The 6 Musketeers

COS 214 - Project Report

u20632429 - Chiara Goncalves

u20444738 - Zoe Liebenberg

u20438151 - Jade Peche

u17030553 - Ben Pietersen

u20498510 - Dylan Pietersen

u20430516 - Steven Schormann

THE SIX MUSKETEERS

Table of Contents

Introduction

Our team was given the task to assist Elon Musk and his team to design a system to simulate SpaceX and Starlink. The final product will contribute to the planning and optimising that is put towards their launches. The Six Musketeers need to figure out a way in order to optimise the cost of each launch by choosing the best rocket configuration for the job. There are 2 rockets (the Falcon9 and the Falcon Heavy), 2 spacecrafts (the Crew Dragon and the Dragon) and the Starlink satellites that orbit around the planet in order to provide fast internet to everyone around the world.
Our crew of computer scientists has decided to code a simulation of a rocket. During our studies, we were taught about design patterns and we all knew that they would be the most helpful tool in making our dream of helping *thee Elon Musk* become a reality. As soon as we got our initial design planned out, we got to work.

Google Docs link:
https://docs.google.com/document/d/1oW8Y85WrR50zQwP26fuNFW9QSemY2XdseO5r_AkQBUw/edit?usp=sharing

GitHub link:
https://github.com/OomBen/COS214-Practical.git

Design Pattern Summary

A description of how each design pattern is implemented and how it is applied to address the functionality that is required by the system.

Design Pattern	Description
Facade	Create a unified interface for the user.
Decorator	Dragon capsule connection to rocket.
Composite	Individual parts of the rocket (engines, cores, ...) connect to the final rocket.
Observer	Users of Satellites (on the ground) observe changes to Satellites.
Mediator	Communication Network between Satellites.
Builder	Builds Rocket using Factory Methods.
Factory Method	Satellite Construction / Engine Construction / Core Construction.
State	Starlink Satellite Orbit state, Dragon capsule Orbit/Dock/etc state, Flight Status (during simulation).
Memento	Store the rocket/simulation state.
Prototype	Clones satellite to make payload batch in single construction.

Facade

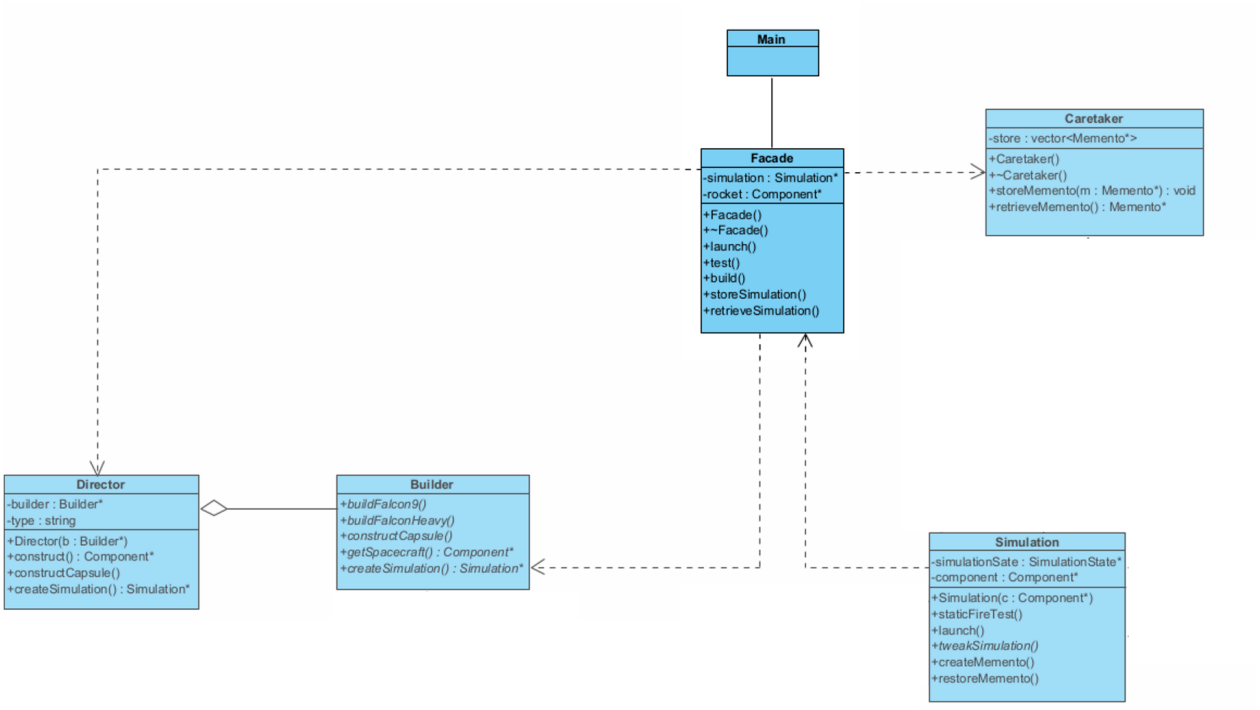
Application:

The Facade design pattern creates a unified interface for the user to interact with the classes. This ends up hiding the complexities of the system from the user. The main class does not need to pass any sort of parameters to the functions that are called which makes the class much simpler and easier to understand. All the client (main) requests are delegated to the appropriate subsystem objects by the Facade because the Facade knows which subsystem classes are responsible for a request. The main, and thus the user, can easily access the system through the Facade.

Participants:

Facade:	Facade
Subsystem classes:	All of the other patterns

UML Diagram:



Builder

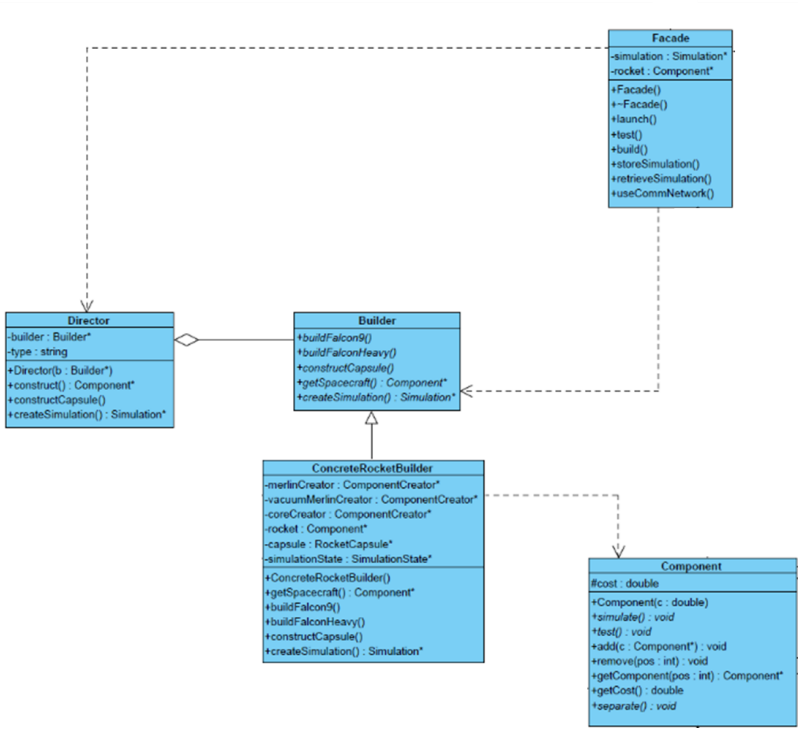
Application:

The Builder design pattern was used as it allows for breaking down large complex structures (such as a complete rocket system) into individual steps, some of which are specialized, optional or omitted according to the specified application. The builder mitigates the need for complicated constructors requiring many parameters. It does so by initializing the Director with all the necessary specifications (in the form of user input prompts provided by the Facade). The Director then calls each component's constructor based on the given requirements and adds individual components together in order to build a complete rocket.

