Model- DB-ORM

- data:string
- returnData:string
- request:

+ bytesToObject(byte[ ]):object
+ objectToBytes(Object): byte [ ]
+ getMilesPerMonth(int month): HashMap<String, Double>
+ getMilesPerMonthYear(): HashMap<Integer, Double>
+ getFleetSize(int month): HashMap<Integer,Integer>
+ activePlane(String,Int):boolean
+ getFlightRecord(String):Timeserie
+ setFlightData (string,String,byte[ ])
+ setFinishedFlight(FlightData)
+ isFirstFlight(String):boolean
+dateFirstFlight(String):Date

observer

Controller

- m : Model
- v : View
- commandMap:hashMap<String, Command>
-viewMap:hashMap<String,Runnable>
- stop: boolean
- c : commands
- es:ExecutorService

- inniteViewMap()
- innitecommandMap()
+ openServer()
+ frontStreamServer()
+ openAgentsSrver()
+ openAgentsStreamServer()
+ sendStreams()
+update(Observable,Observable)

observer

View

- command : String
- connected :Socket
- waiting: socket
+ printWriterView:PrintWriter

+ runViewServer():
+ getCommand():String
+ sendResponse(String)
+ setCommand(String)

<<Command>>

+ execute(command)

Commands

- sharedState : SharedState

+ setAgentStreams()
+ setFrontstreams()

interperter

-Queue:stack<command>

+ run(Queue<command>):
+ getCommands(code) :Queue<command>

ShareState

- out2agent: PrintWriter
- inFromAgent: BufferedReader
- objectInputStream: objectInputStream
- streamInFromAgent:BufferedReader
- out2front: PrintWriter
- inFromFront: BufferedReader
- objectOutputStream:objectOutputStream
- m : Model
- v : View

+ SharedState(Model, View)

lexer

- tokens:list[string]
- delimiter:string

+ lexer(string):
+ lexer (string,delimiter)
+ hasnext():boolean
+ setData(string)
+ next(): token
+ getallTokens(): list[string]

parser

- lexer:Lexer
- cmdsfactory:commandFactory

+ isDouble(string):boolean
+ getArgsforC(string,stack<command>) :command [ ]
+ parse(string):command
+ symbolResolve(command):command
+ symbolResolve(token):token
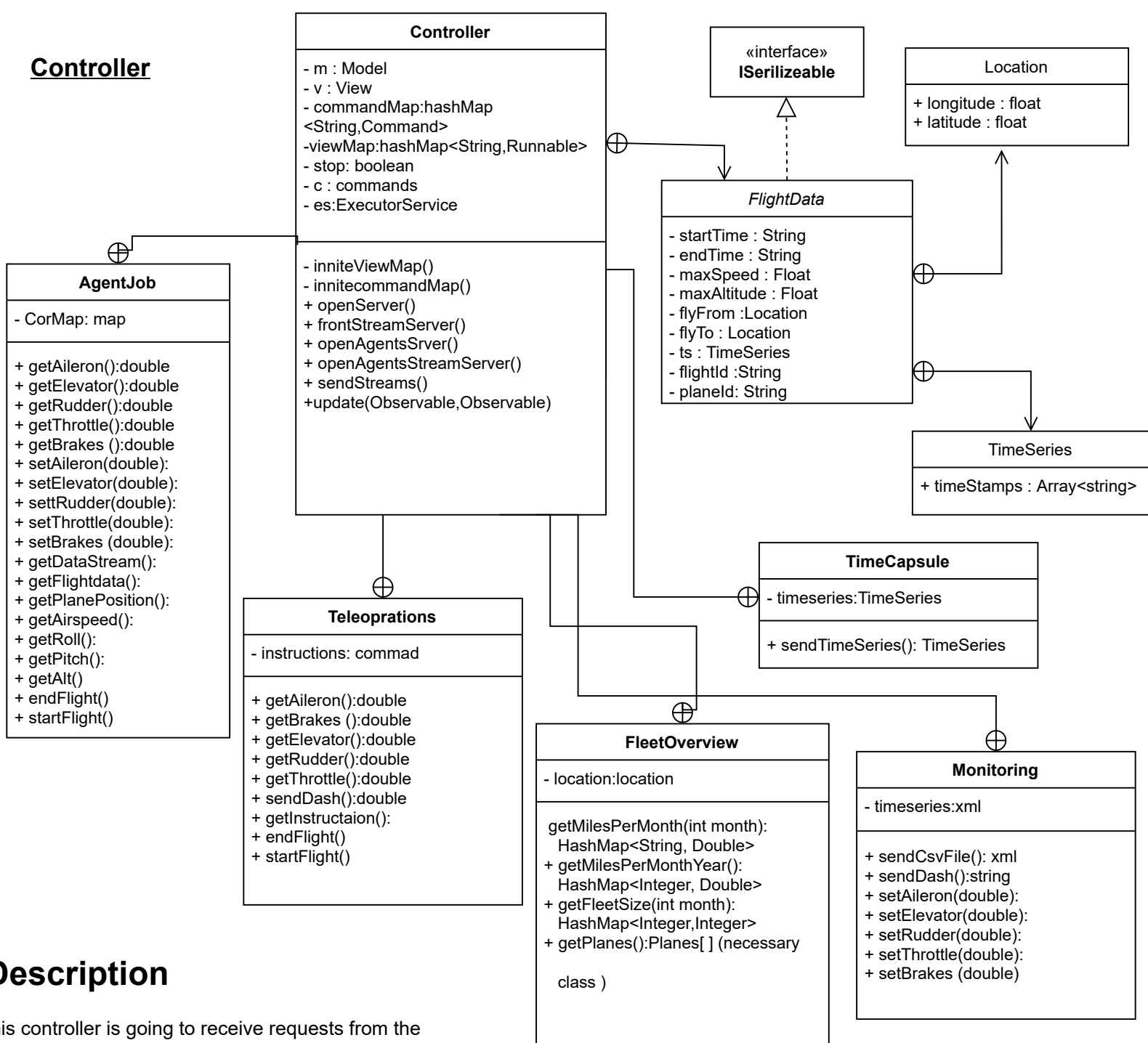+ validateAndPushCommand (stack<command>, command):

