COS214 Semester-Assignment 2021

BGB
Final Report

Group Members

Pair 1:	Pair 2:	Pair 3:
Stefan van Schoor **u20573783**	Dylan Kapnias **u18108467**	Tabitha Jemwa **u19294418**
Kyle Smith **u20435992**	Daniel Burgess **u18055215**	Larisa Botha **u20522623**

Functional requirements

- Simplify the designing and creation of rockets, satellites by hiding the creation logic.
    - ❖ Factory Method
- Ensure the ability to create multiple engines specifically Merlin Engines
    - ❖ Prototype
- Allow Rockets to support a multistage launch.
    - ❖ State
- Ensure a spacecraft's capability to convey and/or manage cargo and/or crew members.
    - ❖ Iterator
    - ❖ Strategy
- Provide a launch interface equipped with predefined commands to guide the user through a test and launch phase.
    - ❖ Command
    - ❖ Chain of Responsibility
- The ability to interrupt a test launch sequence.
- Save and store previous launch simulations to evaluate or rerun at a later stage.
    - ❖ Memento
- Provide starlink satellites the functionality to communicate both with the earth and among themselves.
- Provide starlink satellites the capability to spread out around the earth.
- Provide a static fire capability to inspect engine fire up.
    - ❖ Template Method
- Guarantee control over the rocket engines
    - ❖ Mediator

- Ensure that the engines can recover after overheating
    - ❖ Template Method

- Impart the capability for a Dragon to dock at the International Space Station and unload the cargo and/or crew

Our 10 Design Patterns

1. Strategy
2. State
3. Chain of Responsibility
4. Command
5. Iterator
6. Memento
7. Prototype
8. Factory Method
9. Template Method
10. Mediator

Implementation of system requirements

Static Fire capability:

The static fire functionality should inspect all relevant components of every engine in the targeted rocket. Therefore the Template Method design pattern is used to run each engine's component check procedure, since they vary respectively. Heavily abstracted primitive methods were implemented to represent these procedures.

The implementation:

A static fire of a certain rocket will initiate check procedures on every engine in each merlin core of the rocket. These check procedures include temperature and oil checks followed by starting the engine. Temperature checks compare the current temperature, near the particular motor location, to the prescribed danger-temperature of the engine type. Oil checks inspect every oil container and whether it is sufficiently sealed. Starting the engine may reveal any other issues that might surface before the actual launch commence. These procedures are accounted for by detailed cout statements in their corresponding primitive methods.

In conclusion the general engine check procedure, required by the static fire capability, is broken down into smaller engine specific procedures namely: checkTemperature, checkOil and startEngine.

-Larisa Botha

Simulation:

It should be possible to run a simulation by either selecting a previously saved simulation or building a new simulation. During the building process it should also be possible to do the building in a test mode or not. After the building process has been finished, it should also be possible to save the created simulation for later use.
During the selection process, it should be possible to either run a single simulation or run multiple simulations.

Due to the fact that there are two types of simulations that can be implemented, a Chain of Responsibility Design Pattern was used to determine which object was to be used for the simulation.

When building a new simulation in test mode, due to the need to be able to roll back the previous decision, a Memento Design Pattern was used to save the simulation's state, which has been abstracted into a
separate class, before a decision was made and reinstate the previous state after a decision if so desired.

Due to the simulation being a series of method calls that act upon a singular spacecraft, a Command Design Pattern was used to help simplify the implementation by coupling all the method calls with the singular
Spacecraft.

-Dylan Kapnias, Daniel Burgess

Multiple different stages and states of a Falcon rocket:

To be able to address the different stages of a falcon and to simulate the multiple states including launched, separated, returned and idle the state design pattern was used. We created four different states for every stage of the launch and the states work in a circular fashion. This means that when a falcon has returned to earth it will move to the returned state and then to idle. When the falcon then has to be re-used it will be attached to a new Dragon or Satellite and will be moved to the launched state once again. The state pattern was chosen as the best option because of the logical flow of the different states and a falcon's state can easily be retrieved.

-Stefan van Schoor

Satellite Communication Requirement:

Satellite communication was done using a pseudo-mediator pattern allowing for the direct one-to-one communication between satellites and Mission control on the ground.
The satellites communicate with Mission Control once they were in position with their Keplerian Coordinates. The Keplerian Coordinates system uses a combination of parameters to approximate the orbit of a satellite at some particular moment in time. Furthermore these Coordinates were useful in determining whether satellites' trajectories were going to cross leading to possible collisions and so they could reposition themselves if needed and also indicate their new position to mission control. It was also possible for mission control to send a reposition request to a specific satellite if required. The above mentioned system thus allowed for a two-way communication protocol between satellites and mission control.

-Kyle Smith

Manufacturing of rockets, spacecrafts and satellites functionality:

The system must be able to manufacture spacecrafts in this case CrewDragon and Dragon, rockets in this case Falcon9 and FalconHeavy and satellites. To address this functionality we used the Factory Method. The Factory Method is one of the creational patterns therefore it allows you to easily create objects. The Factory Method allowed for the provision of a singular uniform interface for creating all possible spacecraft required by the system, thus it was convenient.

-Tabitha Jemwa

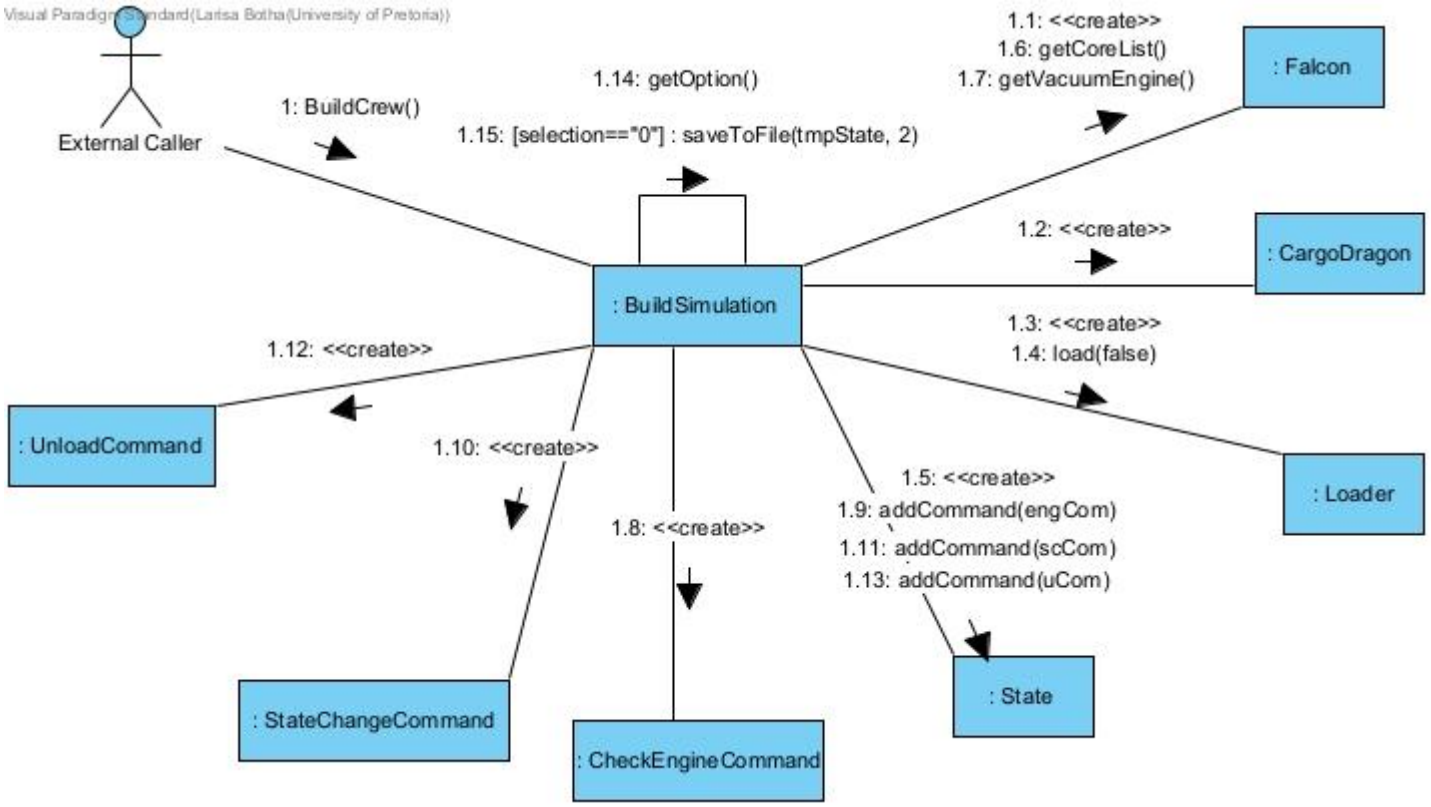
Managing Crew-members and Cargo

Since several Crew members and Cargo can be carried by the rockets a simple interface for managing them is required. A list of crew is implemented as a linked list such that crew members can be ordered in a hierarchical manner and a list of cargo is implemented as a regular vector for simplicity. Since it was decided upon these respective data structures, managing and attendance checking can become quite complicated. Therefore the iterator design pattern was implemented to allow my teammates as well as users to control these lists efficiently and effectively.