Participants:

Builder:	Builder
ConcreteBuilder:	ConcreteRocketBuilder
Director:	Director
Product:	Component

UML Diagram:



Composite

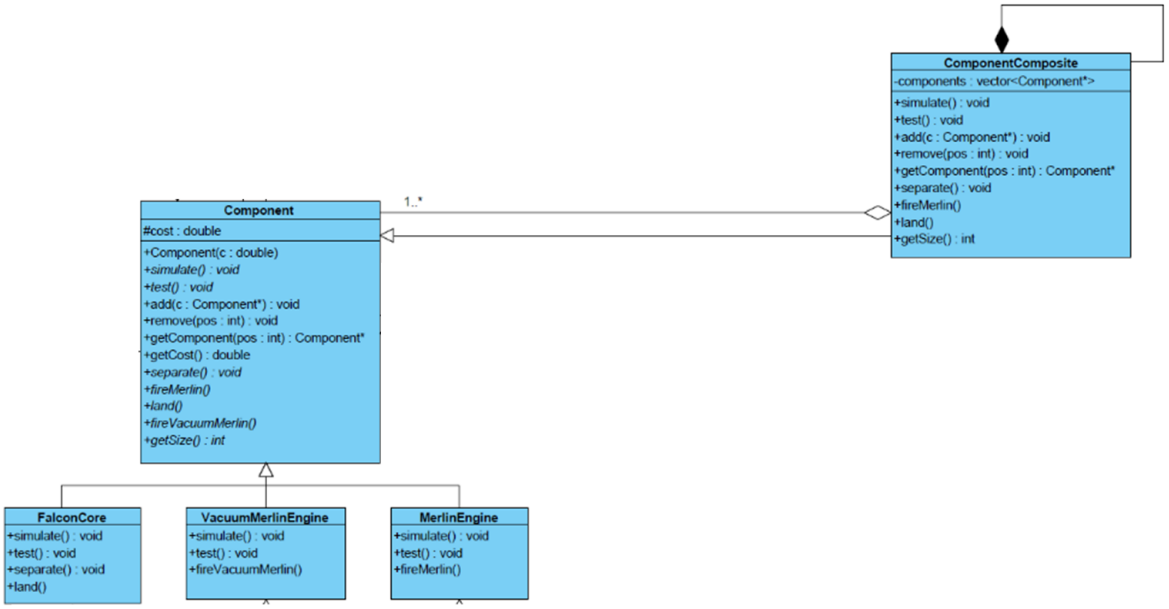
Application:

The Composite design pattern allows the Facade class to treat both individual objects and compositions of objects in the same manner. This was done in order to be able to test and simulate each individual component, as well as testing and simulating compositions of components. This is similar to construction practice at NASA, ULA and most other space industries, each component is tested and verified individually and then the system which contains the component is tested and verified and this effect cascades up to the entire launch system.

Participants:

Component:	Component
Leaf:	FalconCore
Composite:	ConcreteComposite
Client:	Facade

UML Diagram:



Decorator

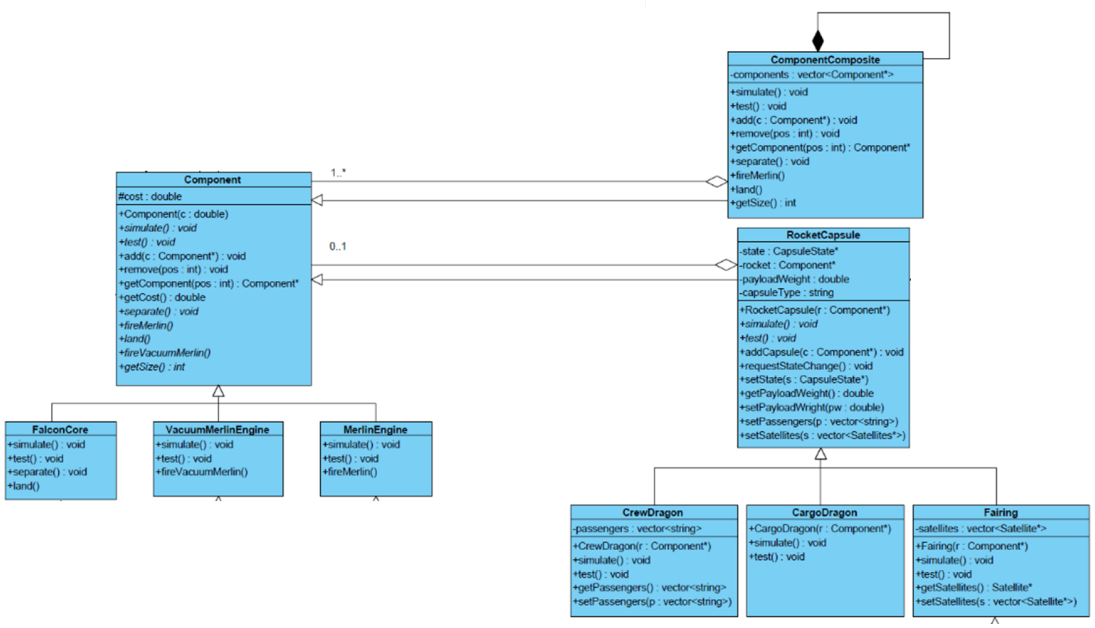
Application:

The Decorator design pattern was used for the pairing of a rocket core (either a Falcon 9 or Falcon Heavy) with a single function to a payload which can have various functions. This allows each rocket core to have extended functionality based on the specified subclass which is attached to it (i.e Crew Dragon Capsule, Cargo Dragon Capsule or a Starlink satellite batch stored in a Fairing). The Decorator pattern allows the rocket component to have specializations, these are defined by each 'decoration' class.

Participants:

Component:	Component
ConcreteComponent:	FalconCore
Decorator:	RocketCapsule
ConcreteDecorator:	CrewDragon, CargoDragon, Fairing

UML Diagram:



Factory Method

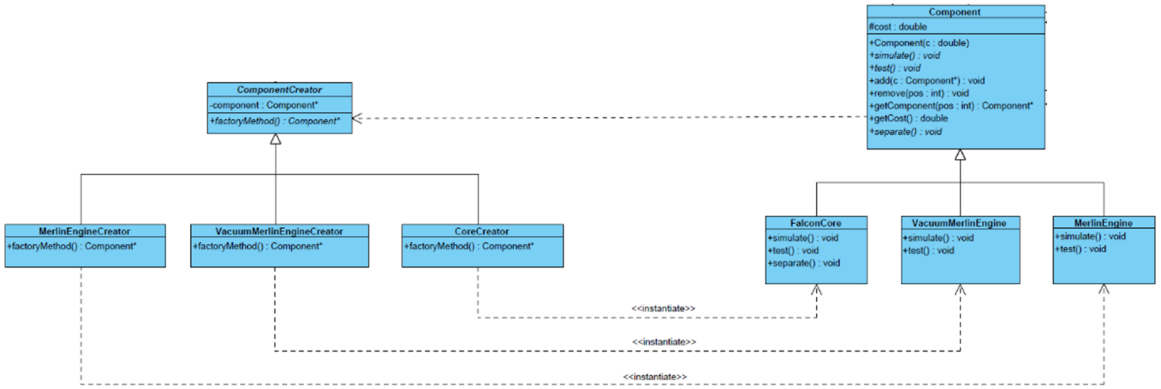
Application:

