COS214 Project: Report

Gang-of-Five

Github Link

https://github.com/brandon-c-k/gangoffive.git

Google Doc. Link

https://docs.google.com/document/d/1rdpxj7wb9vDE23vbLYERxsj-AKM8Tn2HbCtHOkjFYNE/edit?usp=sharing

Brandon Kruger: U18019499
Sarah King: U20481218
Cornel Coetzee: U20586737
Munashe Mashonganyika: U19077450
Gerard Nagura: U19004232

Introduction

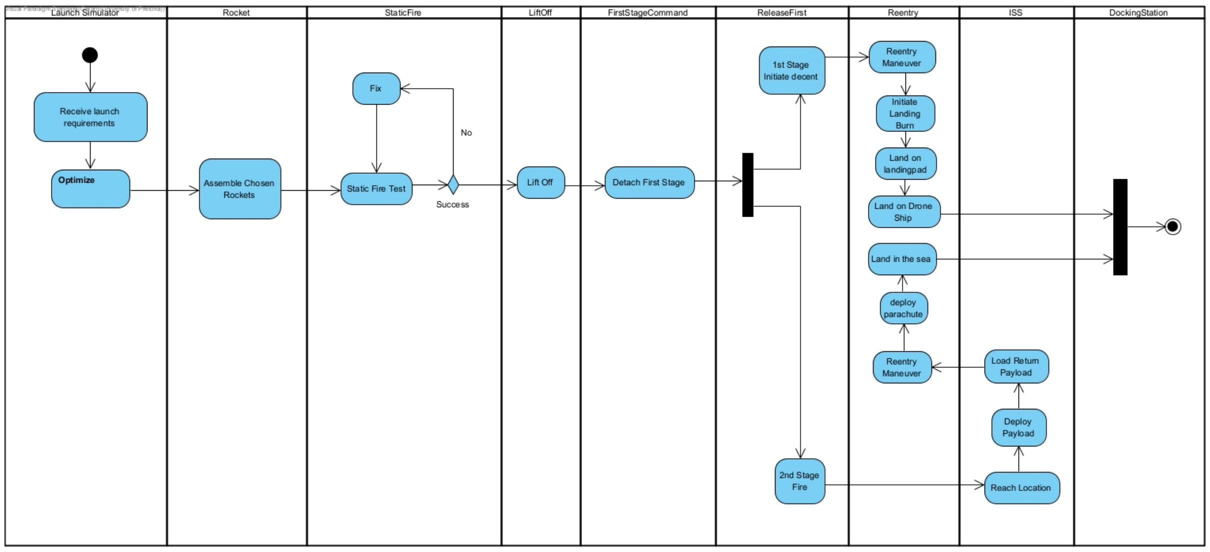
We were tasked with creating a system that simulates rocket launches based on user inputted variables. The system needs to choose the best configuration in order to build a rocket according to the user specifications, with the main goal of optimizing launches in order to reduce costs.

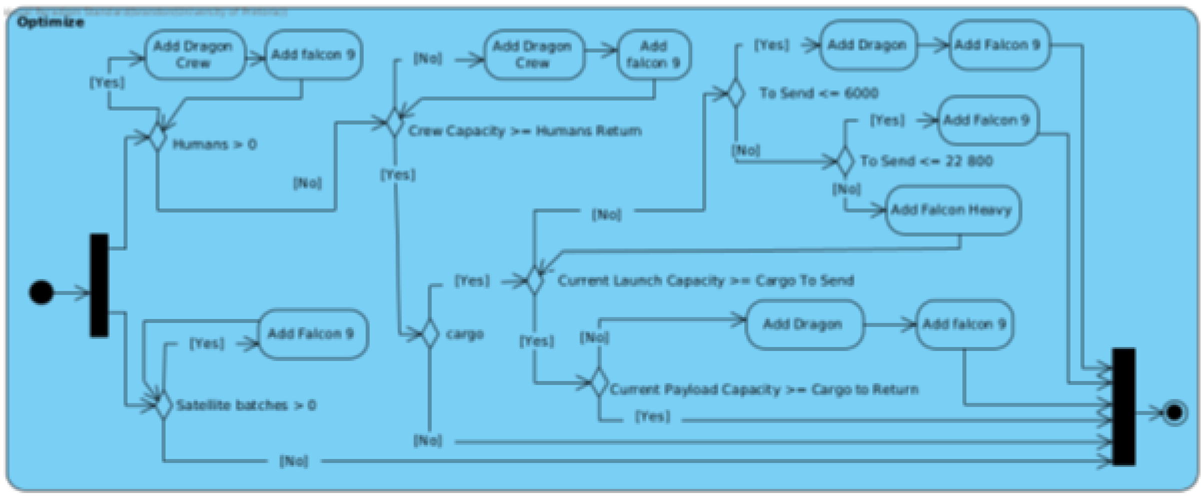
1.Initial Design

1.1 Functional Requirements

1. System needs to choose the best rocket and spacecraft configuration based on user inputted variables in order to minimize cost
2. Rockets need to be tested via a static fire before launch
3. System needs to be able to store actual launch simulations so that they can be later run-in batches.
4. System needs to be able to deal with errors during launches and handle them accordingly.
5. First stage of rocket needs to land in the ocean on a drone ship so that it can be reused
6. Starlink satellites need to be able to communicate with one another and users on the ground once in low-Earth orbit.
7. The coordinates of both stages of the rocket need to be known at all times simultaneously.
8. The system needs to have an interface wherein the user can select what they want to send to space.
9. The Rocket configuration needs to communicate with the system when a major event occurs, e.g., successful connection to ISS and successful detachment of stage 1 .

