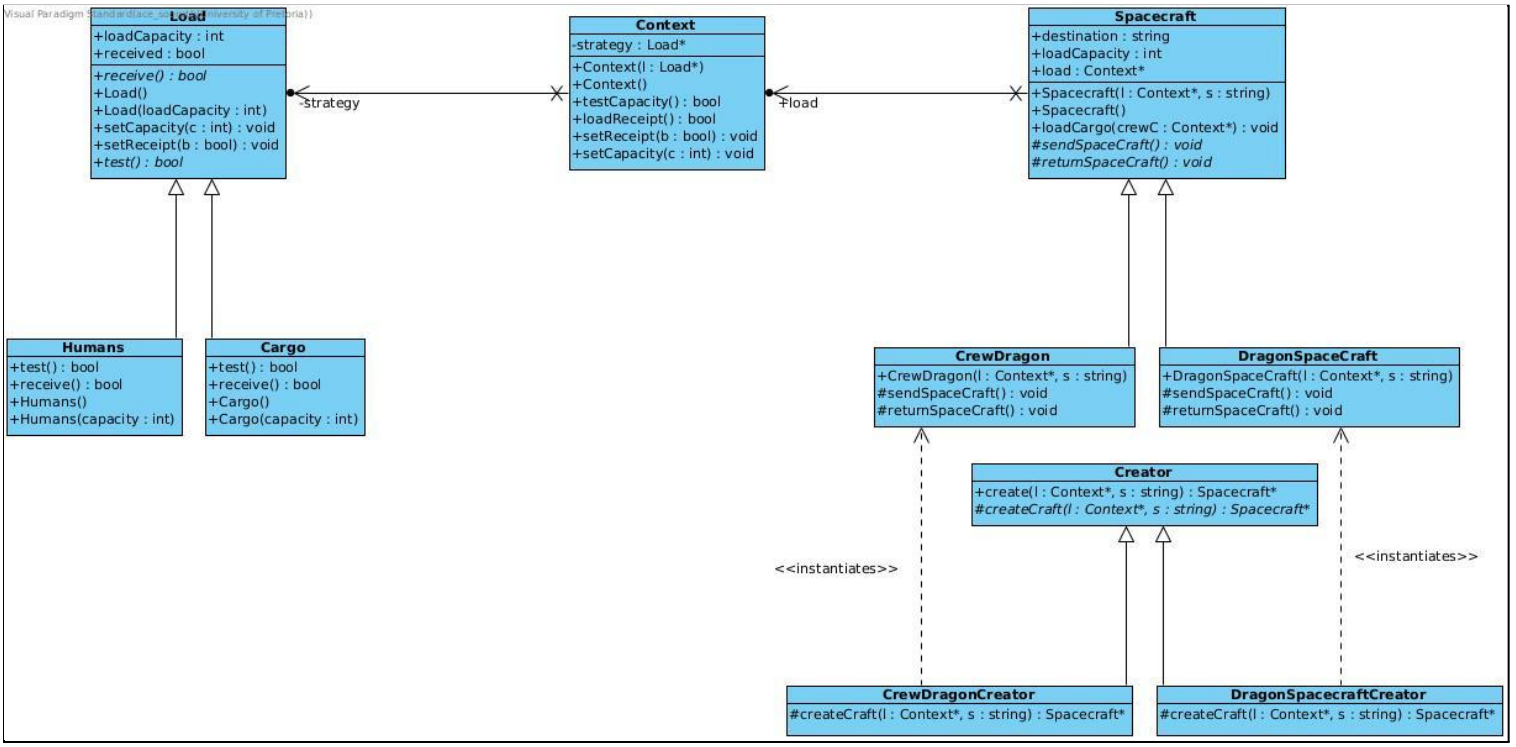
Task 1.4

****FOR DESIGN OF CLASSES, PLEASE REFER TO THE "CLASS DESIGNS" FOLDER CONTAINED WITHIN SUBMITTED ZIP FOLDER****

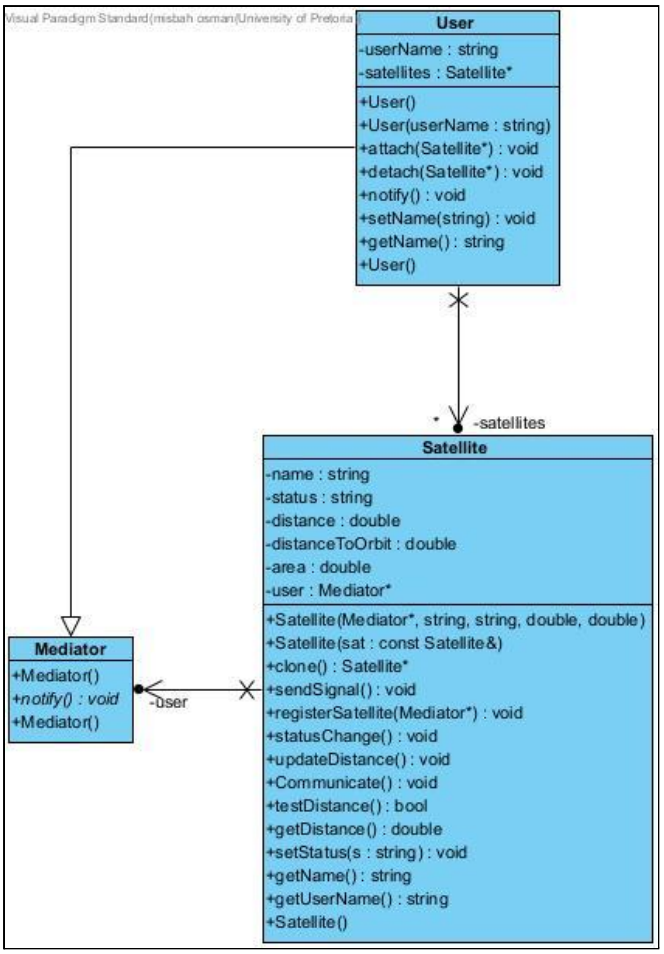
Task 1.5



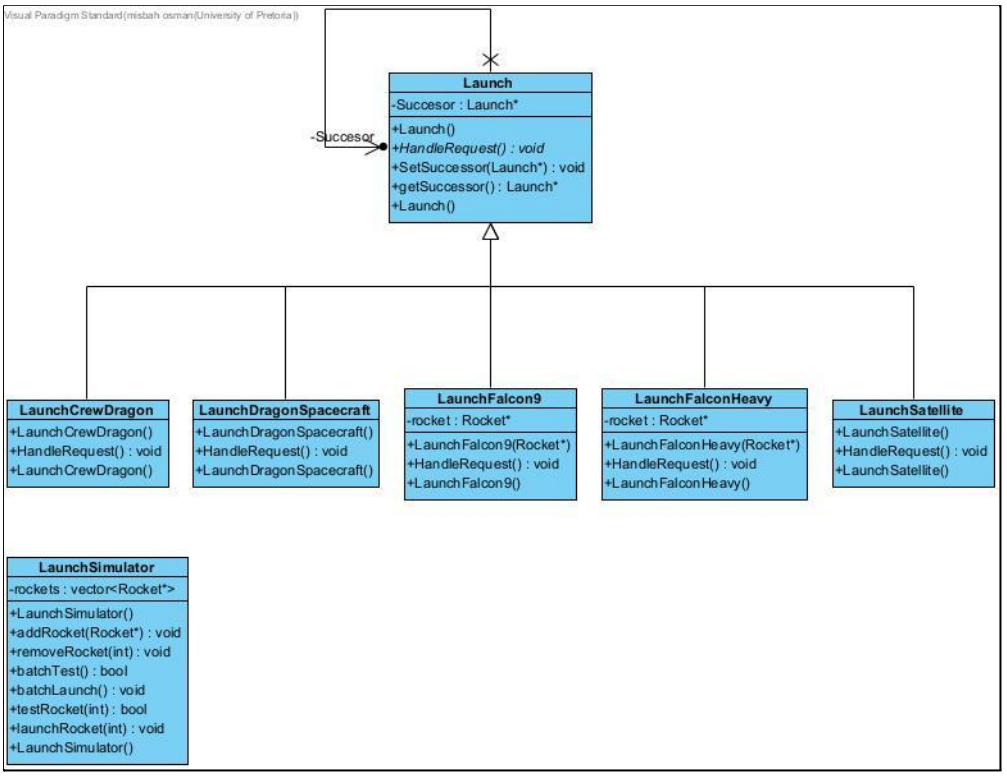
ANNEXURE E – TASK 4.1 CLASS DIAGRAM



ANNEXURE F – TASK 4.2 CLASS DIAGRAM

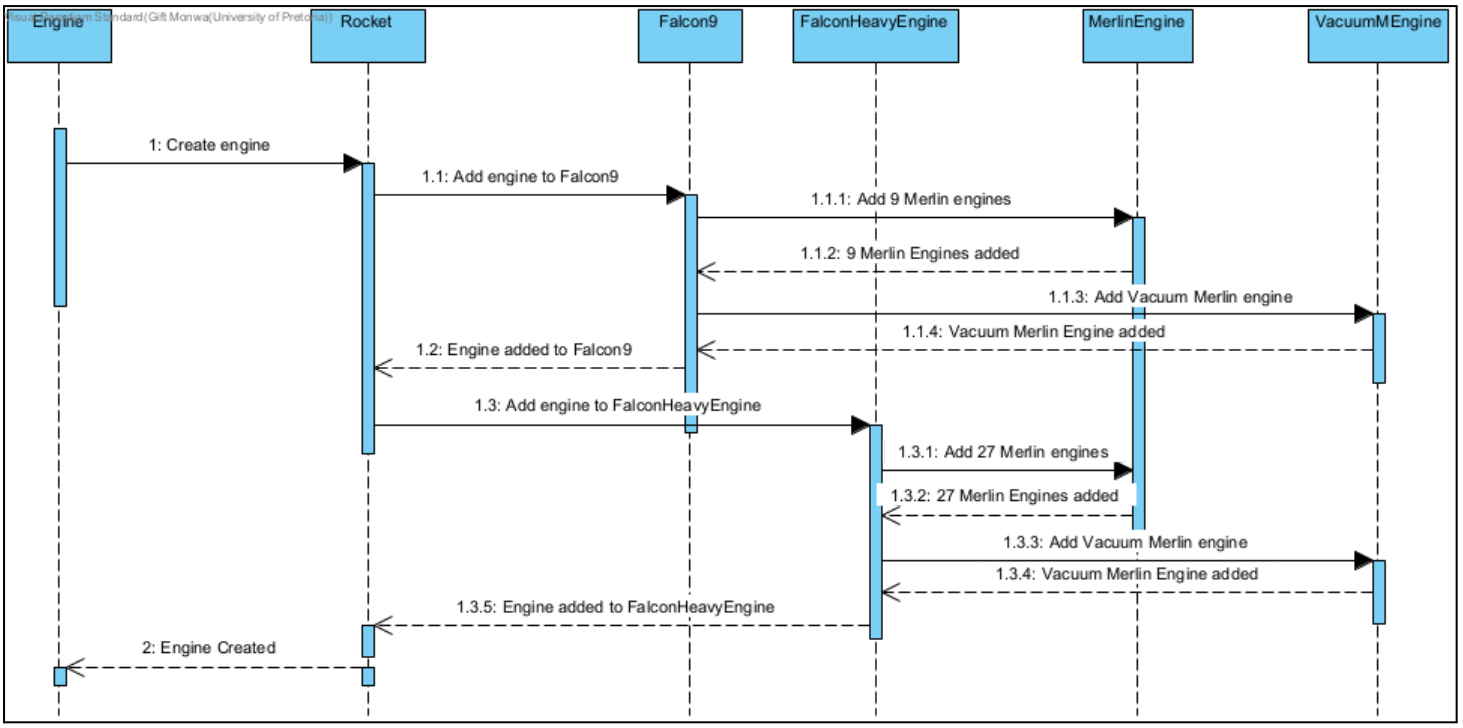


ANNEXURE G – TASK 4.3
CLASS DIAGRAM

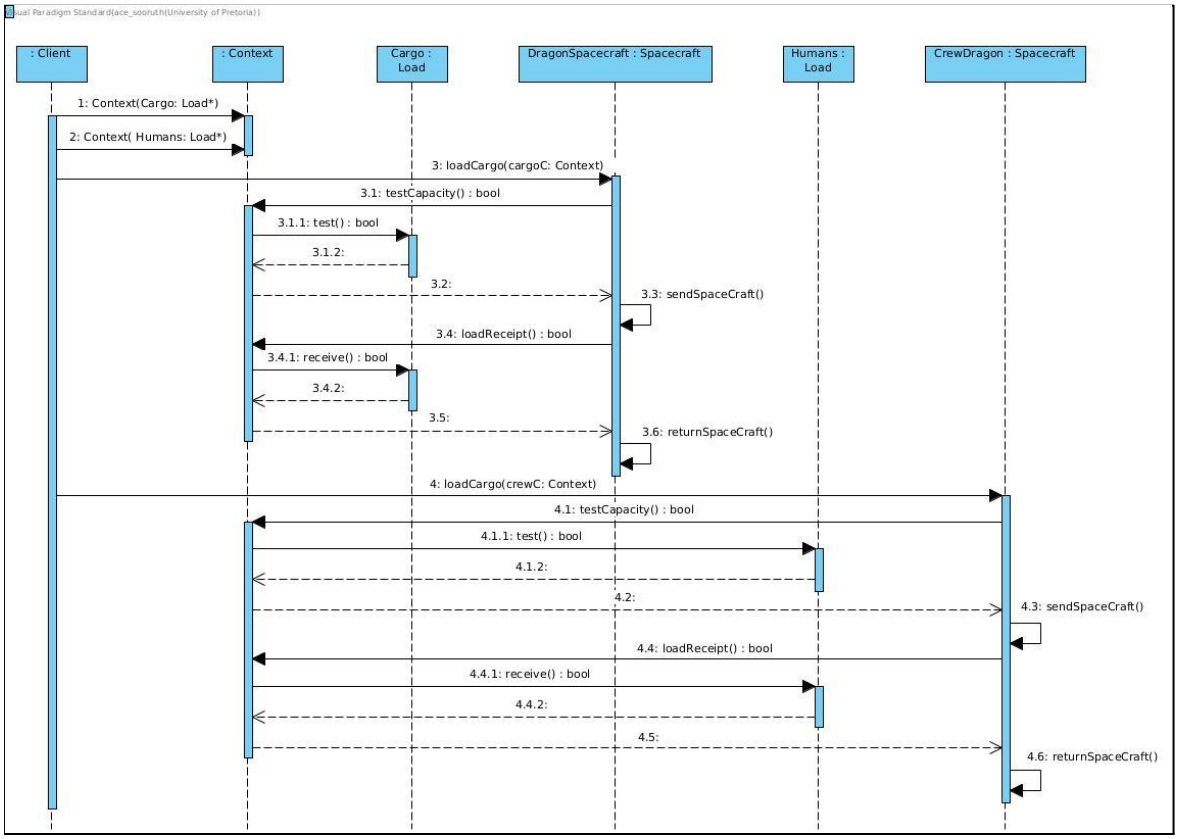


ANNEXURE H – TASK 4.4 CLASS DIAGRAM