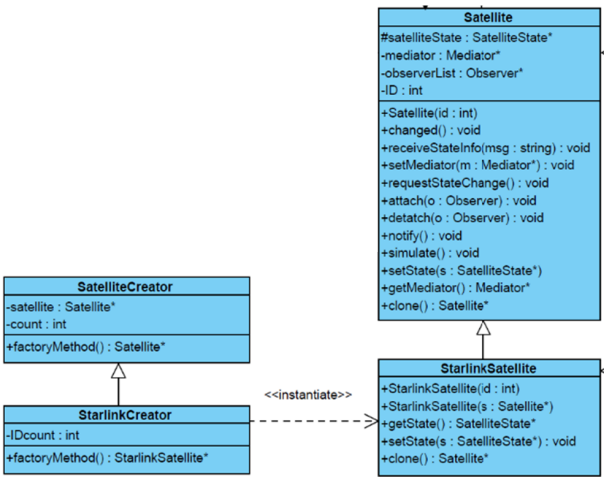
The Factory Method design pattern was implemented in order for the system to create Merlin Engines, Vacuum Merlin Engines and Cores. The subclasses of ComponentCreator will decide which class to instantiate. Instantiation is deferred to subclasses. This Factory Method allows the system to easily create a component.
A second Factory Method is used in conjunction with a prototype to instantiate the Starlink Satellites to easily create multiple Satellites. See the Prototype pattern for more detail.

Participants:

Creator:	ComponentCreator
ConcreteCreator:	MerlinEngineCreator, VacuumEngineCreator, CoreCreator
Product:	Component
ConcreteProduct:	FalconCore, VacuumMerlinEngine, MerlinEngine

UML Diagram:





Prototype

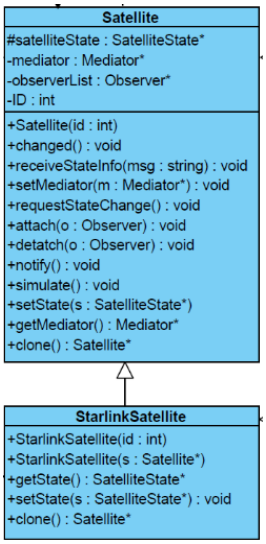
Application:

The prototype design pattern is used to construct multiple instances of a specific satellite object. Since each satellite is relatively basic an object and each satellite is similar to every other satellite they can easily be instantiated by a cloning method. The cloning method is called from the ConcreteRocketBuilder client when it constructs a fairing payload consisting of N (user defined amount) satellites.

Participants:

Client:	ConcreteRocketBuilder
Prototype:	Satellite
ConcretePrototype:	StarlinkSatellite

UML Diagram:



Observer

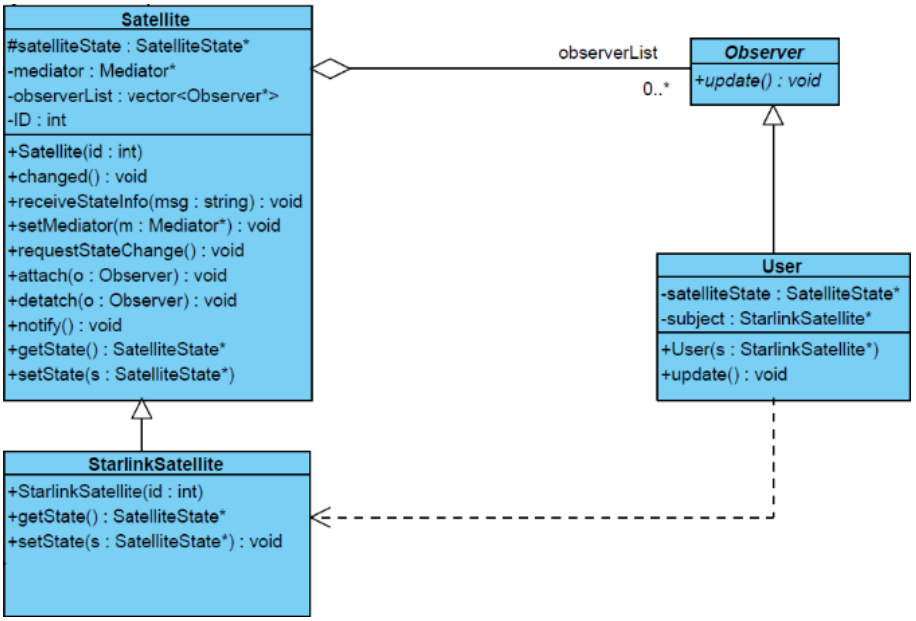
Application:

The Observer design pattern is used for communication between the ground (a User class which acts as a base station) and the satellite. In order to simulate a real world usage case, each Satellite can broadcast to multiple users. If any change in the state of the satellite occurs, all the Users are notified of this change in state.The observer pattern was chosen for this as none of the Users need to be aware of each other and the communication between Satellites and Users is only in one direction when a satellite is in its ‘Broadcasting’ state.

Participants:

Subject:	User
ConcreteSubject:	Satellite
Observer:	Observer
ConcreteObserver:	User

UML Diagram:



Mediator

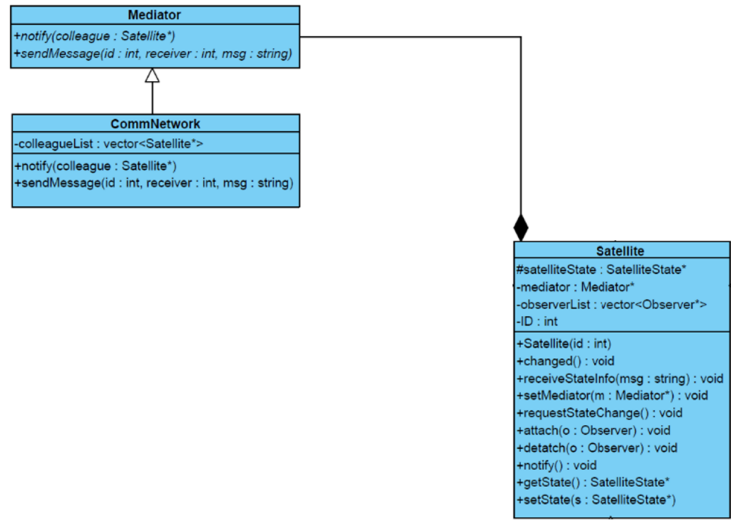
Application:

This pattern handles communication between the satellites. If a satellite changes its state (for whatever reason), it triggers the Mediator class to notify all the other satellites of this change. This pattern was chosen as it greatly simplifies the communication process between satellites by centralising the communication control while removing the need for a satellite object to keep track of any other satellite.

Participants:

Mediator:	Mediator
ConcreteMediator:	CommNetwork
Colleagues:	Satellite

UML Diagram:



State

Application:

There are two applications of the state pattern in our code, the state of the satellite and the state of the capsule.
The Satellite state changes the behaviour of a satellite object allowing full functionality while in the Broadcasting state, limited functionality while in the Online state, and only the ability to change states while in the Offline state.
The Capsule state changes the behaviour of the capsules depending on the stage of the launch procedure. A capsule will have to be in a certain state in order for a launch to be successful.