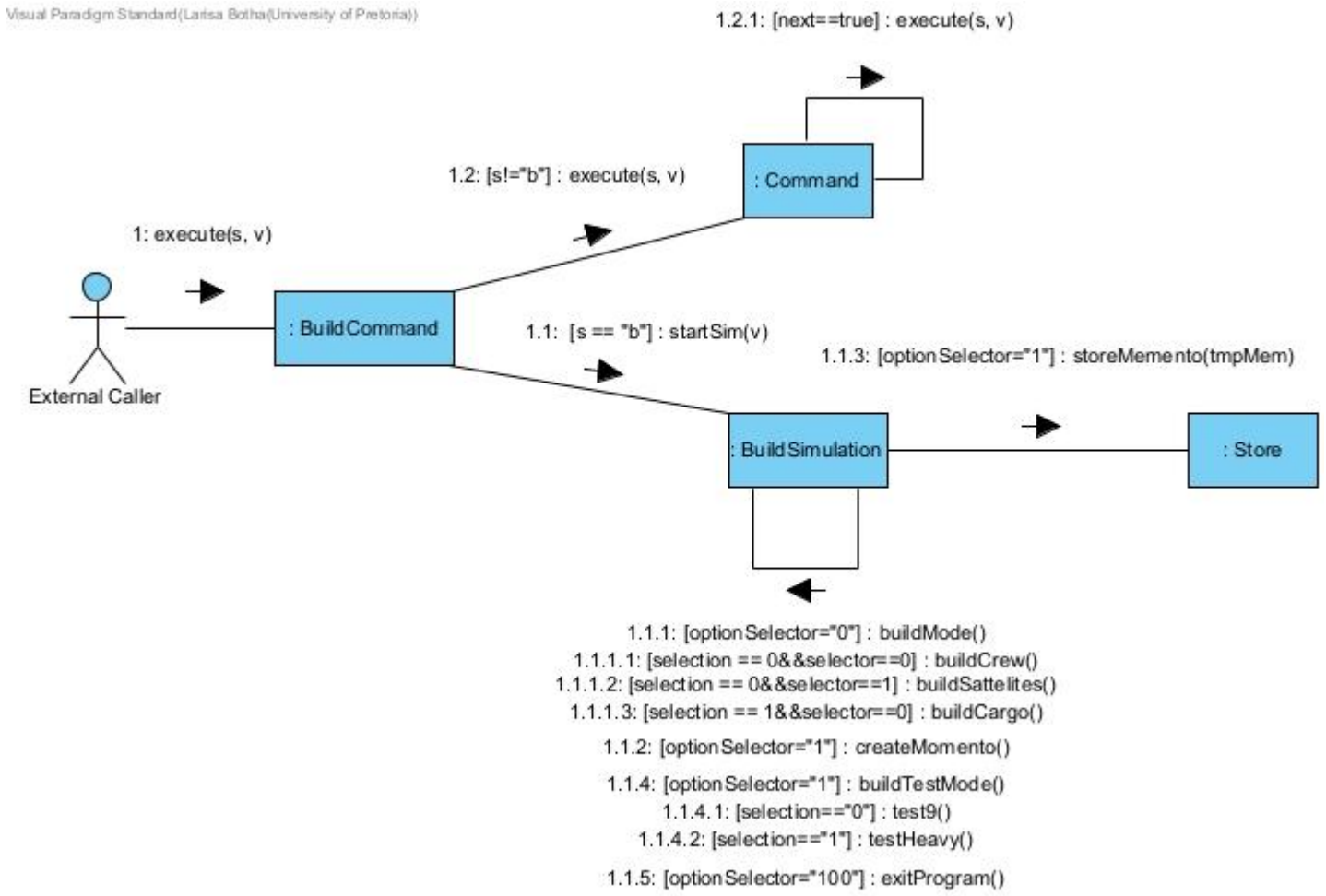
-Larisa Botha

Communication Diagrams

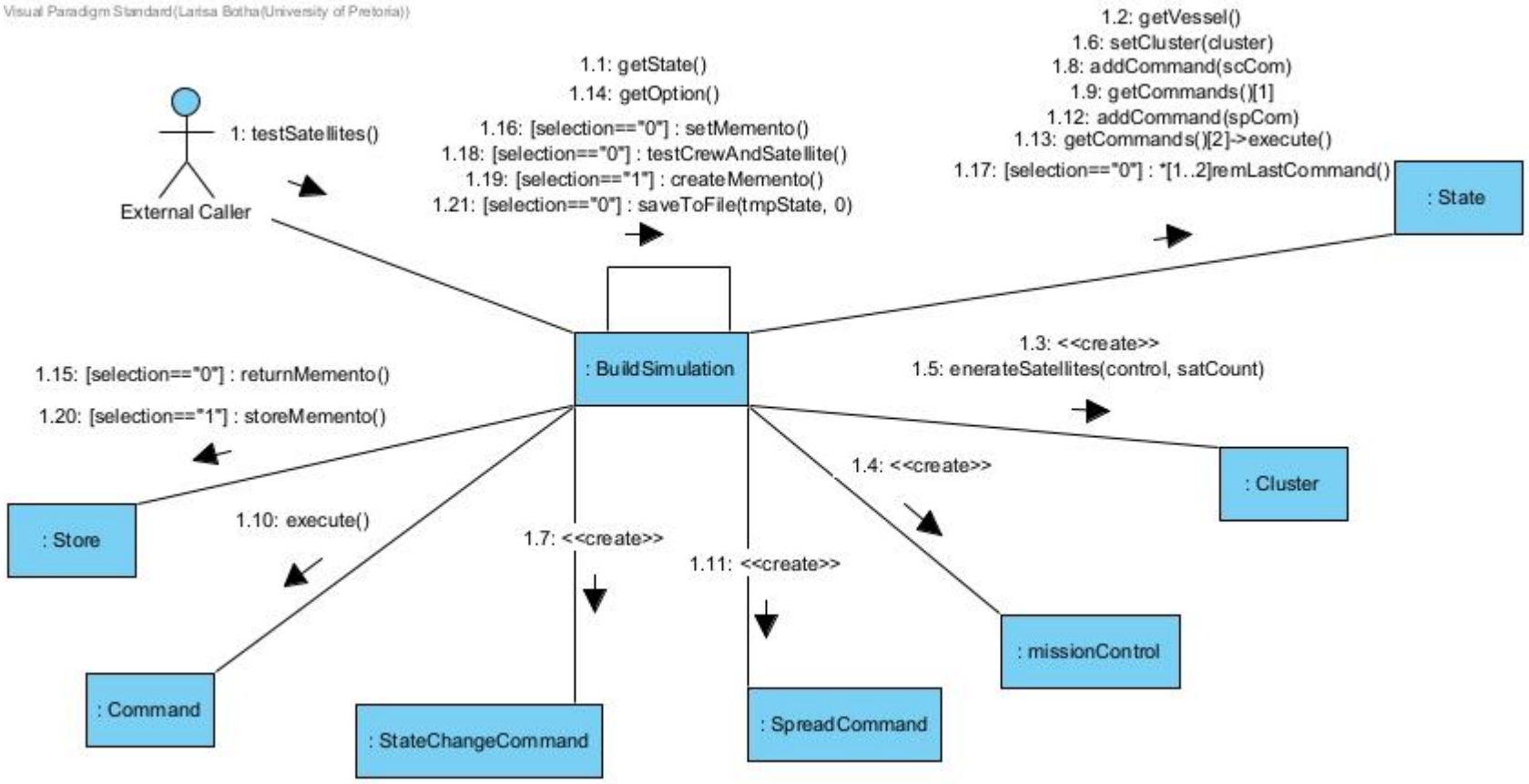
Communication diagram of BuildSimulation's buildCrew method