Description

in this MVC model we will have : view which can give data over our server and do various management tasks. the controller: that will handle and delegate all outside request and direct or process them to the next component. the model: which contains our logic (interperter , orm and database connection and mangement).
we will also use observers to connect the mvc so each update will happen automatically.
for all of those we will use commands, to speak in one language all over the class

design patterns: command pattern, Observer

Controller

Controller

- m : Model
- v : View
- commandMap:hashMap <String,Command>
- viewMap:hashMap<String,Runnable>
- stop: boolean
- c : commands
- es:ExecutorService

- inniteViewMap()
- innitecommandMap()
+ openServer()
+ frontStreamServer()
+ openAgentsSrver()
+ openAgentsStreamServer()
+ sendStreams()
+update(Observable,Observable)

«interface»
ISerilizeable

Location

+ longitude : float
+ latitude : float

FlightData

- startTime : String
- endTime : String
- maxSpeed : Float
- maxAltitude : Float
- flyFrom :Location
- flyTo : Location
- ts : TimeSeries
- flightId :String
- planeId: String

TimeSeries

+ timeStamps : Array<string>

AgentJob

- CorMap: map

+ getAileron():double
+ getElevator():double
+ getRudder():double
+ getThrottle():double
+ getBrakes ():double
+ setAileron(double):
+ setElevator(double):
+ settRudder(double):
+ setThrottle(double):
+ setBrakes (double):
+ getDataStream():
+ getFlightdata():
+ getPlanePosition():
+ getAirspeed():
+ getRoll():
+ getPitch():
+ getAlt()
+ endFlight()
+ startFlight()

Teleoprations

- instructions: commad

+ getAileron():double
+ getBrakes ():double
+ getElevator():double
+ getRudder():double
+ getThrottle():double
+ sendDash():double
+ getInstructaion():
+ endFlight()
+ startFlight()

TimeCapsule

- timeseries:TimeSeries

+ sendTimeSeries(): TimeSeries

FleetOverview

- location:location

getMilesPerMonth(int month):
  HashMap<String, Double>
+ getMilesPerMonthYear():
  HashMap<Integer, Double>
+ getFleetSize(int month):
  HashMap<Integer,Integer>
+ getPlanes():Planes[ ] (necessary

  class)

Monitoring

- timeseries:xml

+ sendCsvFile(): xml
+ sendDash():string
+ setAileron(double):
+ setElevator(double):
+ setRudder(double):
+ setThrottle(double):
+ setBrakes (double)

Description

this controller is going to receive requests from the Agent and Frontend

each one of them will be assigned to an active object to deal with and those will reside inside a thread Pool for reuse later when they finish dealing with their current request

design patterns: Concurrnet desing pattern (active object) +

ThreadPool.