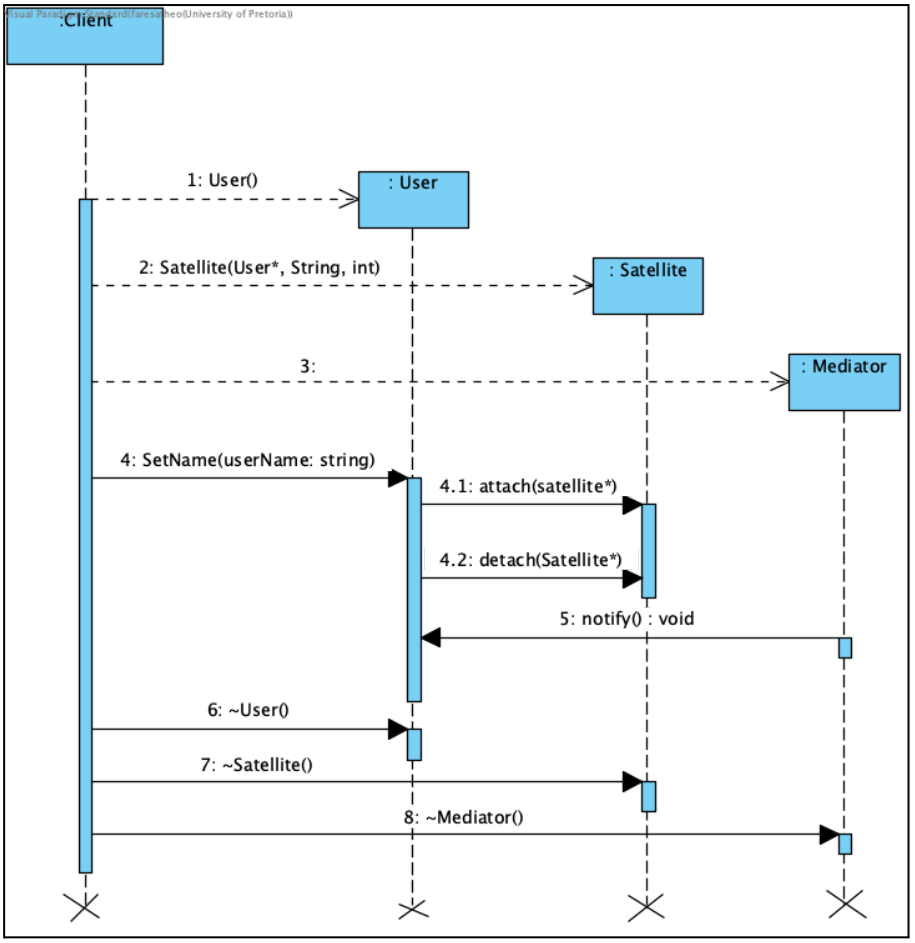
Task 1.6



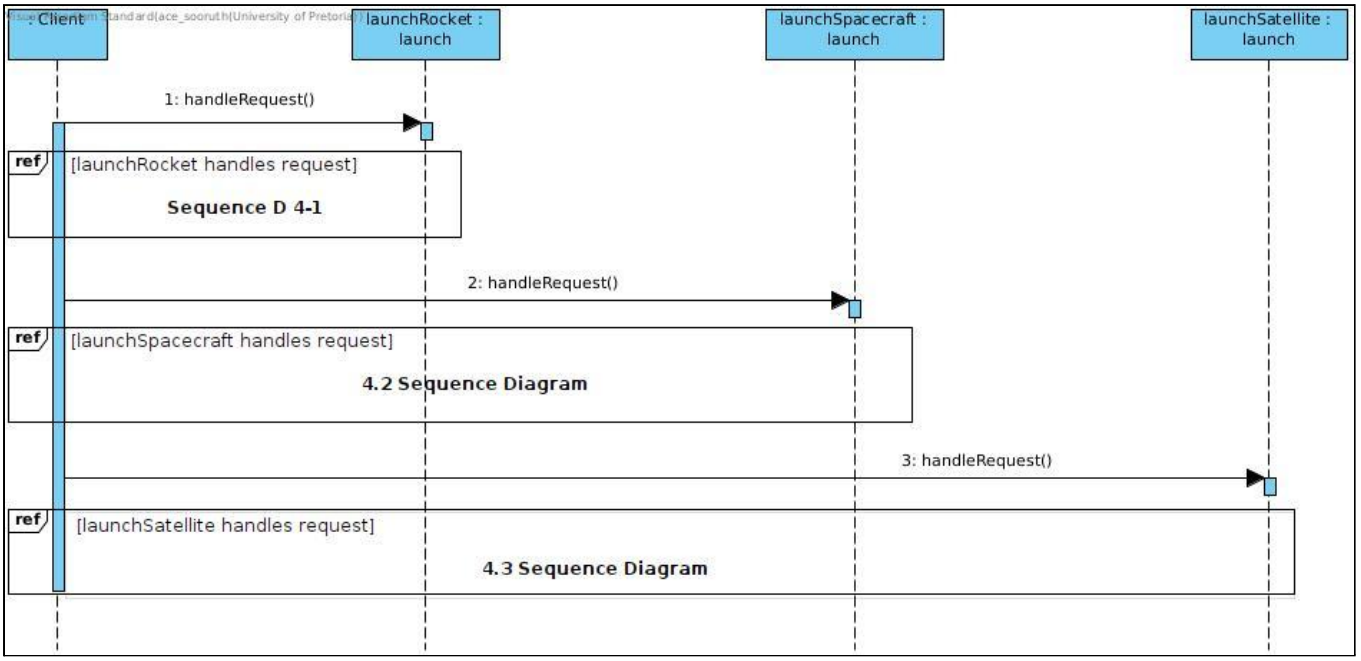
ANNEXURE I – TASK 4.1 SEQUENCE DIAGRAM



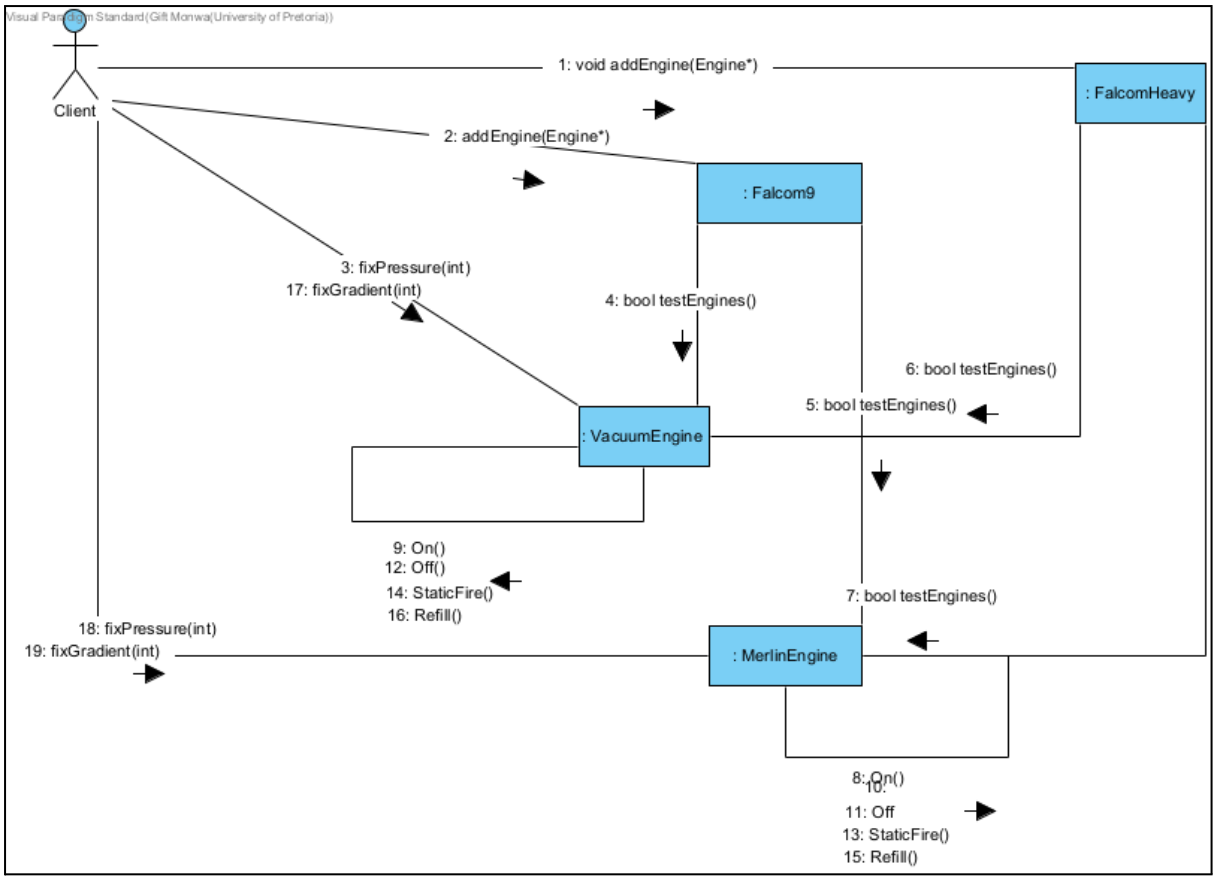
ANNEXURE J – TASK 4.2 SEQUENCE DIAGRAM



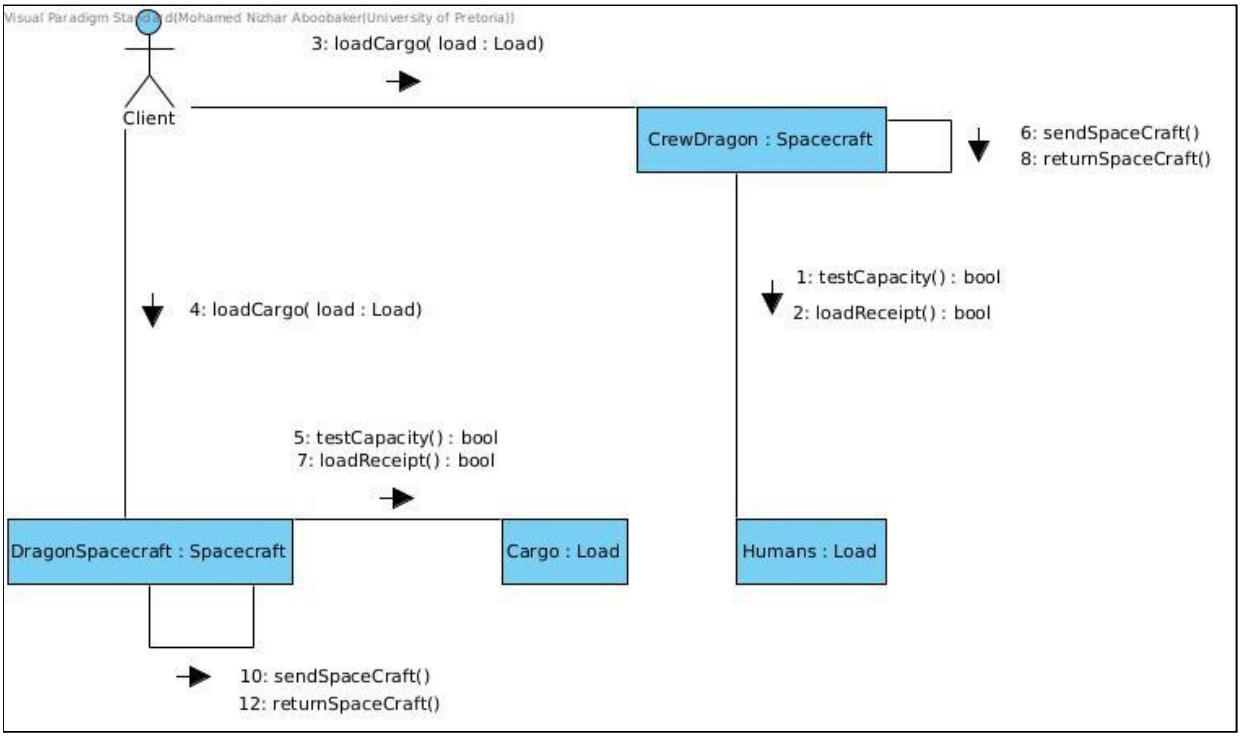
ANNEXURE K – TASK 4.3 SEQUENCE DIAGRAM



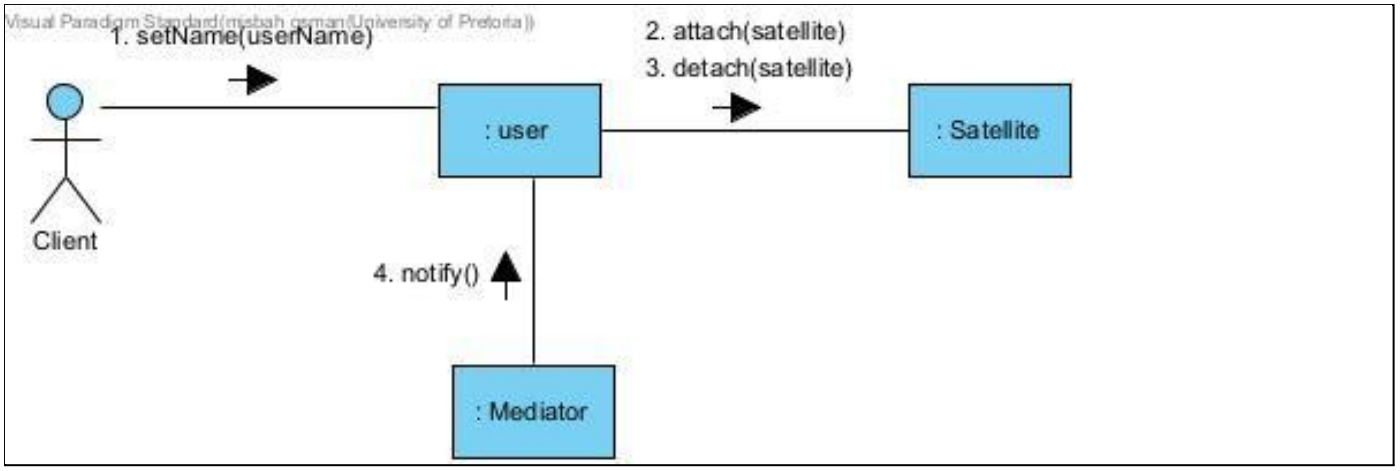
ANNEXURE L – TASK 4.4 SEQUENCE DIAGRAM



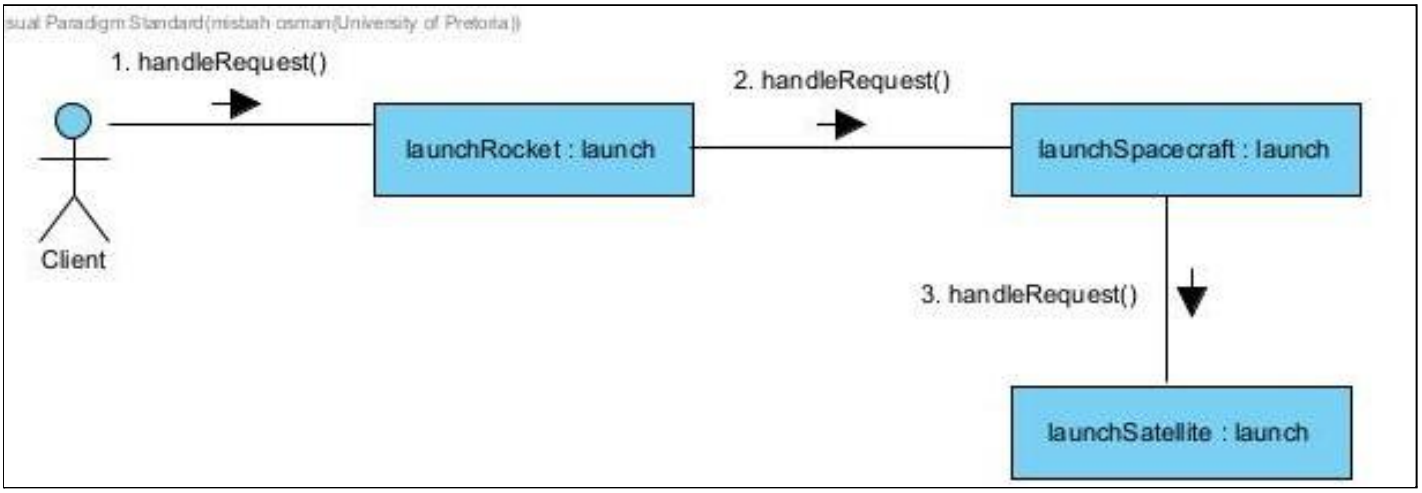
ANNEXURE M – TASK 4.1 COMMUNICATION DIAGRAM



ANNEXURE N – TASK 4.2 COMMUNICATION DIAGRAM