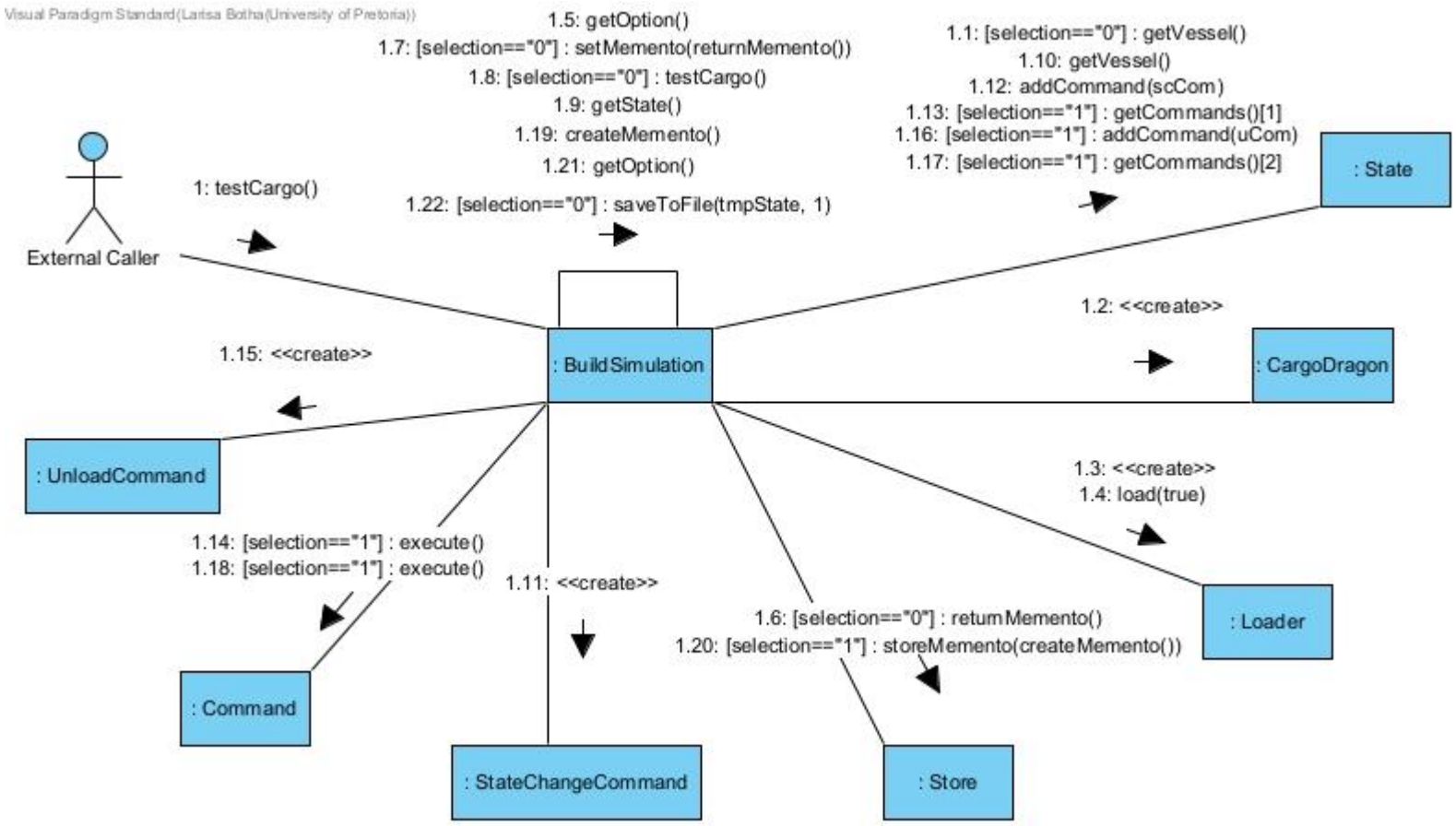
Communication diagram of BuildSimulation's execute method



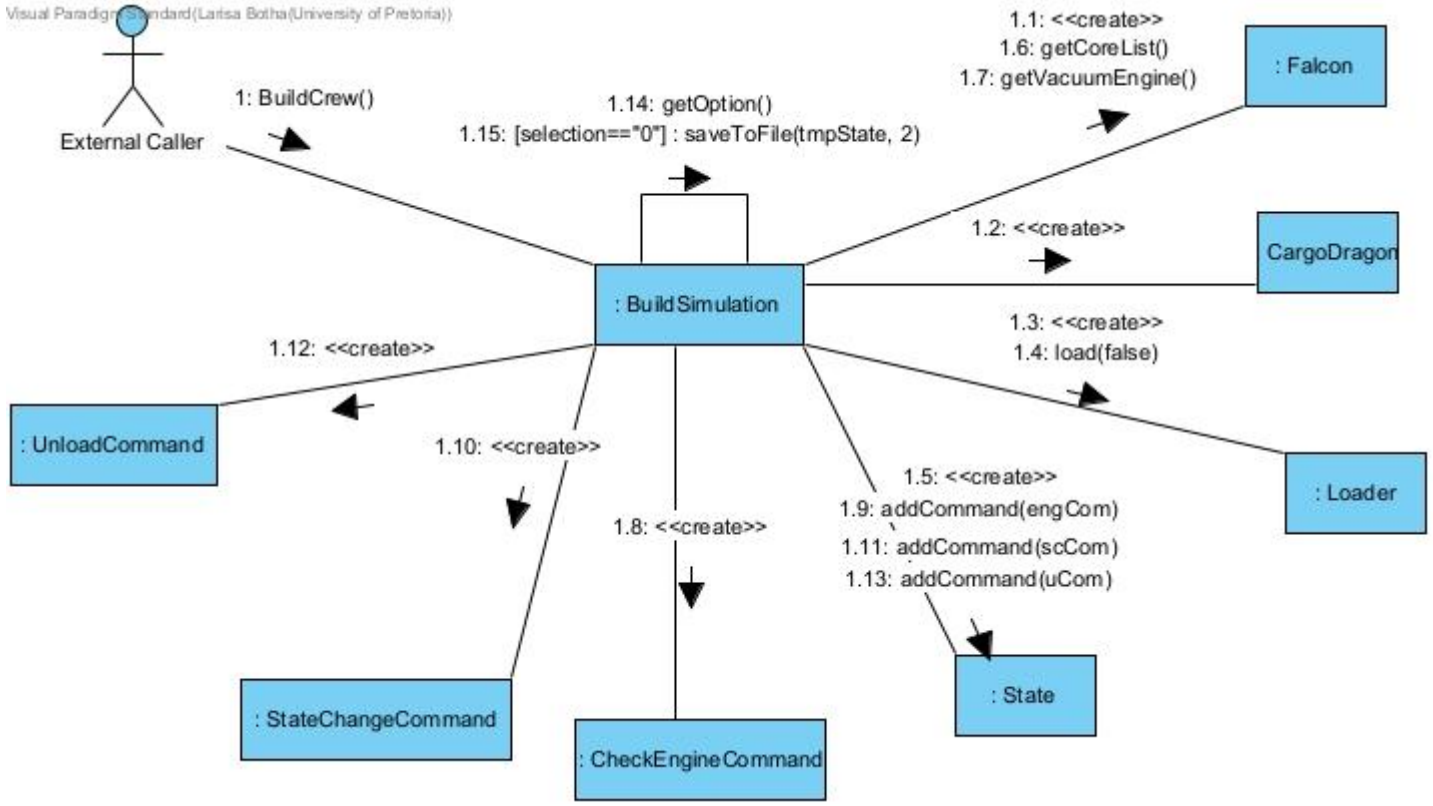
Communication diagram of BuildSimulation's testSatellites method