Participants:

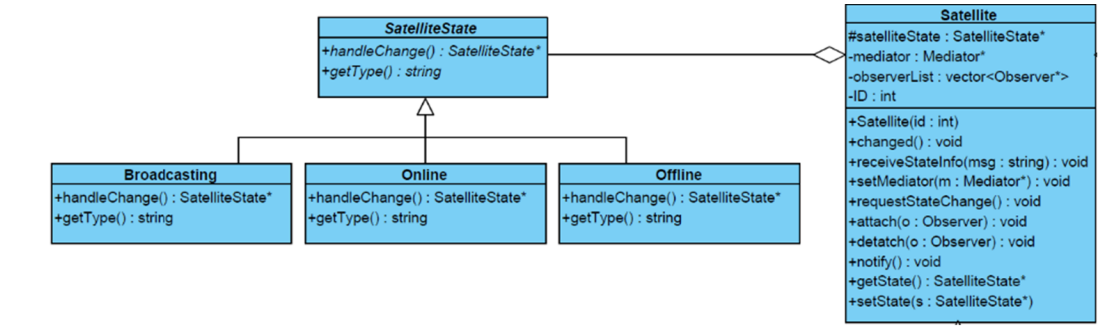
Satellite:

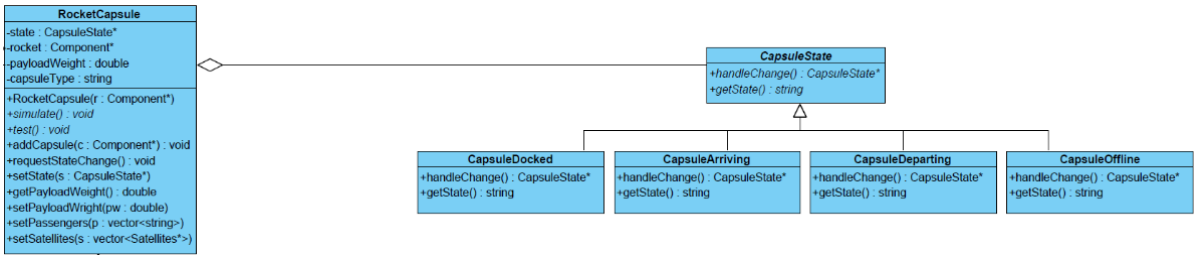
State:	SatelliteState
ConcreteState:	Broadcasting, Online, Offline
Context:	Satellite

Capsule:

State:	CapsuleState
ConcreteState:	CapsuleDocked, CapsuleArriving, CapsuleDeparting, CapsuleOffline
Context:	Satellite, RocketCapsule

UML Diagram:





Memento

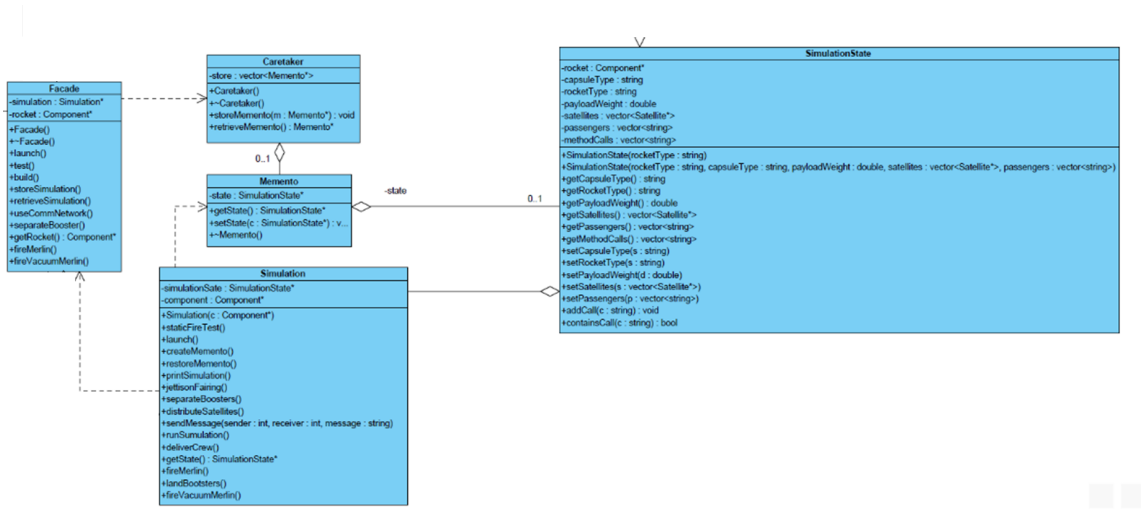
Application:

The memento design pattern allows the system to store actual launch simulations and run it in batches. Encapsulation is still complied with in the project. A launch simulation's internal state will be captured and externalised in order for this state to be restored to this state at a later stage. Memento promotes a rollback to full object status.

Participants:

Memento: Memento
Originator: SimulationState
Caretaker: Caretaker

UML Diagram:



Classes Summary

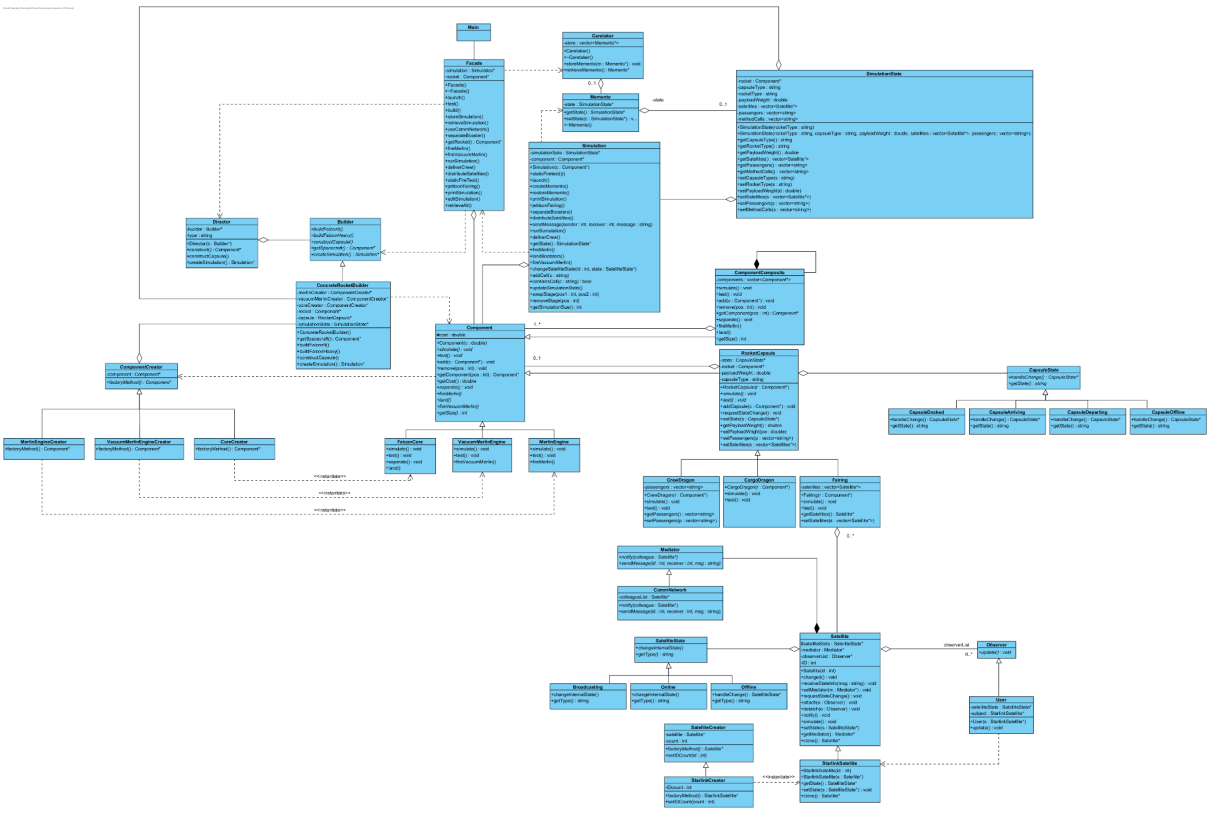
Design pattern	Participants	Class
Facade	Facade	Facade
	Subsystem Classes	All other classes
Decorator	Component	Component
	ConcreteComponent	FalconCore
	Decorator	RocketCapsule
	ConcreteDecorator	CrewDragon, CargoDragon, Fairing
Composite	Component	Component
	Leaf	FalconCore
	Composite	ConcreteComposite
	Client	Main
Observer	Subject	Subject
	ConcreteSubject	Satellite
	Observer	Observer
	ConcreteObserver	User
Mediator	Mediator	Mediator
	ConcreteMediator	CommNetwork
	Colleagues	Satellite
Builder	Builder	Builder
	ConcreteBuilder	ConcreteRocketBuilder
	Director	Director
	Product	Component
Factory Method	Creator	ComponentCreator