1.2 Activity Diagrams



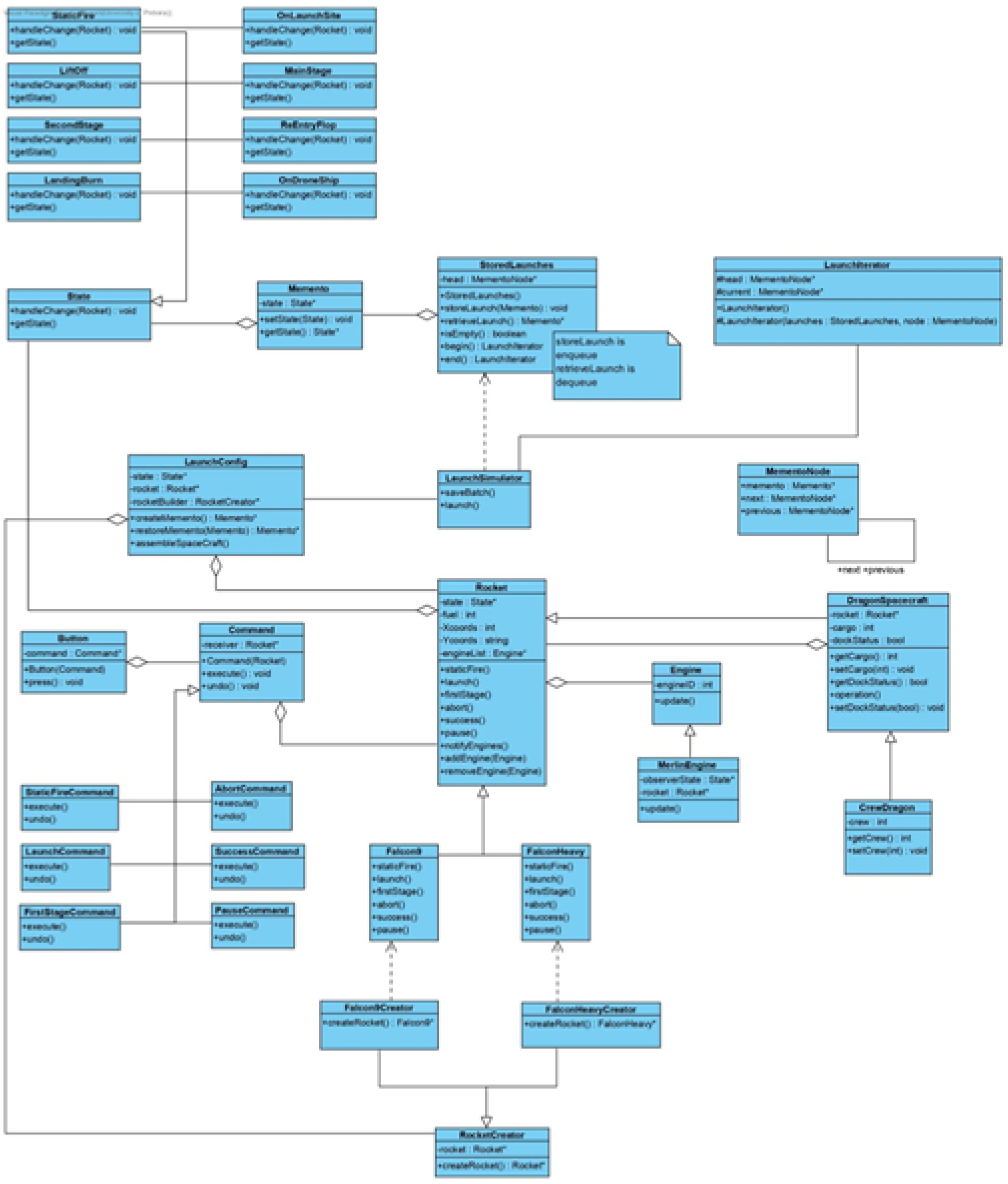


1.3 Design Patterns

These are the design patterns we have chosen in order to address our previously defined functional requirements:

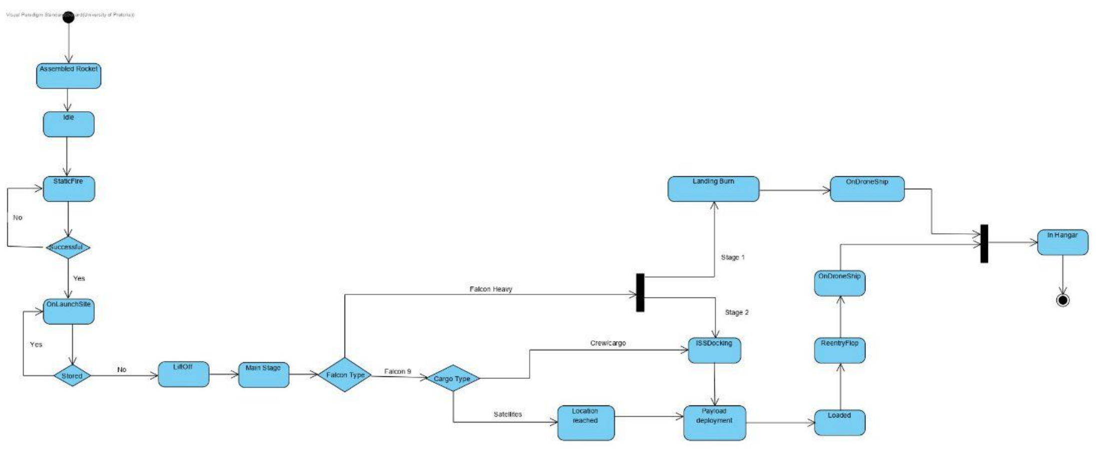
1.1 The Abstract Factory method will be used to construct the different rockets
1.2 The Decorator method will be used to add the spacecraft and relevant cargo/crew to the rocket configuration
1.3 The Template method will be used to create the Crew Dragon and Dragon spacecraft from the abstract class Spacecraft.
1.4 Observer method will be used on the engines to be updated/notified when a rocket changes its state.
1.5 The Memento design pattern will be used in order to store and launch the batches of launch configurations
1.6 The Iterator design pattern will be used in order to run single launches stored in a batch.
1.7 The Command pattern will be used to control different phases of the launch process.
1.8 The State design pattern is used to indicate and interact with the state of the rocket in the different phases of its launch process.
1.9 The Prototype design pattern is used to create and clone the engines of the rockets.
1.10 The Strategy pattern will be used in order to select an appropriate rocket creator for the launch configuration class.

1.5 Class Diagram & Design of Classes



<u>1.6 State Diagram</u>

<u>State Diagram showing the different states the rocket goes through from assembly to re-entry to the Earth's atmosphere:</u>



Design Alterations & Challenges

- We decided to change the Rocket coordinate variables to a single altitude position as this is more relevant with respect to stage detachment and fuel calculations.
- We added an ISS class along with a DockingStation class which are participants in the Visitor design pattern.
- We added a clone function to the engine class which is used by the Prototype design pattern.
- We decided to simplify our initial state diagram, correct errors made and make sure that states included were defined in our class diagrams.
- The implementation of the pause command was changed from being part of the command pattern to be an intermediary stage between each state of the rocket. When a Rocket object changes its state, the user is prompted with options: abort, change parameters, continue. This stage acts as the pause for the test simulations and gives the user full control of test launches.
- We decided to add a RocketNode class. This has a similar implementation to the MementoNode class, but instead of holding a Memento object, it holds a Rocket object.
- The previous change results in the StoredLaunches class now holding a RocketNode instead of a MementoNode. The Iterator class, LaunchIterator, now iterates through RocketNodes , instead of MementoNode's.
- The Abstract Factory design pattern classes have been updated to add the relevant spacecraft during creation of the chosen rocket type.
- The implementation of the launching of rockets has been altered, such that when rockets are run through the test mode, they will be saved as a Memento, after any user changes, they will be prompted to save the changes. If the user selects yes, a new Memento will be created and saved, if not , the previous rocket will be restored using the memento stored previously.
- A capacity attribute was added to both types of Rockets in order to help differentiate between them when optimizing launches. The cargo weight, which is determined by user input, will be used against the set capacity of each rocket in order to choose the right rocket for the specified variables.