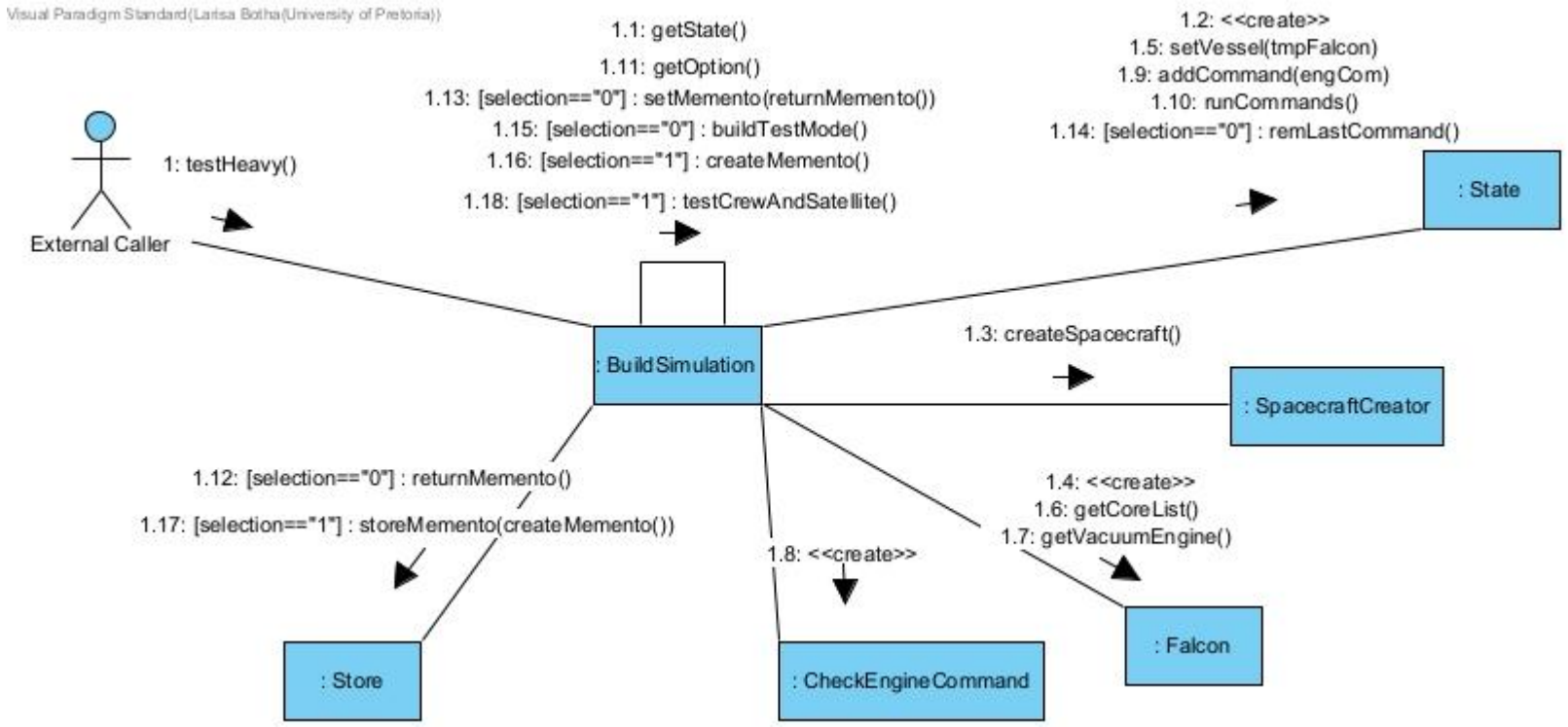
Communication diagram of BuildSimulation's testCargo method



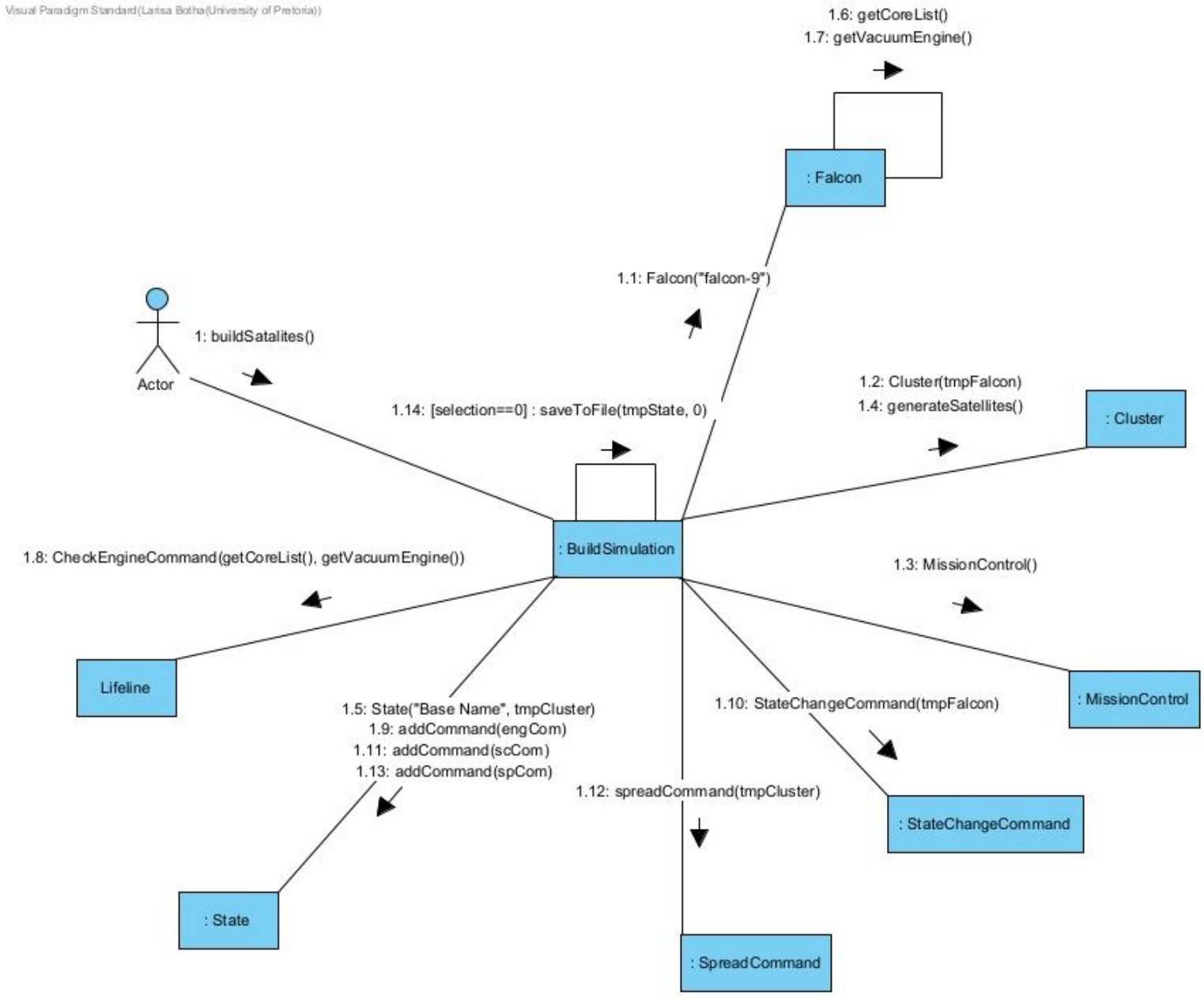
Communication diagram of BuildSimulation's buildCargo method



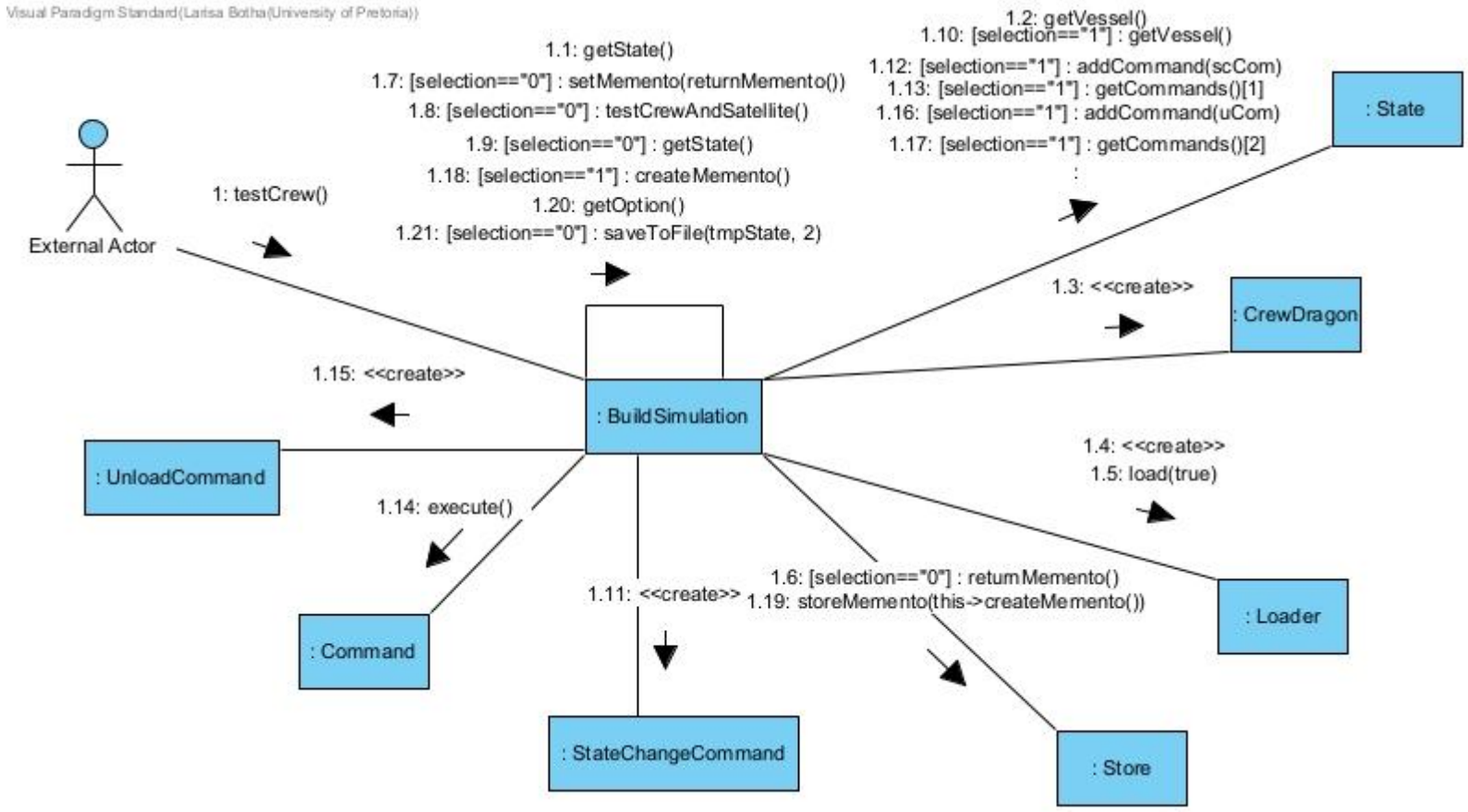
Communication diagram of BuildSimulation's testHeavy method