	ConcreteCreator	MerlinEngineCreator, VacuumEngineCreator, CoreCreator
	Product	Component
	ConcreteProduct	FalconCore, VacuumMerlinEngine, MerlinEngine
Prototype	Prototype	Satellite
	ConcretePrototype	StarlinkSatellite
	Client	ConcreteRocketBuilder
State	State	SatelliteState, CapsuleState
	ConcreteState	Broadcasting, Online, Offline, CapsuleDocked, CapsuleArriving, CapsuleDeparting, CapsuleOffline
	Context	Satellite, RocketCapsule
Memento	Memento	Memento
	Originator	MementoState
	Caretaker	Caretaker

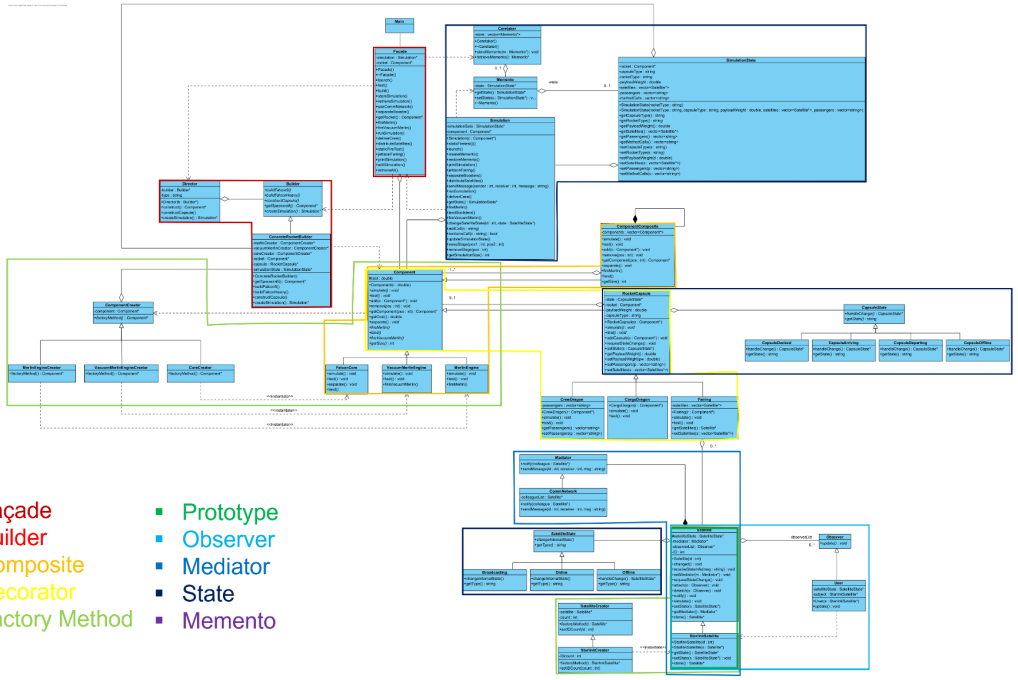
Diagrams

*For larger PDF diagrams see attached Diagrams folder

Class Diagrams



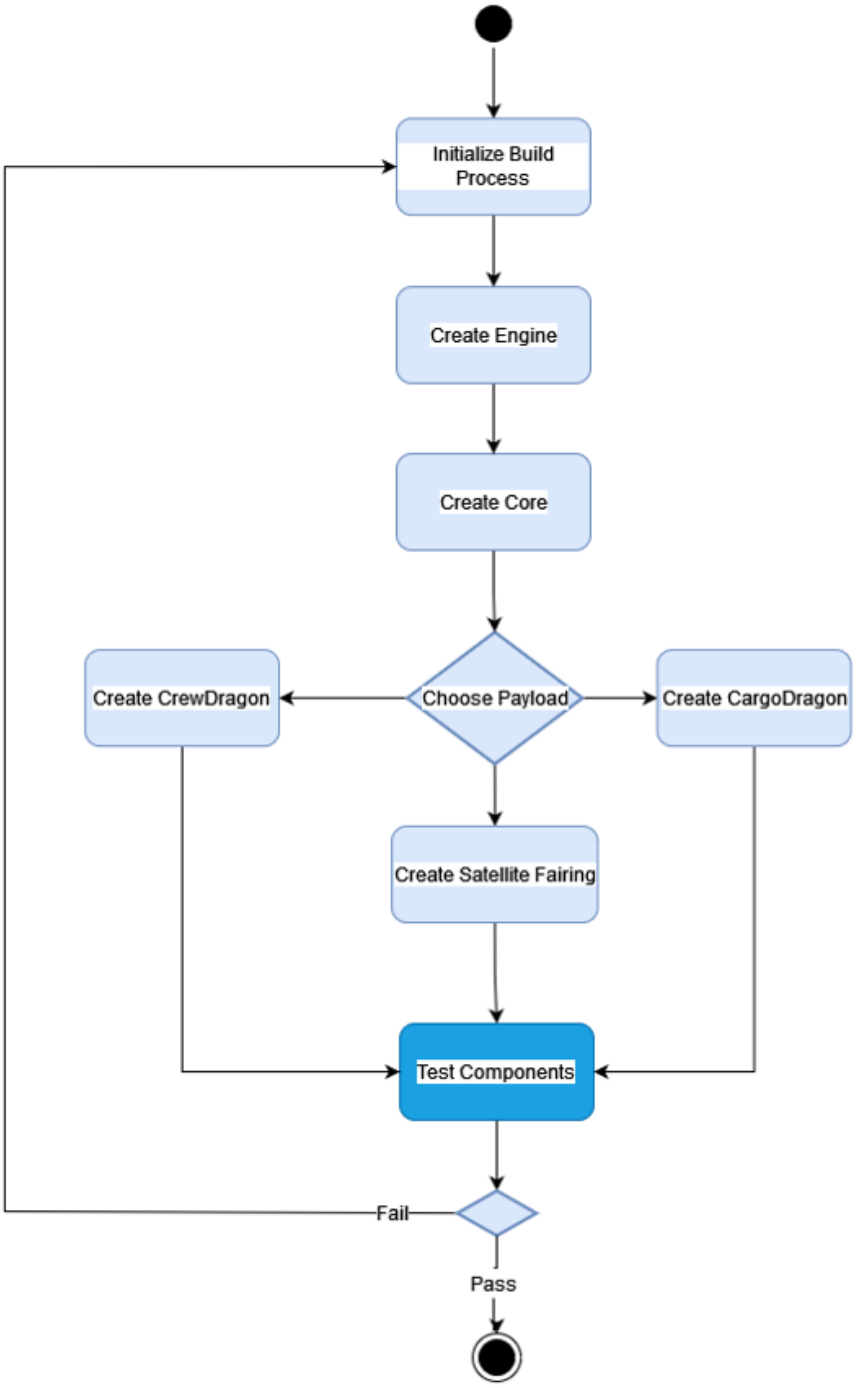
Class Diagram with the different design patterns highlighted