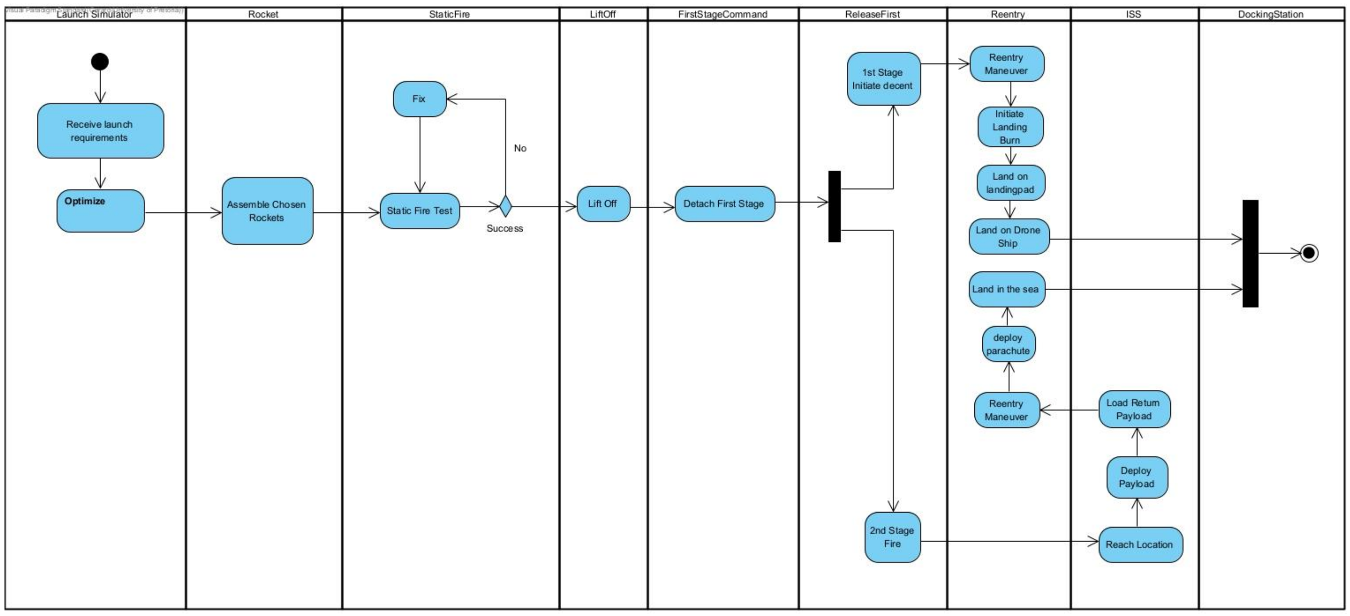
- The engine class was updated so that it can also access the state of the Rocket object.
- The Memento class has been fully implemented in order to properly store the state and configuration of a selected Rocket.
- A ReverseCommand has been incorporated into the Command design pattern in order to cater for changing of variables by the user during test launches.
- The DockCommand has been added to the lists of commands to cater for the docking of the Rocket object into the ISS.

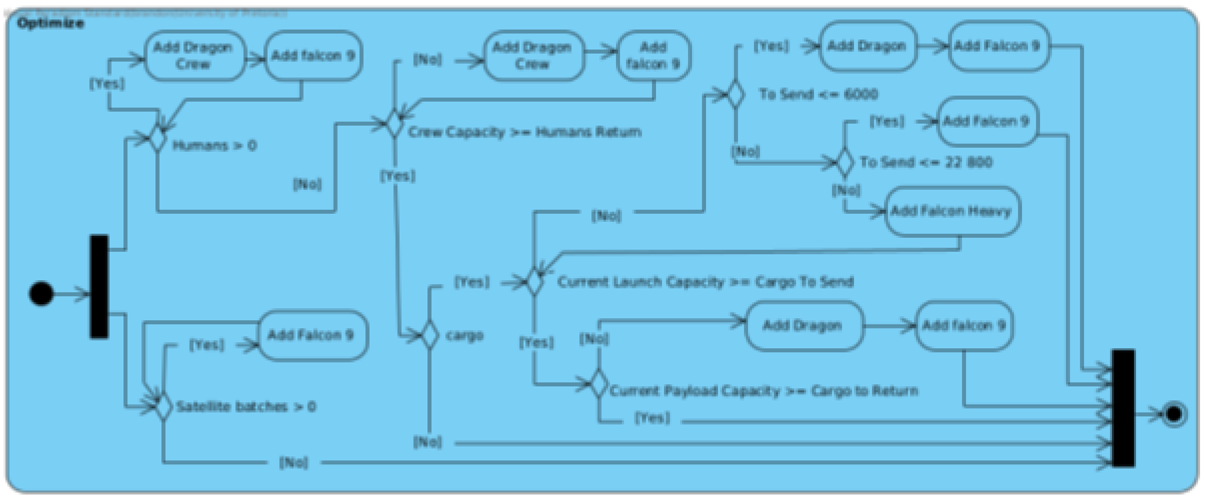
2. Final Design

2.1 Functional Requirements

1. System needs to choose the best rocket and spacecraft configuration based on user inputted variables in order to minimize cost
2. Rockets need to be tested via a static fire before launch.
3. System needs to be able to store actual launch simulations so that they can be later run-in batches.
4. System needs to be able to implement test launches where users can alter their previously inputted variables.
5. First stage of rocket needs to land in the ocean on a drone ship so that it can be reused
6. The altitude of the rocket needs to be known at the different stages of the rocket object.
7. The system needs to have an interface wherein the user can select what they want to send to space.
8. The Rocket configuration needs to communicate with the system when a major event occurs, e.g., successful connection to ISS and successful detachment of stage 1.