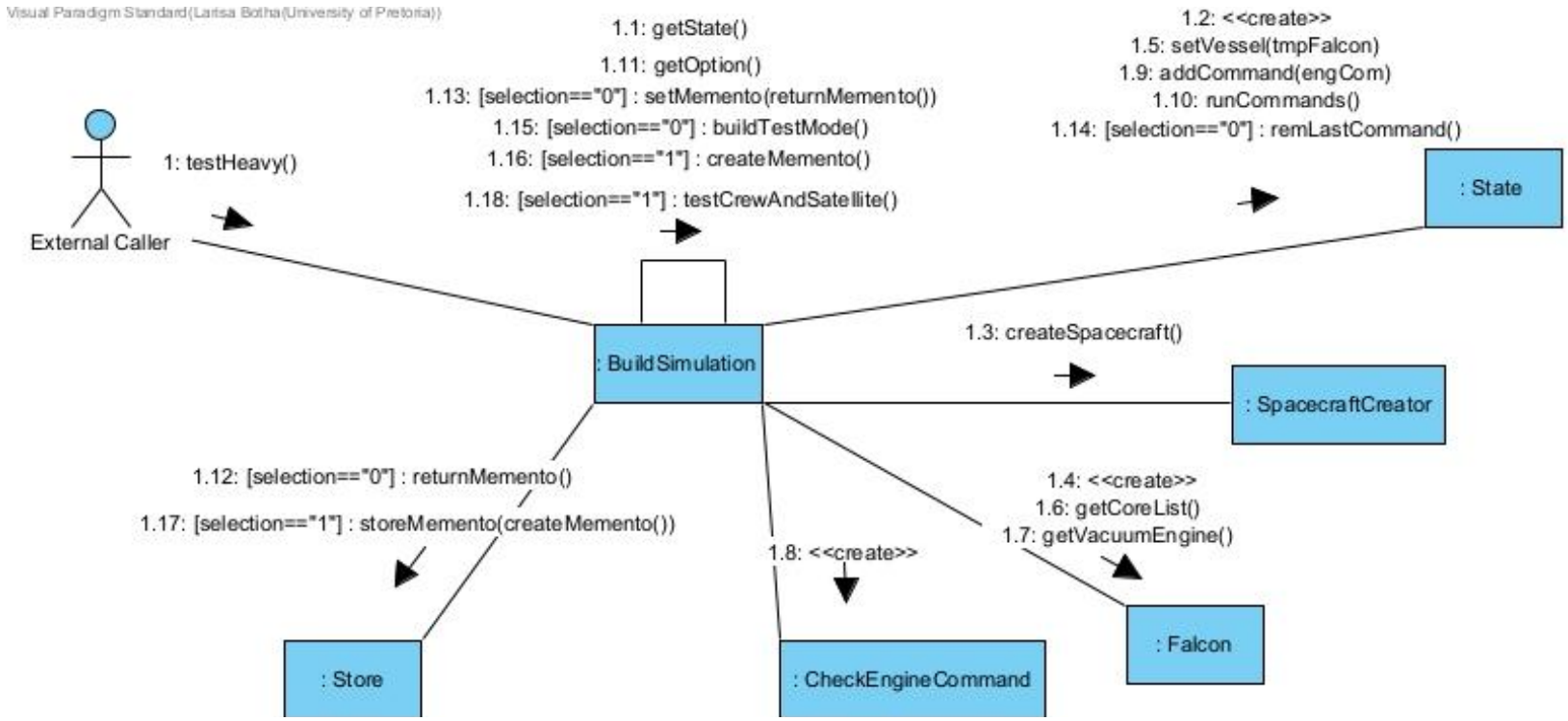
Communication diagram of BuildSimulation's buildSatellites method



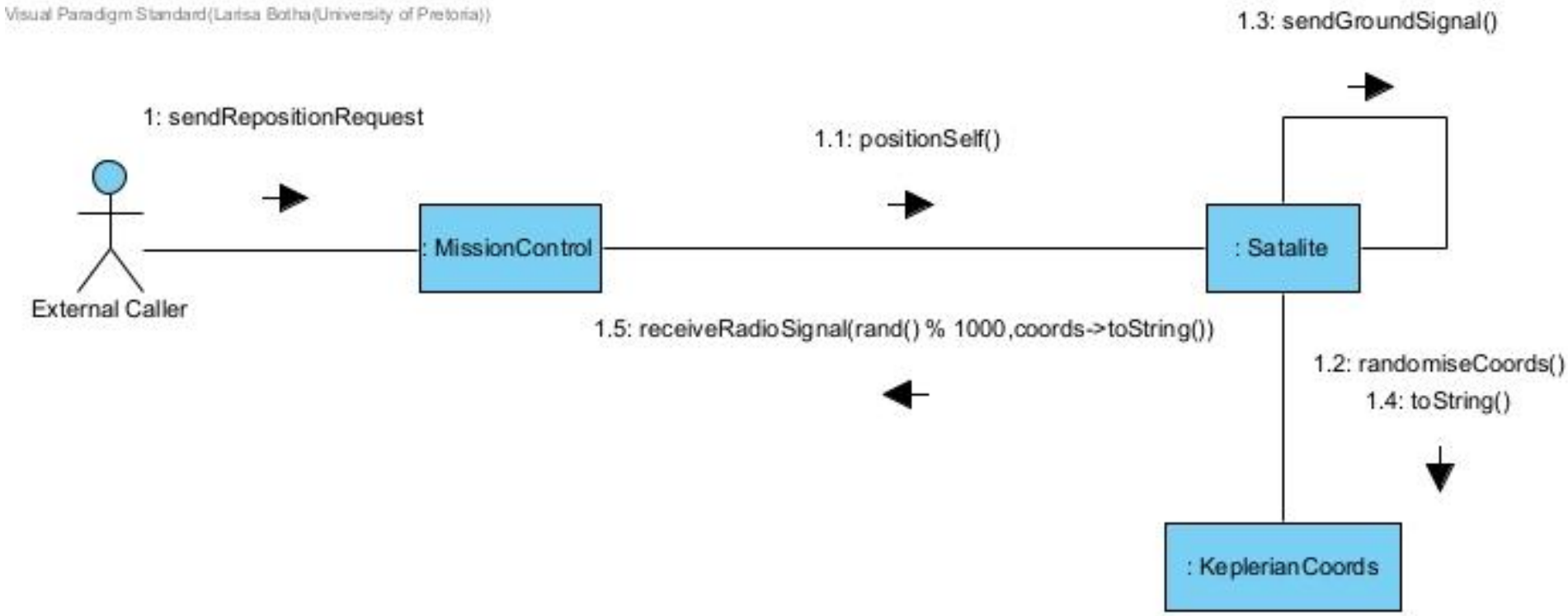
Communication diagram of BuildSimulation's testCrew method