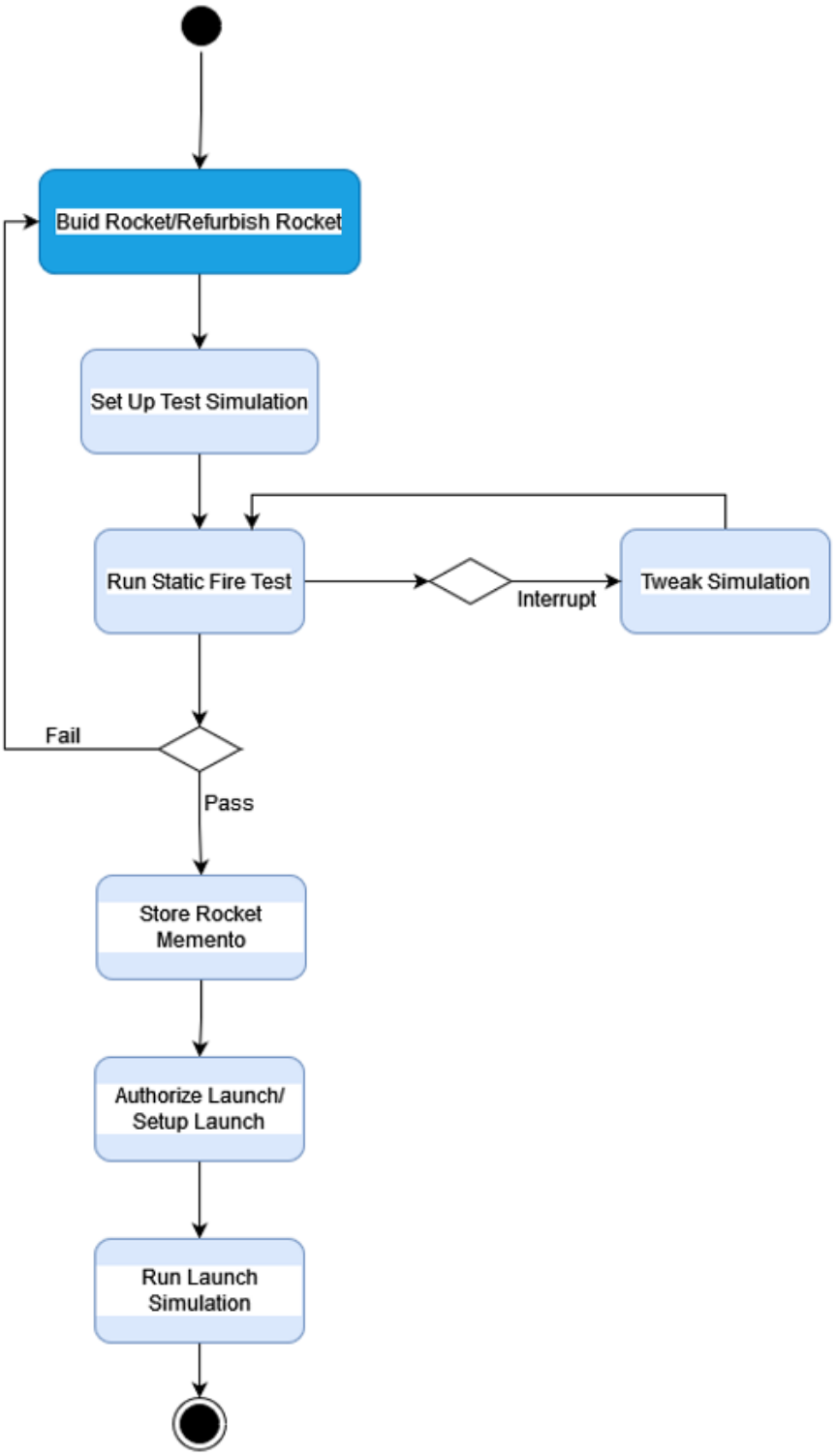
- Façade
- Builder
- Composite
- Decorator
- Factory Method
- Prototype
- Observer
- Mediator
- State
- Memento