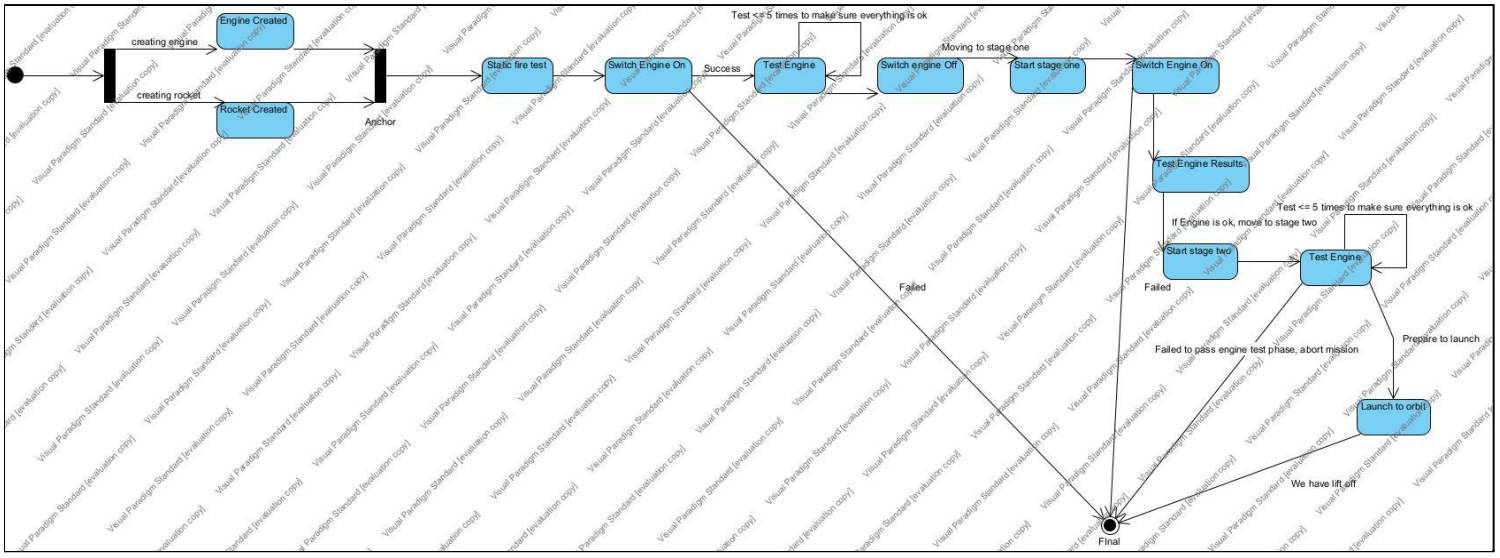


ANNEXURE O – TASK 4.3 COMMUNICATIONS DIAGRAM

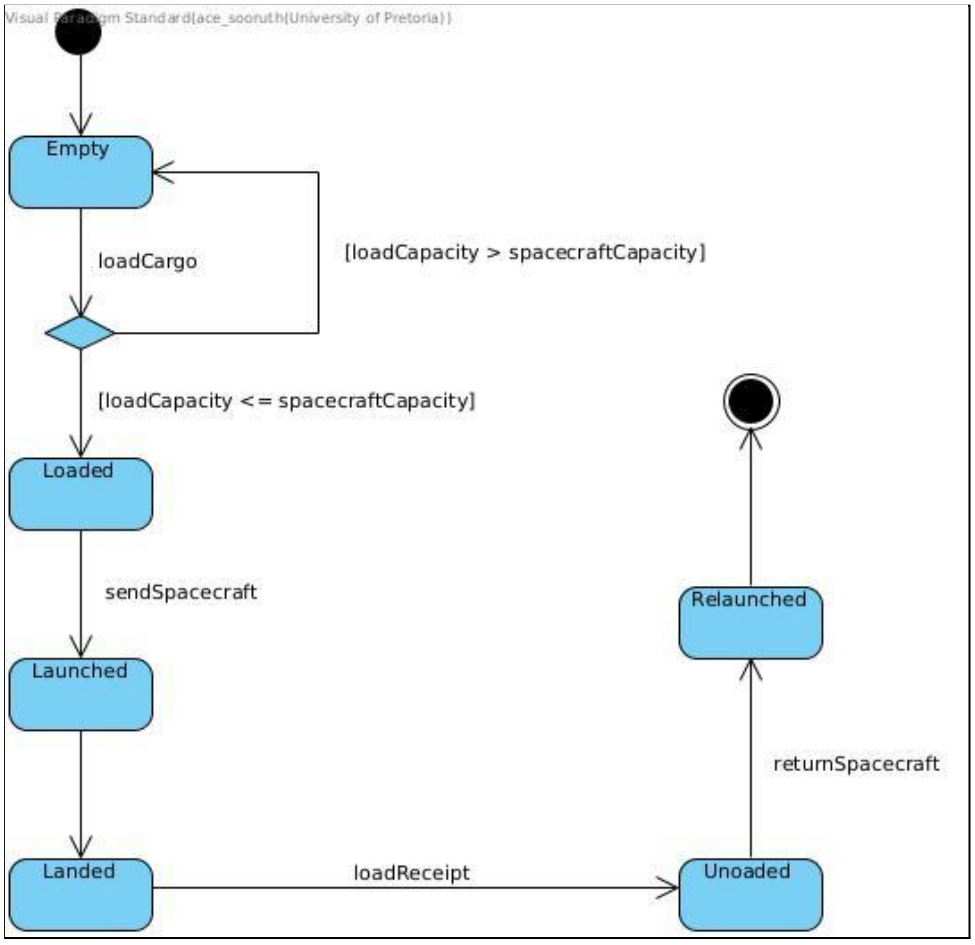


ANNEXURE P – TASK 4.4 COMMUNICATION DIAGRAM

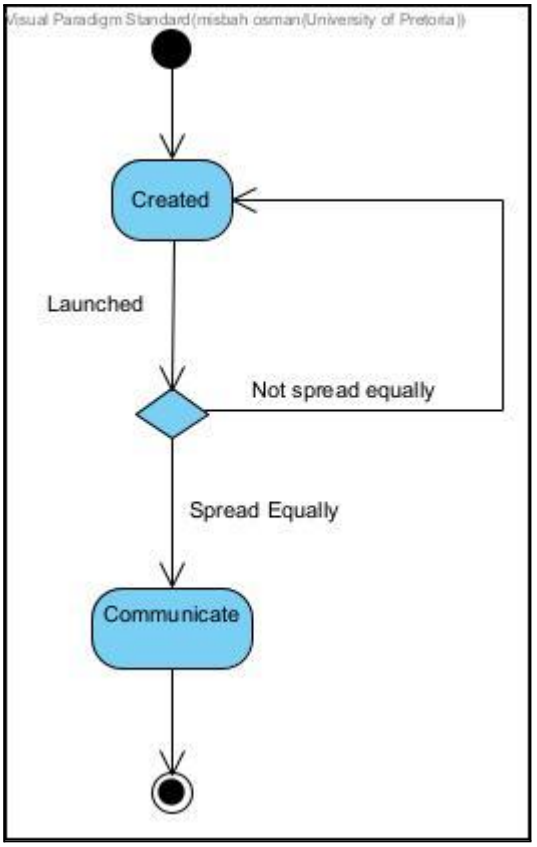
Task 1.7



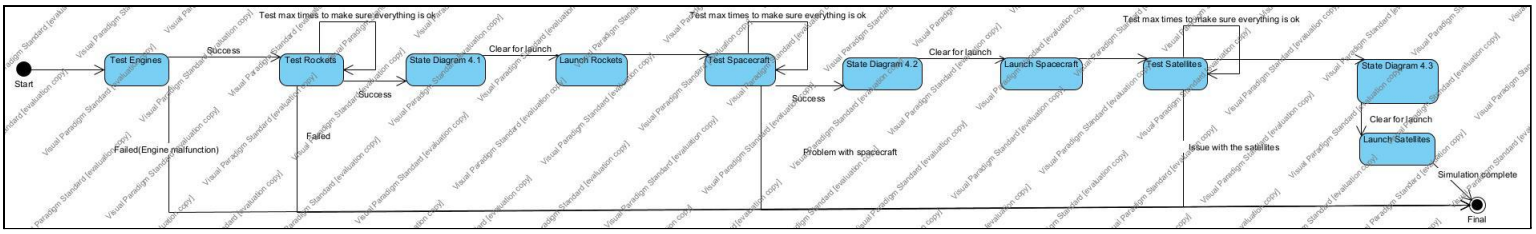
ANNEXURE Q – TASK 4.1 STATE DIAGRAM



ANNEXURE R – TASK 4.2 STATE DIAGRAM

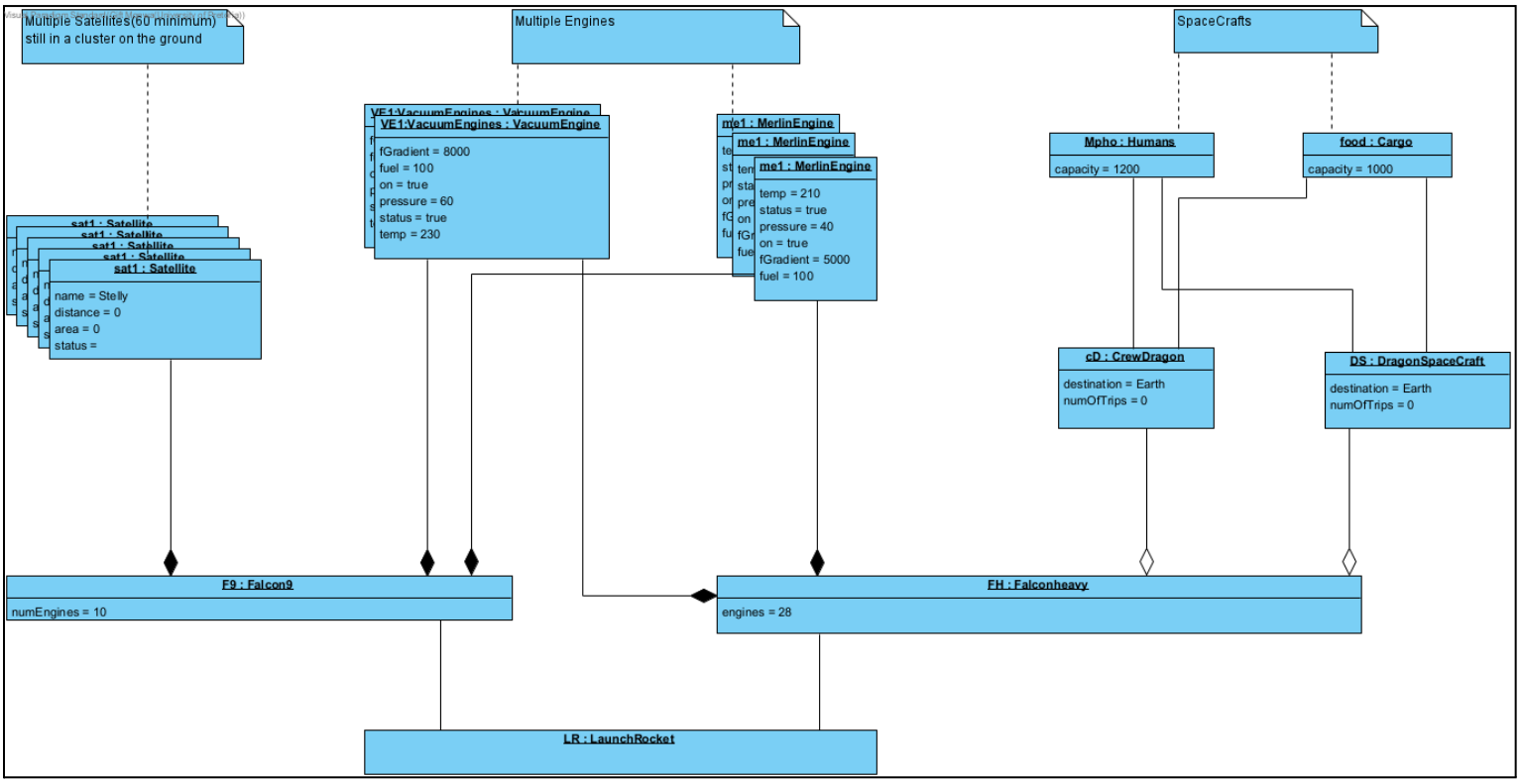


ANNEXURE S – TASK 4.3 STATE DIAGRAM

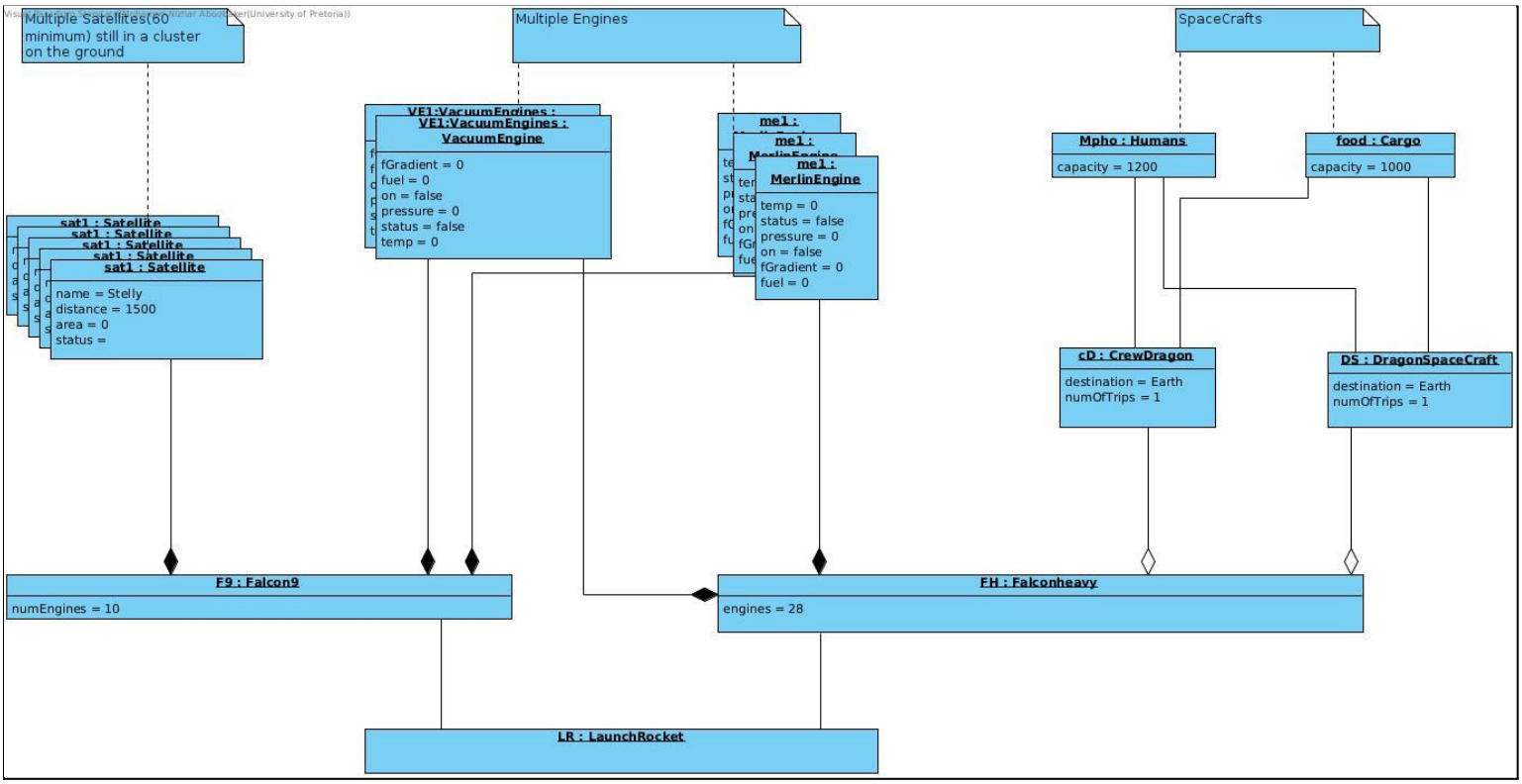


ANNEXURE T – TASK 4.4 STATE DIAGRAM

Task 1.8