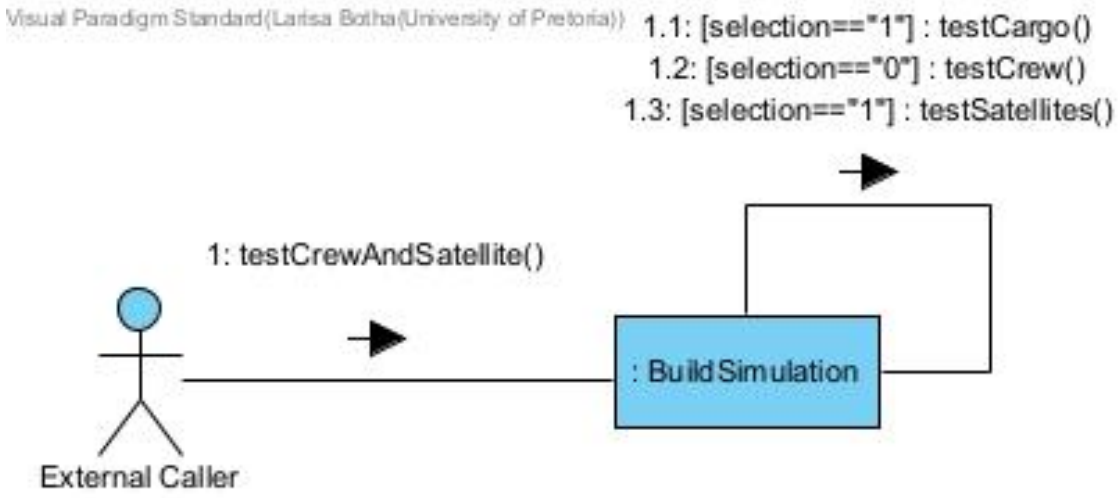
Communication diagram of BuildSimulation's test9 method

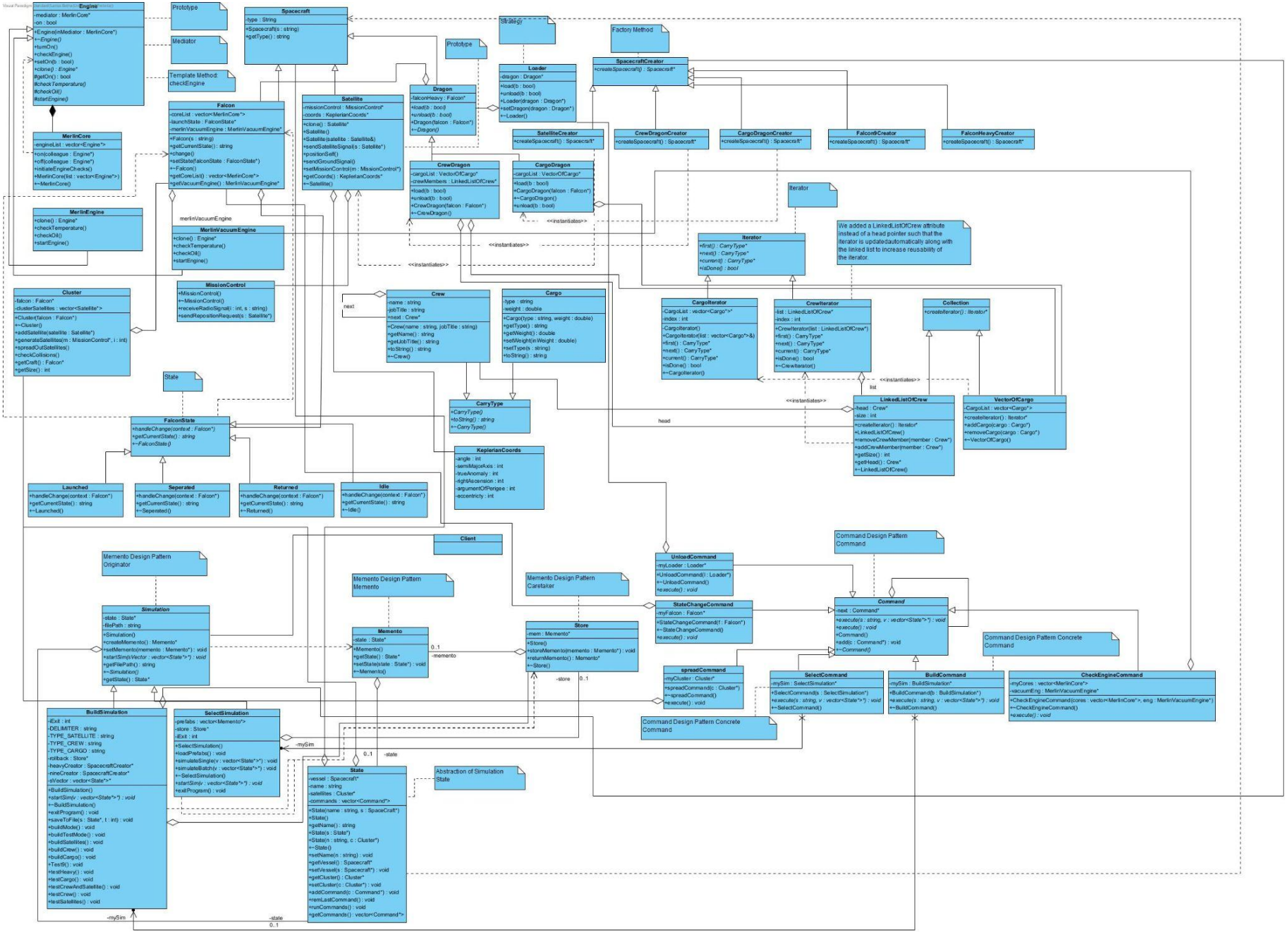


Communication diagram of BuildSimulation's sendRepositionRequest method



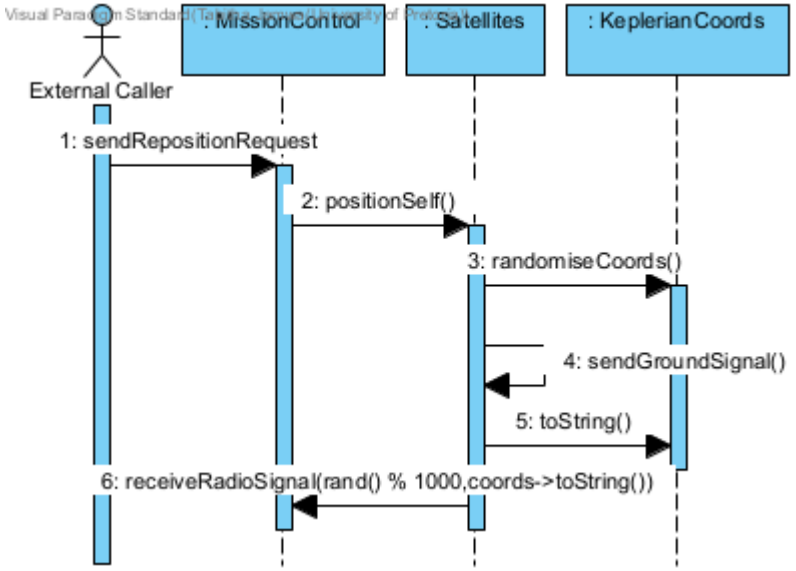
Communication diagram of BuildSimulation's testCrewAndSatellites method