Activity Diagrams

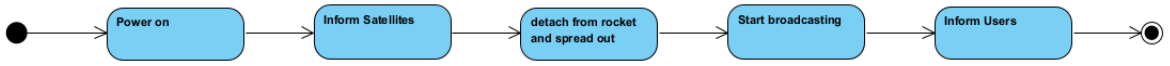
Construction Process



Testing and Simulation Process

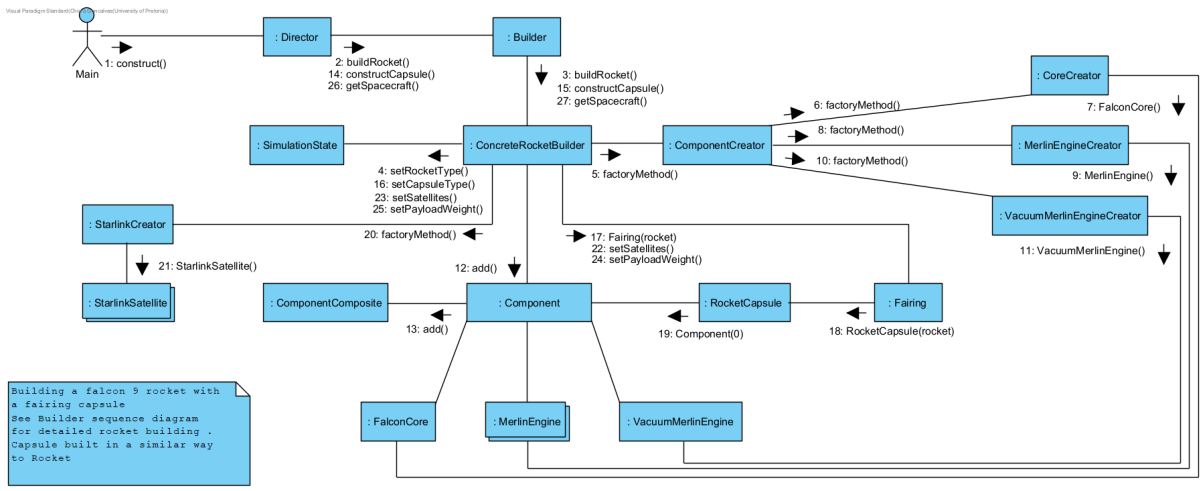


Satellites in space

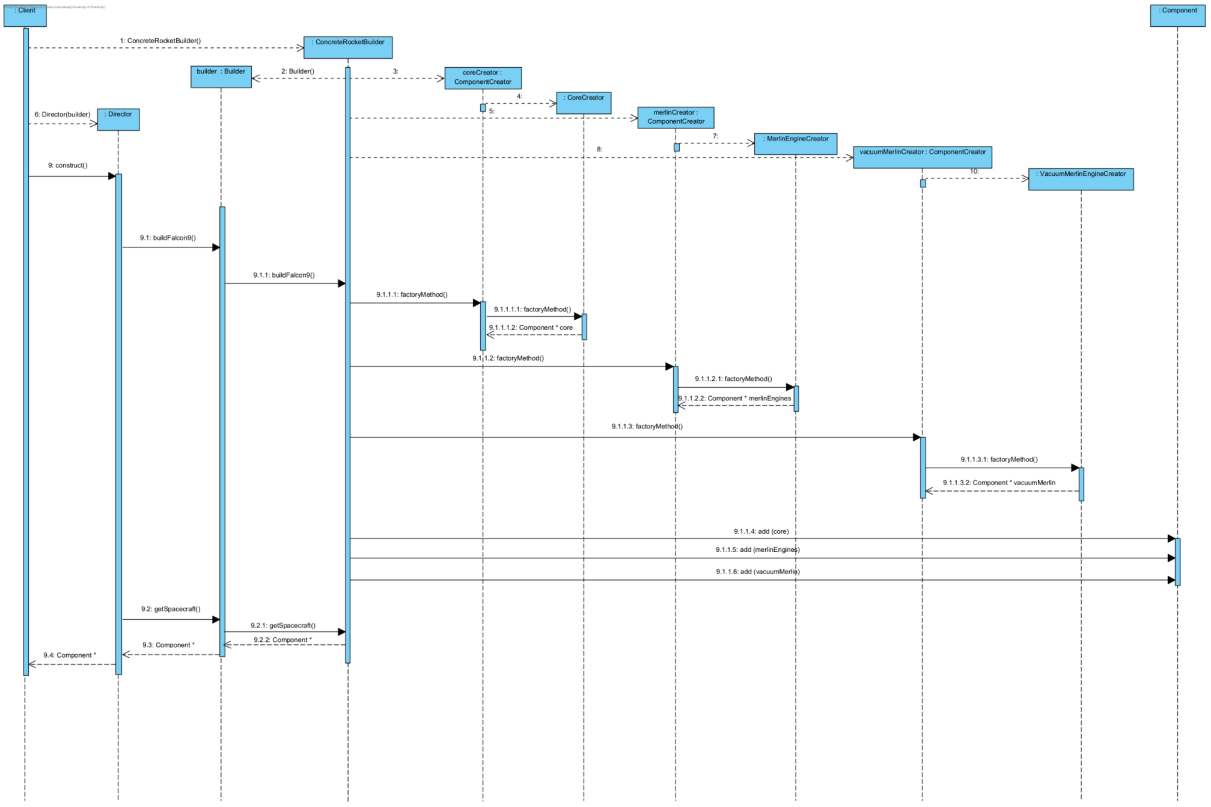


Sequence and Communication Diagrams

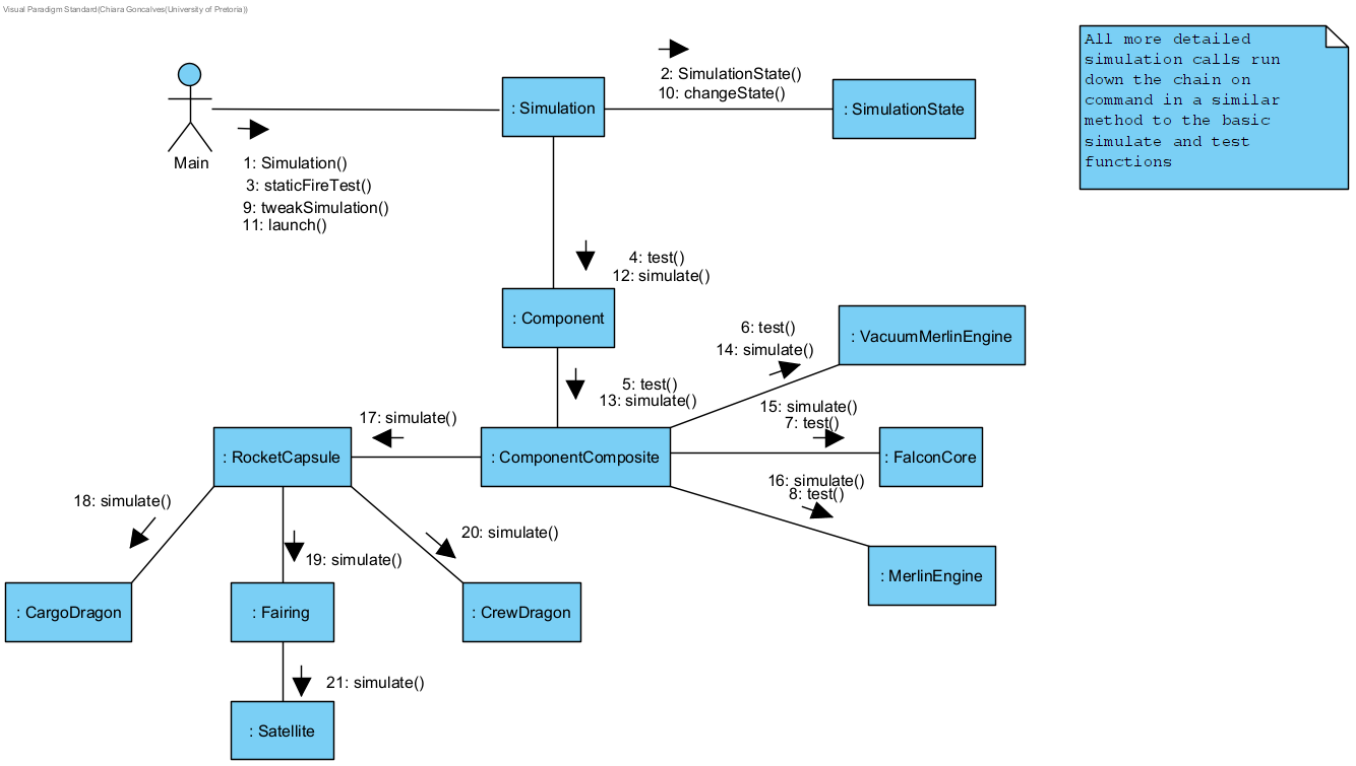
Building Rocket and Capsule - Communication Diagram