2.2 Activity Diagram



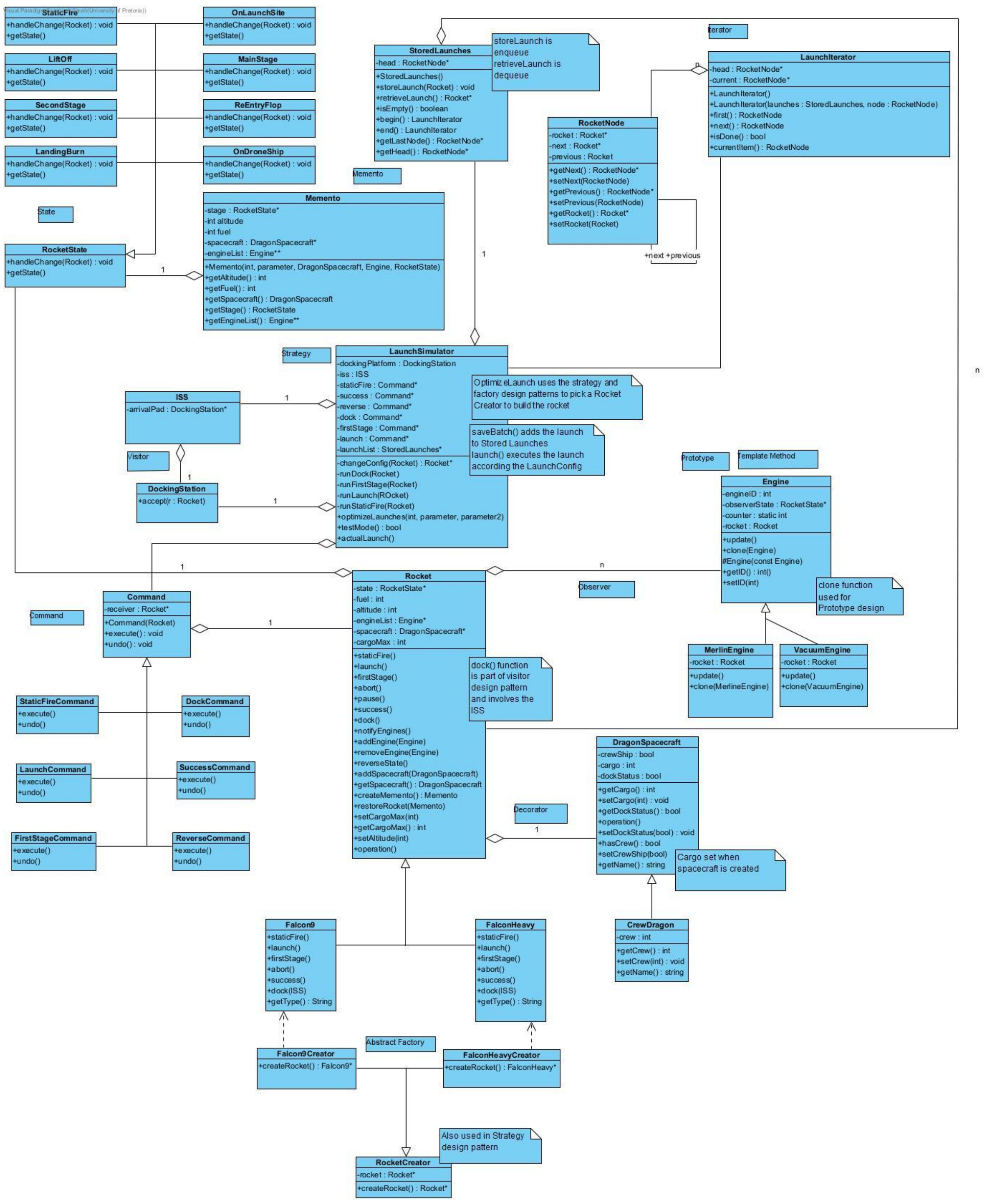


2.3 Design Patterns

These are the design patterns we have chosen in order to address our previously defined functional requirements:

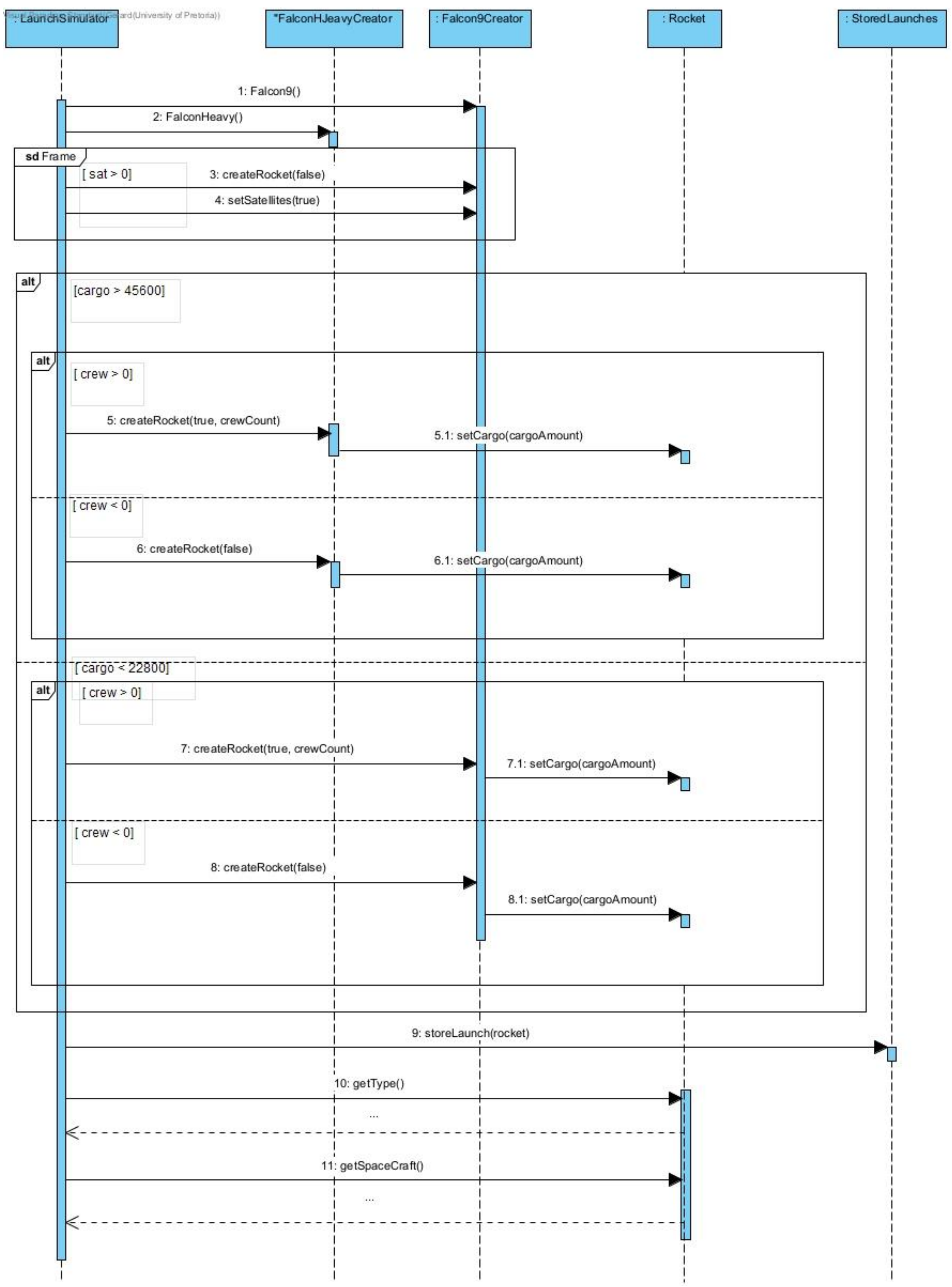
1. The Factory method will be used to construct the different rockets
2. The Decorator method will be used to add the spacecraft and relevant cargo/crew to the rocket configuration
3. The Template method will be used to create the Crew Dragon and Dragon spacecraft from the abstract class Spacecraft.
4. Observer method will be used on the engines to be updated/notified when a rocket changes its state.
5. The Memento design pattern will be used in order to store and launch the batches of launch configurations
6. The Iterator design pattern will be used in order to run single launches stored in a batch.
7. The Command pattern will be used to control different phases of the launch process.
8. The State design pattern is used to indicate and interact with the state of the rocket in the different phases of its launch process.
9. The Prototype design pattern is used to create and clone the engines of the rockets.
10. The Strategy pattern will be used in order to select an appropriate rocket creator for the launch configuration class.
11. The Visitor design pattern will be used in order to dock the relevant Rocket object into the ISS.