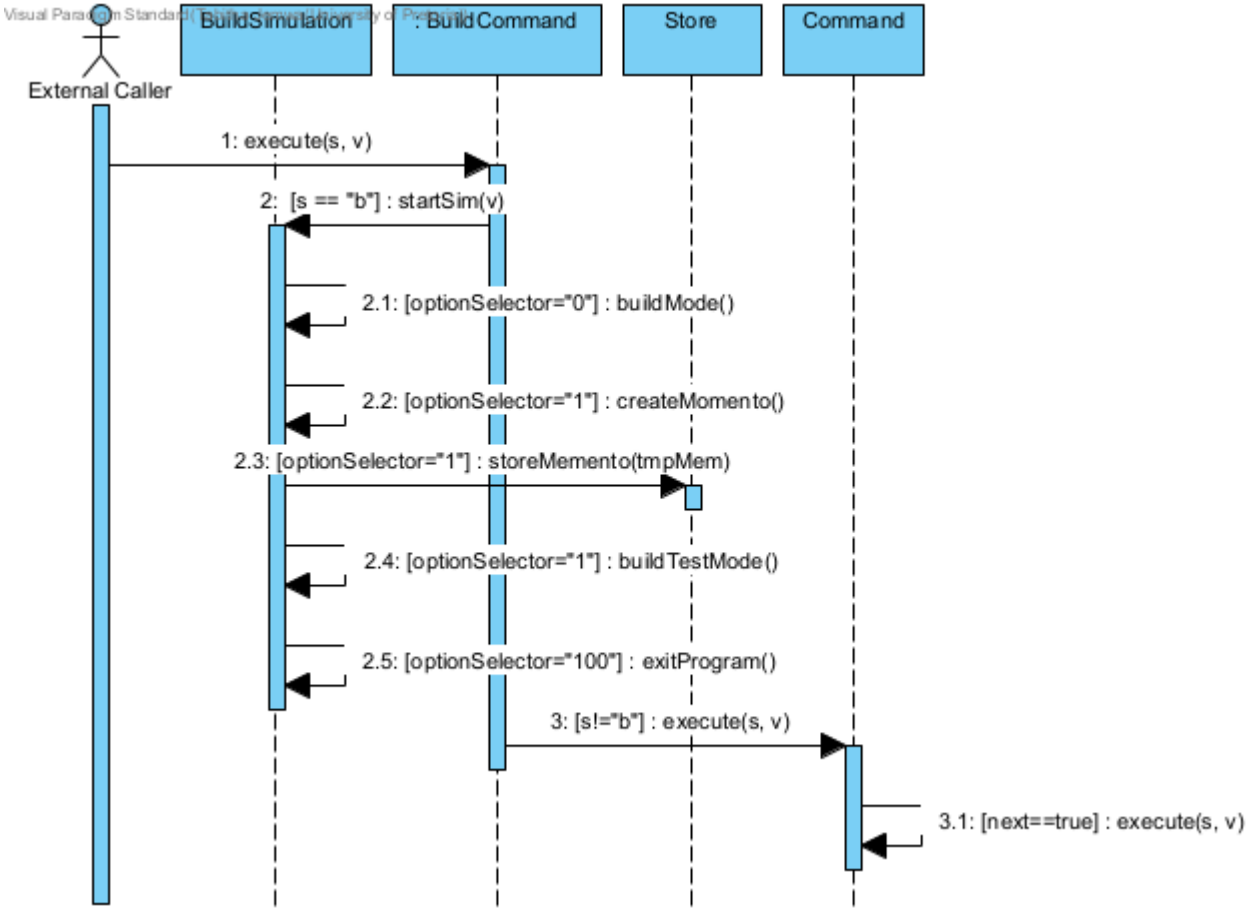

Engine
-mediator : MerlinCore*
-on : bool
+Engine(inMediator : MerlinCore*)
+~Engine()
+turnOn()
+checkEngine()
+setOn(b : bool)
+clone() : Engine*
#getOn() : bool
#checkTemperature()
#checkOil()
#startEngine()
Prototype
Mediator
Template Method: checkEngine
Spacecraft
-type : String
+Spacecraft(s : string)
+getType() : string
Strategy
Factory Method
Prototype
Loader
-dragon : Dragon*
+load(b : bool)
+unload(b : bool)
+Loader(dragon : Dragon*)
+setDragon(dragon : Dragon*)
+~Loader()
SpacecraftCreator
+createSpacecraft() : Spacecraft*
MerlinCore
-engineList : vector<Engine*>
+on(colleague : Engine*)
+off(colleague : Engine*)
+initiateEngineChecks()
+MerlinCore(list : vector<Engine*>)
+~MerlinCore()
Falcon
-coreList : vector<MerlinCore*>
-launchState : FalconState*
-merlinVacuumEngine : MerlinVacuumEngine*
+Falcon(s : string)
+getCurrentState() : string
+change()
+setState(falconState : FalconState*)
+~Falcon()
+getCoreList() : vector<MerlinCore*>
+getVacuumEngine() : MerlinVacuumEngine*
Satellite
-missionControl : MissionControl*
-coords : KeplerianCoords*
+clone() : Satellite*
+Satellite()
+Satellite(satellite : Satellite&)
+sendSatelliteSignal(s : Satellite*)
+positionSelf()
+sendGroundSignal()
+setMissionControl(m : MissionControl*)
+getCoords() : KeplerianCoords*
+~Satellite()
Dragon
-falconHeavy : Falcon*
+load(b : bool)
+unload(b : bool)
+Dragon(falcon : Falcon*)
+~Dragon()
SatelliteCreator
+createSpacecraft() : Spacecraft*
CrewDragonCreator
+createSpacecraft() : Spacecraft*
CargoDragonCreator
+createSpacecraft() : Spacecraft*
Falcon9Creator
+createSpacecraft() : Spacecraft*
FalconHeavyCreator
+createSpacecraft() : Spacecraft*
CrewDragon
-cargoList : VectorOfCargo*
-crewMembers : LinkedListOfCrew*
+load(b : bool)
+unload(b : bool)
+CrewDragon(falcon : Falcon*)
+~CrewDragon()
CargoDragon
-cargoList : VectorOfCargo*
+load(b : bool)
+CargoDragon(falcon : Falcon*)
+~CargoDragon()
+unload(b : bool)
Iterator
MerlinEngine
+clone() : Engine*
+checkTemperature()
+checkOil()
+startEngine()
merlinVacuumEngine
MerlinVacuumEngine
+clone() : Engine*
+checkTemperature()
+checkOil()
+startEngine()
<<instantiates>>
<<instantiates>>
<<instantiates>>
Iterator
+first() : CarryType*
+next() : CarryType*
+current() : CarryType*
+isDone() : bool
We added a LinkedListOfCrew attribute instead of a head pointer such that the iterator is updatedautomatically along with the linked list to increase reusability of the iterator.
Cluster
-falcon : Falcon*
-clusterSatellites : vector<Satellite*>
+Cluster(falcon : Falcon*)
+~Cluster()
+addSatellite(satellite : Satellite*)
+generateSatellites(m : MissionControl*, i : int)
+spreadOutSatellites()
+checkCollisions()
+getCraft() : Falcon*
+getSize() : int
MissionControl
+MissionControl()
+~MissionControl()
+receiveRadioSignal(i : int, s : string)
+sendRepositionRequest(s : Satellite*)
Crew
-name : string
-jobTitle : string
+next : Crew*
+Crew(name : string, jobTitle : string)
+getName() : string
+getJobTitle() : string
+toString() : string
+~Crew()
next
Cargo
-type : string
-weight : double
+Cargo(type : string, weight : double)
+getType() : string
+getWeight() : double
+setWeight(inWeight : double)
+setType(s : string)
+toString() : string
CargoIterator
-CargoList : vector<Cargo*>*
-index : int
-CargoIterator()
+CargoIterator(list : vector<Cargo*>&)
+first() : CarryType*
+next() : CarryType*
+current() : CarryType*
+isDone() : bool
+~CargoIterator()
CrewIterator
-list : LinkedListOfCrew*
-index : int
+CrewIterator(list : LinkedListOfCrew*)
+first() : CarryType*
+next() : CarryType*
+current() : CarryType*
+isDone() : bool
+~CrewIterator()
Collection
+createIterator() : Iterator*
<<instantiates>>
<<instantiates>>
list
head
LinkedListOfCrew
-head : Crew*
-size : int
+createIterator() : Iterator*
+LinkedListOfCrew()
+removeCrewMember(member : Crew*)
+addCrewMember(member : Crew*)
+getSize() : int
+getHead() : Crew*
+~LinkedListOfCrew()
VectorOfCargo
-CargoList : vector<Cargo*>*
+createIterator() : Iterator*
+addCargo(cargo : Cargo*)
+removeCargo(cargo : Cargo*)
+~VectorOfCargo()
State
FalconState
+handleChange(context : Falcon*)
+getCurrentState() : string
+~FalconState()
CarryType
+CarryType()
+toString() : string
+~CarryType()
KeplerianCoords
-angle : int
-semiMajorAxis : int
-trueAnomaly : int
-rightAscension : int
-argumentOfPerigee : int
-eccentricity : int
Launched
+handleChange(context : Falcon*)
+getCurrentState() : string
+~Launched()
Seperated
+handleChange(context : Falcon*)
+getCurrentState() : string
+~Seperated()
Returned
+handleChange(context : Falcon*)
+getCurrentState() : string
+~Returned()
Idle
+handleChange(context : Falcon*)
+getCurrentState() : string
+~Idle()
Client
Memento Design Pattern Originator
Memento Design Pattern Memento
Memento Design Pattern Caretaker
Simulation
-state : State*
-filePath : string
+Simulation()
+createMemento() : Memento*
+setMemento(memento : Memento*) : void
+startSim(sVector : vector<State*>*) : void
+getFilePath() : string
+~Simulation()
+getState() : State*
Memento
-state : State*
+Memento()
+getState() : State*
+setState(state : State*) : void
+~Memento()
0..1
-memento
Store
-mem : Memento*
+Store()
+storeMemento(memento : Memento*) : void
+returnMemento() : Memento*
+~Store()
-store
0..1
UnloadCommand
-myLoader : Loader*
+UnloadCommand(l : Loader*)
+~UnloadCommand()
+execute() : void
StateChangeCommand
-myFalcon : Falcon*
+StateChangeCommand(f : Falcon*)
+~StateChangeCommand()
+execute() : void
spreadCommand
-myCluster : Cluster*
+spreadCommand(c : Cluster*)
+~spreadCommand()
+execute() : void
Command Design Pattern Command
Command
-next : Command*
+execute(s : string, v : vector<State*>*) : void
+execute() : void
+Command()
+add(c : Command*) : void
+~Command()
Command Design Pattern Concrete Command
Command Design Pattern Concrete Command
SelectCommand
-mySim : SelectSimulation*
+SelectCommand(s : SelectSimulation*)
+execute(s : string, v : vector<State*>*) : void
+~SelectCommand()
BuildCommand
-mySim : BuildSimulation*
+BuildCommand(b : BuildSimulation*)
+execute(s : string, v : vector<State*>*) : void
+~BuildCommand()
CheckEngineCommand
-myCores : vector<MerlinCore*>
-vacuumEng : MerlinVacuumEngine*
+CheckEngineCommand(cores : vector<MerlinCore*>, eng : MerlinVacuumEngine*)
+~CheckEngineCommand()
+execute() : void
BuildSimulation
-iExit : int
-DELIMITER : string
-TYPE_SATELLITE : string
-TYPE_CREW : string
-TYPE_CARGO : string
-rollback : Store*
-heavyCreator : SpacecraftCreator*
-nineCreator : SpacecraftCreator*
-isVector : vector<State*>*
+BuildSimulation()
+startSim(v : vector<State*>*) : void
+~BuildSimulation()
+exitProgram() : void
+saveToFile(s : State*, t : int) : void
+buildMode() : void
+buildTestMode() : void
+buildSatellites() : void
+buildCrew() : void
+buildCargo() : void
+Test9() : void
+testHeavy() : void
+testCargo() : void
+testCrewAndSatellite() : void
+testCrew() : void
+testSatellites() : void
SelectSimulation
-prefabs : vector<Memento*>*
-store : Store*
-iExit : int
+SelectSimulation()
+loadPrefabs() : void
+simulateSingle(v : vector<State*>*) : void
+simulateBatch(v : vector<State*>*) : void
+~SelectSimulation()
+startSim(v : vector<State*>*) : void
+exitProgram() : void
-mySim
0..1
-state
State
-vessel : Spacecraft*
-name : string
-satellites : Cluster*
-commands : vector<Command*>*
+State(name : string, s : SpaceCraft*)
+State()
+getName() : string
+State(s : State*)
+State(n : string, c : Cluster*)
+~State()
+setName(n : string) : void
+getVessel() : Spacecraft*
+setVessel(s : Spacecraft*) : void
+getCluster() : Cluster*
+setCluster(c : Cluster*) : void
+addCommand(c : Command*) : void
+remLastCommand() : void
+runCommands() : void
+getCommands() : vector<Command*>*
Abstraction of Simulation State
-mySim
-state
0..1