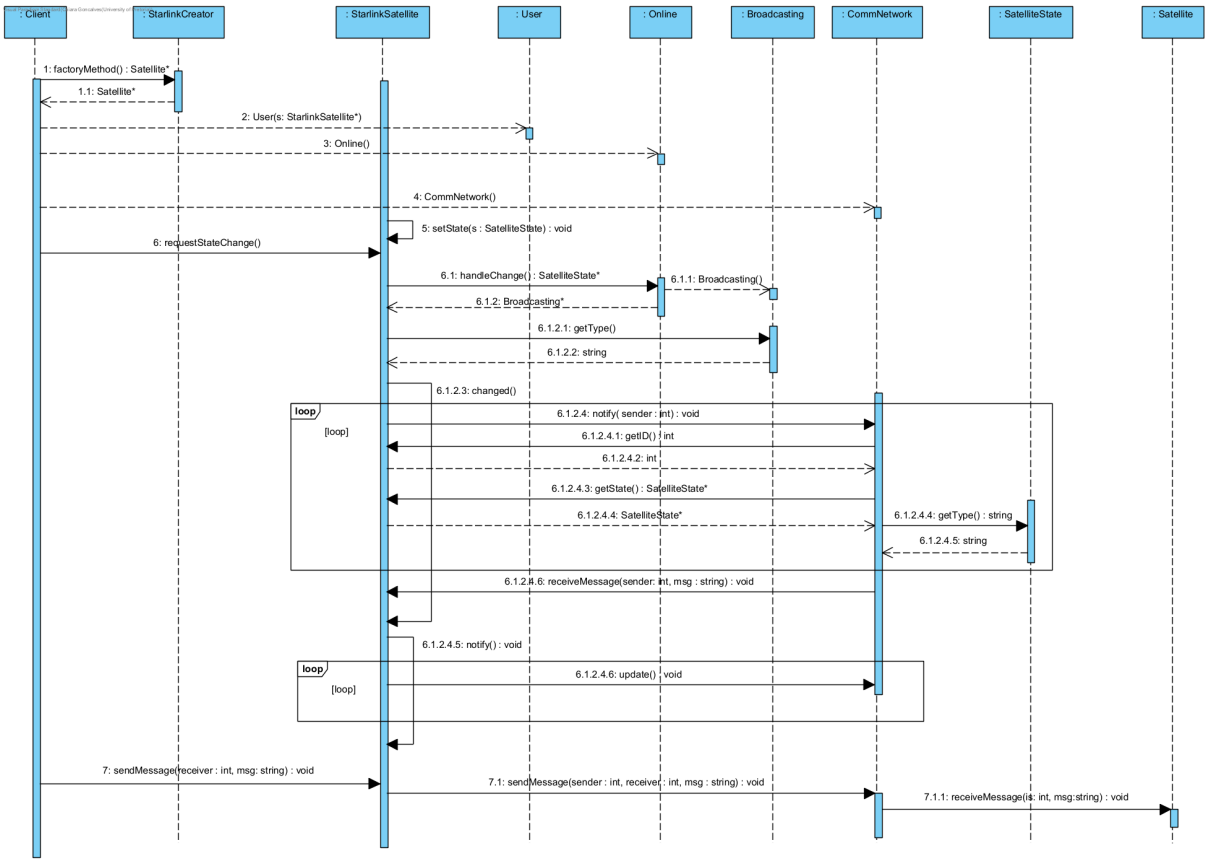
Building Rocket - Sequence Diagram



Launch - Communication Diagram

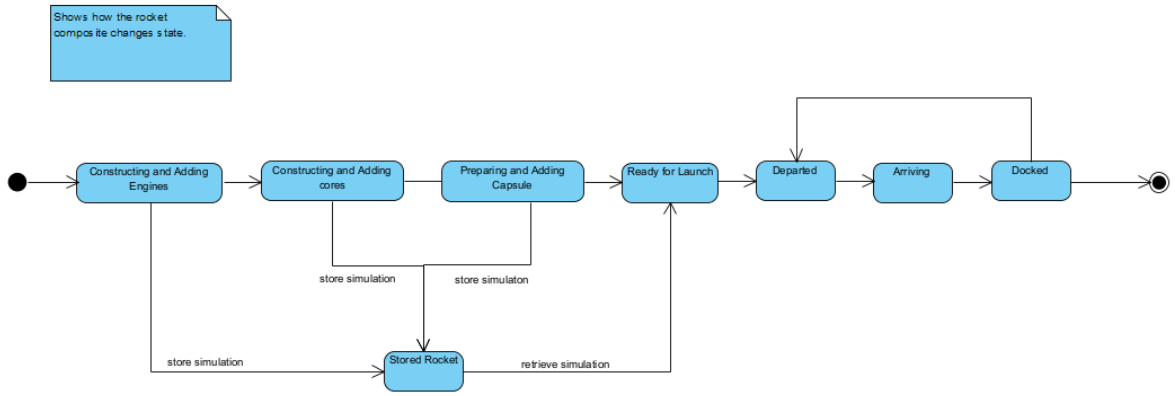


Satellite Communication - Sequence Diagram

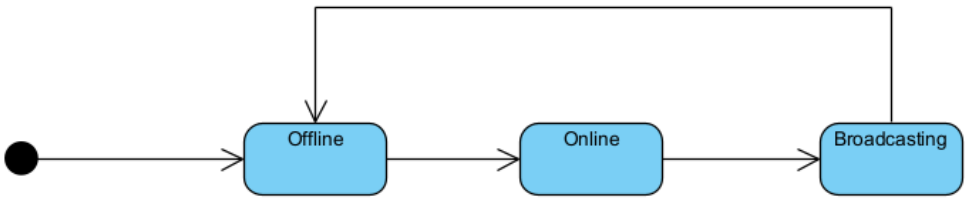


State Diagrams

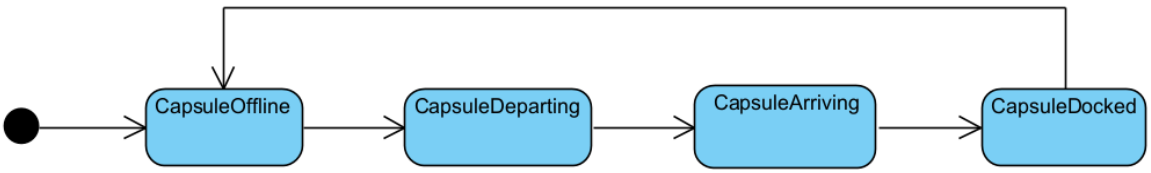
Rocket Composites state changes



Satellite state changes

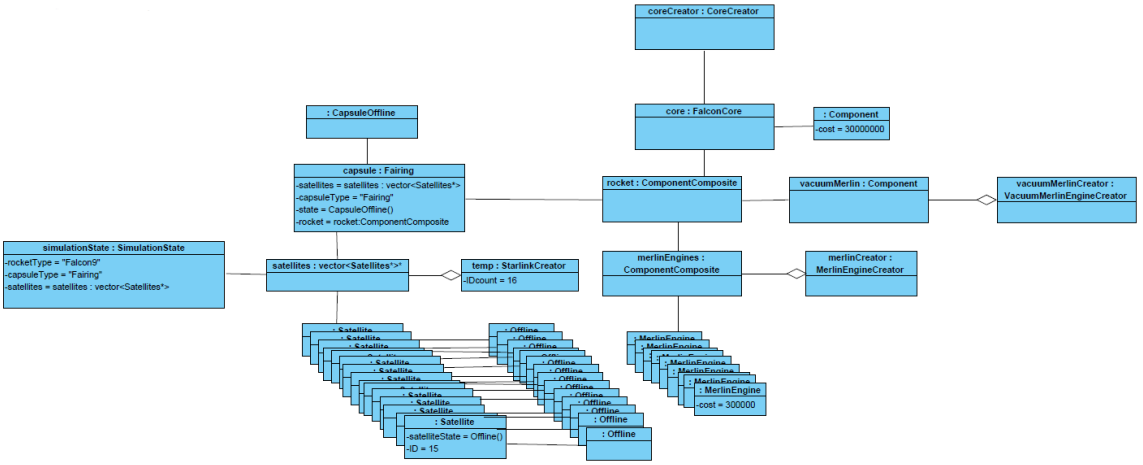


Capsule state changes



Object Diagrams

Before launch



After launch