2.5 Class Diagram & Design of Classes

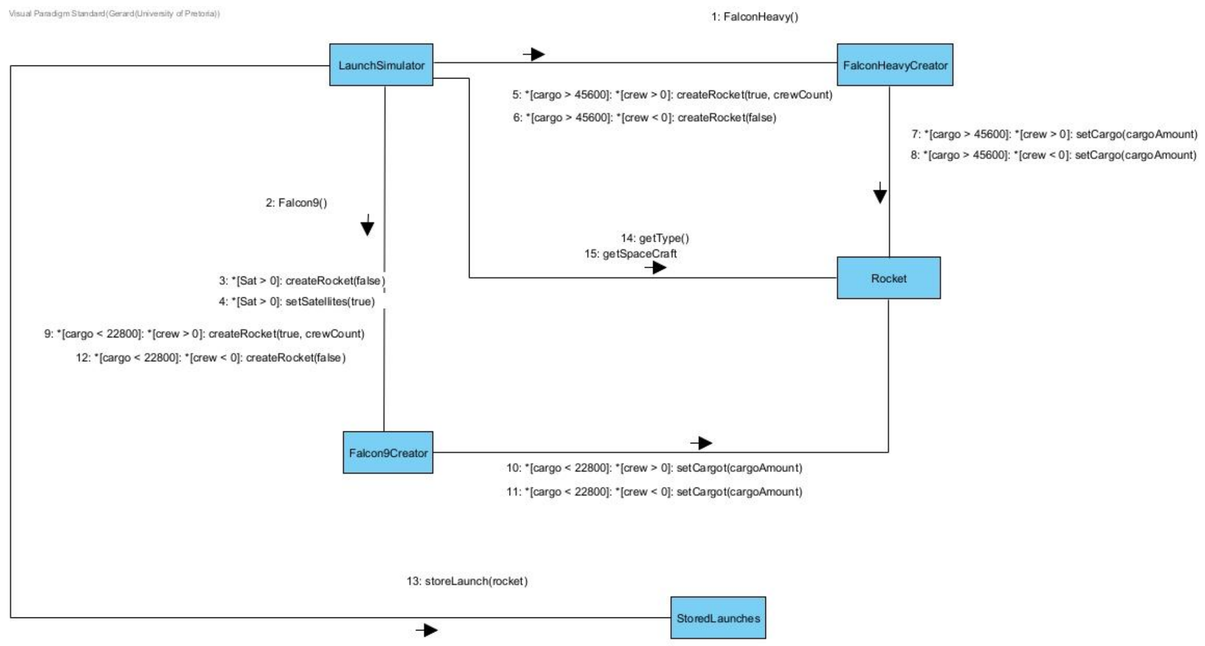


2.6 Sequence and Communication Diagrams

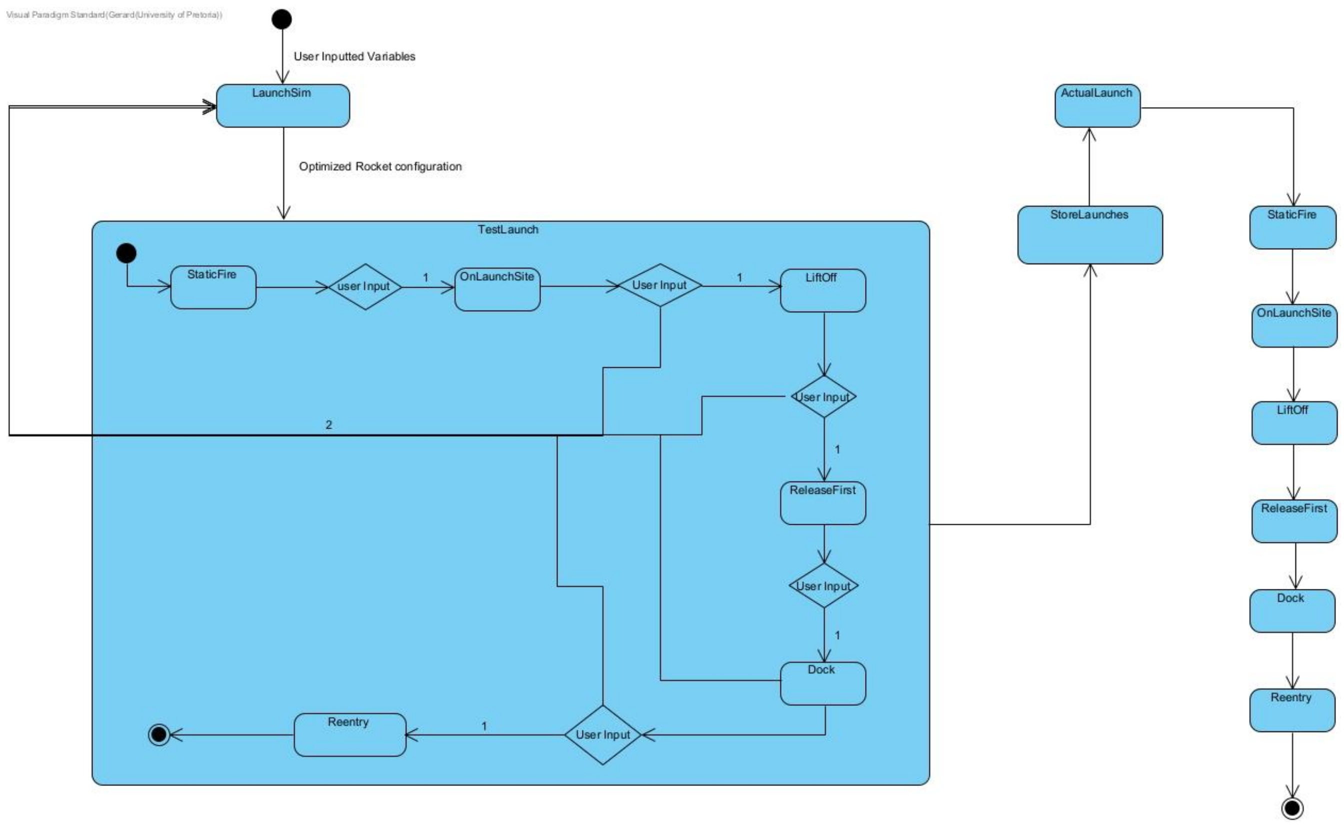
Sequence diagram of optimizeLaunches():



