Model- DB-ORM

- data:string
- returnData:string
- request:

+ askInfo(request):
+ saveToDB(string)
+ isEmpty ()
+ executeAll

observer

Controller

- activeobject:ExecutorService
- threadPool: ExecutorService

+ new()
+ shutdown()
+ reset()
+ print_stream()

observer

View

+ Reset():
+ ListisOfTasks(): list
+ ListOfOpenThreads(): list

interperter

-Queue:stack<command>

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

<<Command>>

+ execute(command)

Connect

- socket: java.net.socket
- input : inputdatastream
- output: outputdatastream

+ socket(java.net.socket): socket
+ readLine (outputstream):
+ writeLine (string):
close():

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
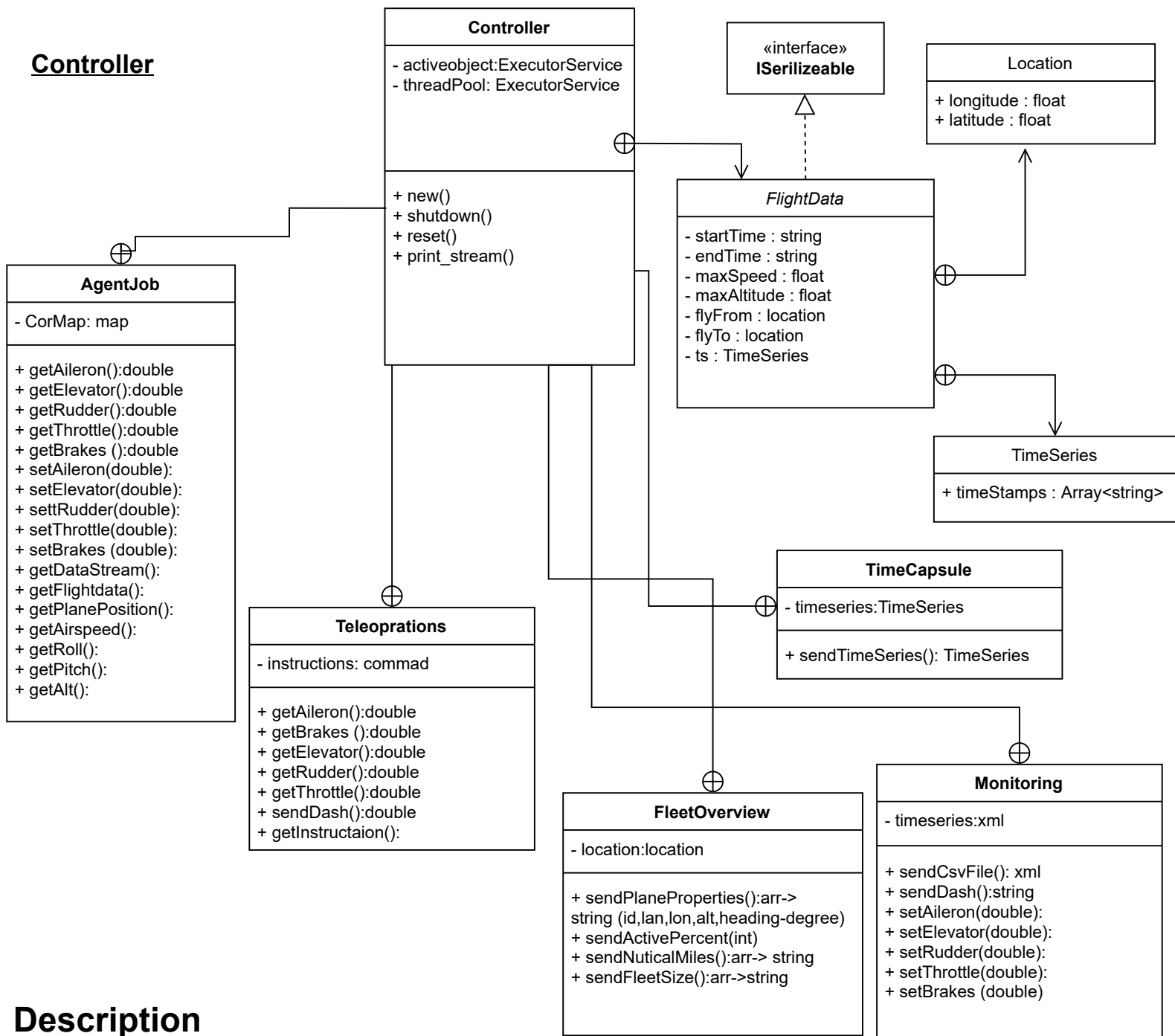+ validateAndPushCommand
(stack<command>, command):

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

- activeobject:ExecutorService
- threadPool: ExecutorService

+ new()
+ shutdown()
+ reset()
+ print_stream()

«interface»
ISerilizeable

Location

+ longitude : float
+ latitude : float

FlightData

- startTime : string
- endTime : string
- maxSpeed : float
- maxAltitude : float
- flyFrom : location
- flyTo : location
- ts : TimeSeries

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():

TimeCapsule

- timeseries:TimeSeries

+ sendTimeSeries(): TimeSeries

Teleoprations

- instructions: commad

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

FleetOverview

- location:location

+ sendPlaneProperties():arr->
string (id,lan,lon,alt,heading-degree)
+ sendActivePercent(int)
+ sendNuticalMiles():arr-> string
+ sendFleetSize():arr->string

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.