ANNEXURE U – OBJECT DIAGRAM BEFORE LAUNCH



ANNEXURE V – OBJECT DIAGRAM AFTER LAUNCH

Task 3 - Report

Task 4.1

We initially included adapter Design pattern in our design plan but upon inspection and evaluation there was no way that we could've effectively used an adapter to connect engine to Rockets as engines are just independent classes that are just added to the Rocket.Since Rocket was an aggregregate class containing engine objects ,an iterator design pattern was best suited to iterate through the engine objects. The Engine class had 2 subclasses and they each had their unique testing(testEngine()) functions hence the strategy design pattern was best suited to facilitate the choice of function depending on the engine subclass that we'd go with.The command design pattern was very useful in turning on the engines remotely from rocket without directly accessing the engine.

Task 4.2

Upon construction and development of the code required to implement the Spacecraft and Cargo task, we discovered that the design patterns assigned to Task 4.2 worked well. As expected a Strategy design pattern was needed to implement the Cargo and Crew for each Spacecraft to encapsulate the algorithms and use them interchangeably, based off the Spacecraft being simulated. The Factory Method design pattern also worked well to implement the Spacecraft objects because the subclasses created could decide which object to instantiate, based on which cargo was being loaded.

Task 4.3

Regardless of few challenges we came across during the implementation following the Observer design patterns, the design patterns worked well. The Observer design pattern was used to notify users whenever the satellite sent a signal or had any changes that a user should know of.
Instead of creating new 60 satellites one by one, we created only one and cloned it. We achieved this by using the Prototype design pattern as they are all identical.
Mediator design pattern implementation allowed interaction between the user and the satellites.

Task 4.4

As expected, the Chain of Responsibility design pattern worked well to implement a simulation for the project as a whole. The user and task was unaware of how each individual simulation was run, but it allowed for traversal through all objects in the simulation to appear as one working system.

Project Links

GitHub Repository Link:
https://github.com/214-CodersInc/COS214_Project.git

Google Documents Link:
https://docs.google.com/document/d/1sgGxACLRhwNTAScq4Bmo4eFkVXgVKy3iKQmS7J_qw_0/edit?usp=sharing