Sequence Diagrams

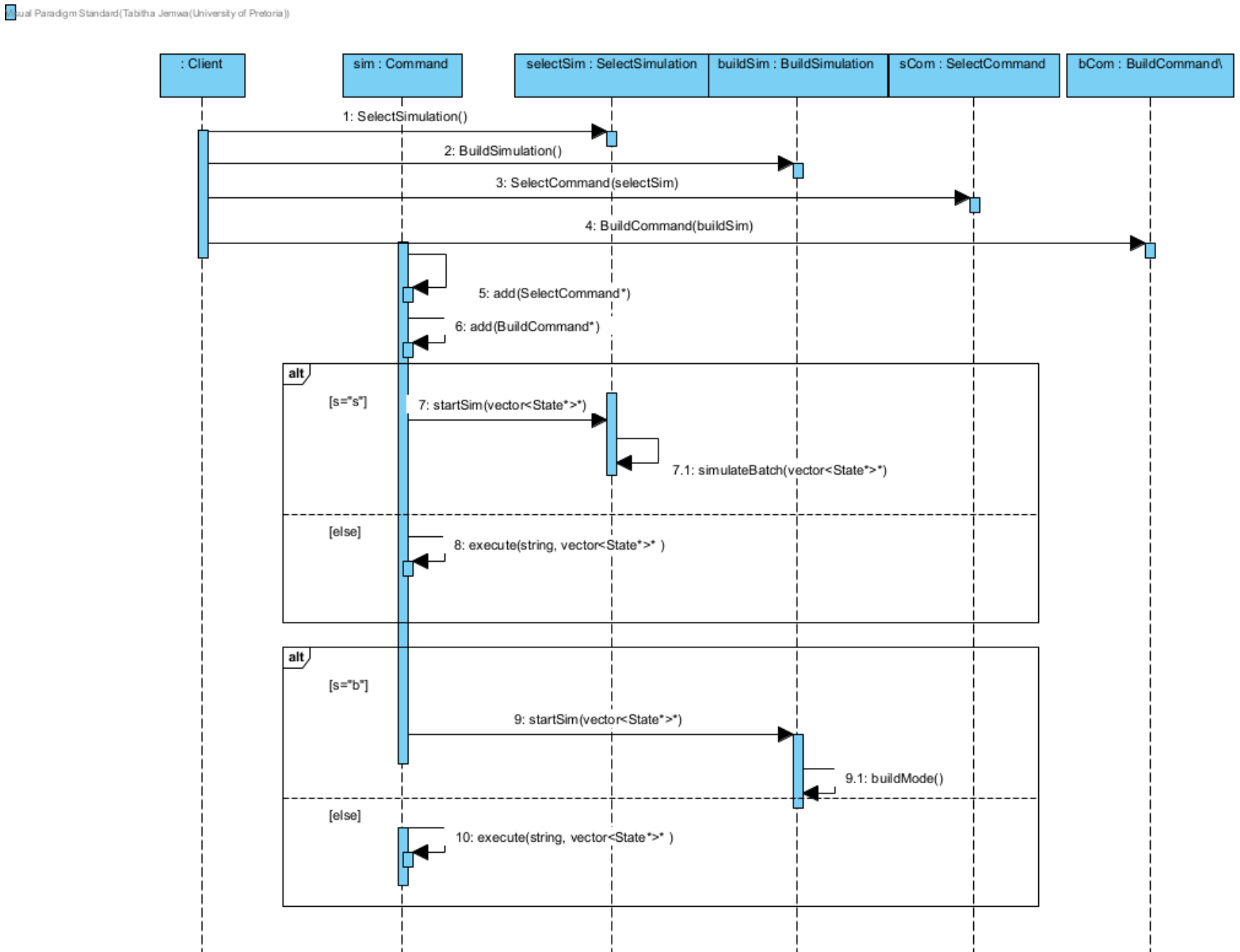
Sequence diagram for satellites



Sequence diagram for the commands

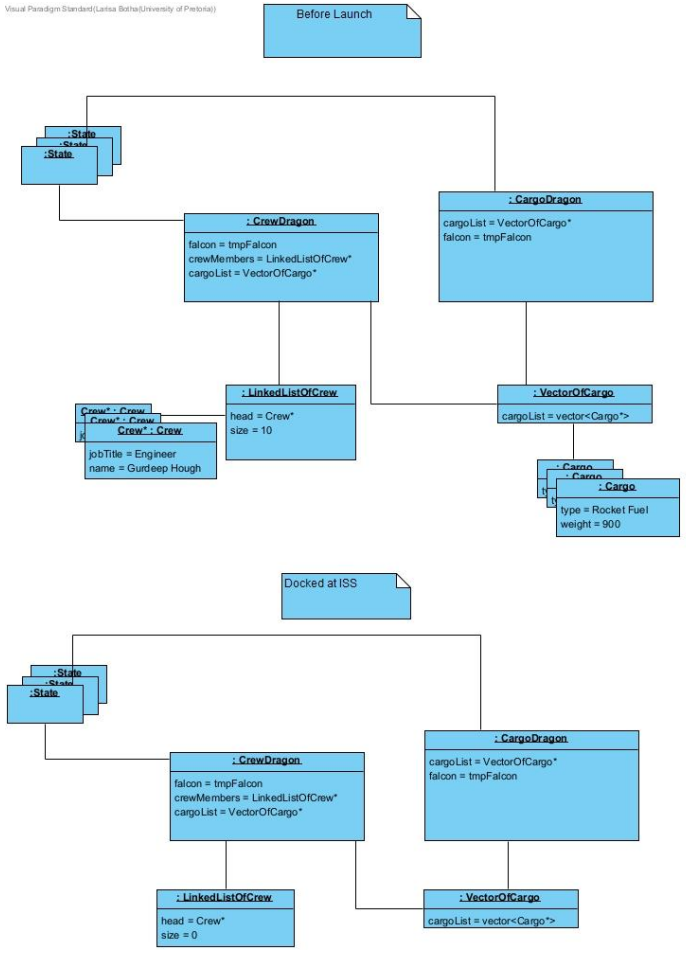


Sequence diagram of main



Object Diagrams

Object diagram describing the dragon docking at ISS



Links

- https://docs.google.com/document/d/18W3HGigsUMSgcnyGHnG72jync0piIzXwSMsuE5YkM88/edit
- https://github.com/sKorpion19091/Semester-Assignment-COS214