<u>Communication diagram of optimizeLaunches():</u>

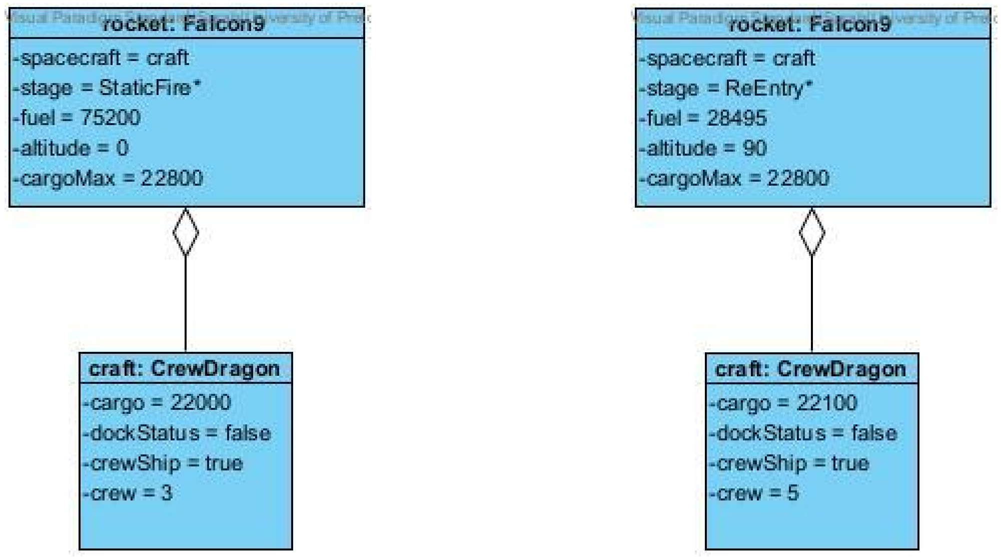

Visual Paradigm Standard(Gerard(University of Pretoria))
LaunchSimulator
FalconHeavyCreator
Rocket
Falcon9Creator
StoredLaunches
1: FalconHeavy()
5: *[cargo > 45600]: *[crew > 0]: createRocket(true, crewCount)
6: *[cargo > 45600]: *[crew < 0]: createRocket(false)
7: *[cargo > 45600]: *[crew > 0]: setCargo(cargoAmount)
8: *[cargo > 45600]: *[crew < 0]: setCargo(cargoAmount)
2: Falcon9()
14: getType()
15: getSpaceCraft
3: *[Sat > 0]: createRocket(false)
4: *[Sat > 0]: setSatellites(true)
9: *[cargo < 22800]: *[crew > 0]: createRocket(true, crewCount)
12: *[cargo < 22800]: *[crew < 0]: createRocket(false)
10: *[cargo < 22800]: *[crew > 0]: setCargot(cargoAmount)
11: *[cargo < 22800]: *[crew < 0]: setCargot(cargoAmount)
13: storeLaunch(rocket)

2.7 State Diagram



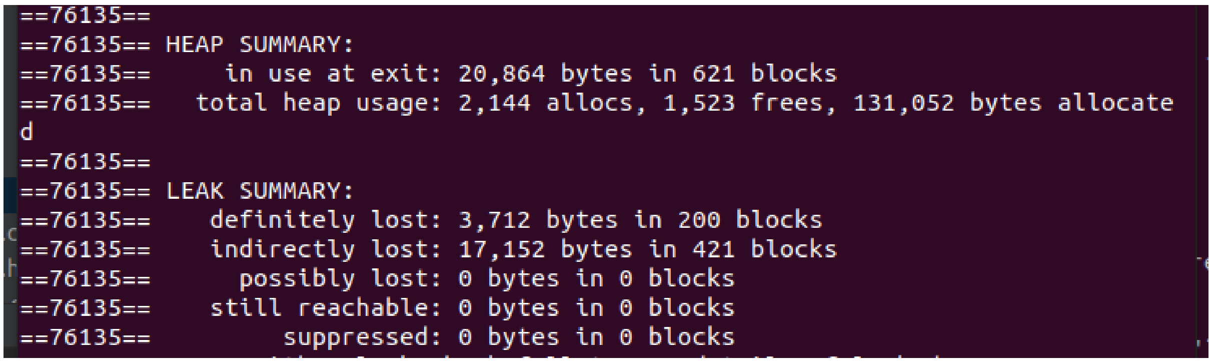
2.8 Object Diagrams

Object diagrams of a Falcon9 Rocket and Crew Dragon after it has gone through test mode (where the cargo was increased by 100 and crew was increased by 2) and an actual launch.



2.9 Valgrind use

Valgrind was used to check if all memory was deallocated. Memory leaks were detected and resolved.



```
==76135==
==76135== HEAP SUMMARY:
==76135==     in use at exit: 20,864 bytes in 621 blocks
==76135==   total heap usage: 2,144 allocs, 1,523 frees, 131,052 bytes allocate
d
==76135==
==76135== LEAK SUMMARY:
==76135==    definitely lost: 3,712 bytes in 200 blocks
==76135==    indirectly lost: 17,152 bytes in 421 blocks
==76135==      possibly lost: 0 bytes in 0 blocks
==76135==    still reachable: 0 bytes in 0 blocks
==76135==         suppressed: 0 bytes in 0 blocks
```