Weather Simulator

This document design and flow of weather simulator which generates weather data in following format:

```
Phoenix|33.44838,-112.074043,10|2019-02-14T18:49:20.708128-07:00|Tornado|+1.4035016934765832|1081.0635177491724|82.98373114373605
```

Location is optional
Latitude. Longitude, Sea-level
Timestamp in isoformat
Weather Condition
Temperature
Pressure
Humidity

The simulator can be externally configured using ymal config. It expects config in following format

```
simulation_config:
  name: CityWeatherSimulator
  iterations: 1000
  maximum_task_to_run: 200
  city_config:
    default:
      model_used: random
    Vancouver:
      model_used: naive
    Denver:
      model_used: random
    Nashville:
      model_used: latlong

simulation_report:
  type: filebased
  details:
    path: .
    filename: weather_data.output
    mode: w
    delimiter_type: cba
    keys:
      - cityname
      - latitude
      - longitude
      - sea_level
      - time
      - weather_condition
      - temperature
      - pressure
      - humidity
    keys_separator:
      - "|"
      - ","
      - ","
      - "|"
      - "|"
      - "|"
      - "|"
      - "|"
```

Details:

simulation_config : This part of config is used to decide simulator configuration

name field can be used to choice the type of simulation, current implementation has two choices:

- BasicSimulator
- CityWeatherSimulator

CityWeatherSimulator class extends BasicSimulator class which is a more generic simulator to add tasks and run the simulation. CityWeatherSimulator class simulator customized to output weather data.

Iterations field (integer) gives the number of times simulation runs. Simulator implementation provides interfaces to add tasks (explained later). *iterations* is number of times each task run. Default value is 1

maximum_task_to_run field caps the capacity of number of tasks simulator can run Default value is 10

city_config field if one choice to use BasicSimulator then this field is ignored. This part of config is only valid for CityWeatherSimulator. city_config provides a flexibility to pick and choice weather models for different cities. For example the example shows “Vancouver” city weather data will be generated using “naive” algorithm while “Denver” will be through “random” algorithm “Nashville” through “latlong” algorithm. CityWeatherSimulator supports 3 weather generation algorithm however, the design is such that more complex algorithms can be plugged in and used. “default” parameter is set for generating weather data for cities for which were algorithm is not defined.

simulation_report: This part of config is used to setup the reporting of simulation output. Questions like where should be tasks output be stored in a file or database or kafka medium. Further, it can be configured to pick the output save format. Current we support Pipe and CBA formatter.

type field decides on type of reporter currently the design supports filebased ie FileBasedReporter or string StringBasedReporter again it is easy to plugin new reports

details field is ignored for StringBasedReporter but for FileBasedReporter it gives file level details such as path and filename , mode at which one wants to open the file (overrwrite or append mode).

delimiter_type field is used to decide the formatter of output generated by simulator. Currently the design supports PipeFormatter and CBAFormatter more formatters can be easily added.

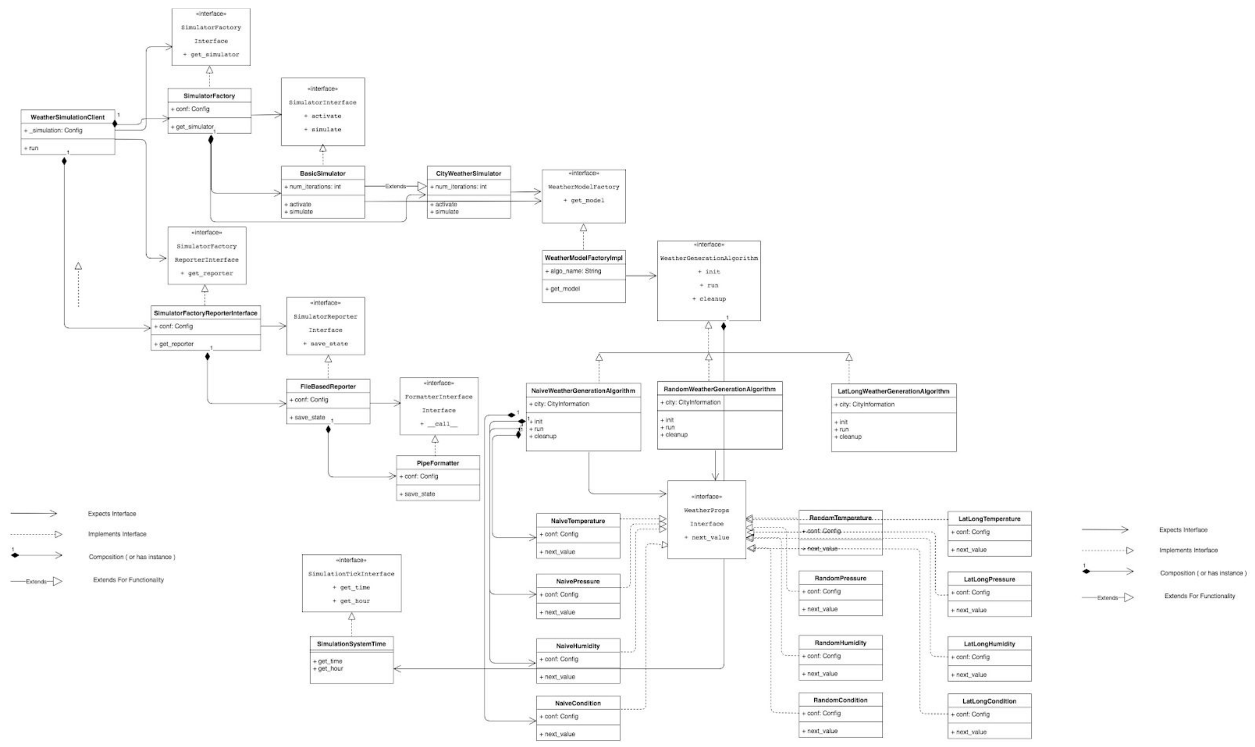
key field one can list the fields that needs to be persisted from simulator output. Because we are required to output the data in particular sequence and format *keys_separator fields* specifies the character to be used between two values of simulator output.
For example CBA expects format in following form

Phoenix|33.44838,-112.074043,10|2019-02-14T18:49:20.708128-07:00|Tornado|+1.4035016934765832|1081.0635177491724|82.98373114373605

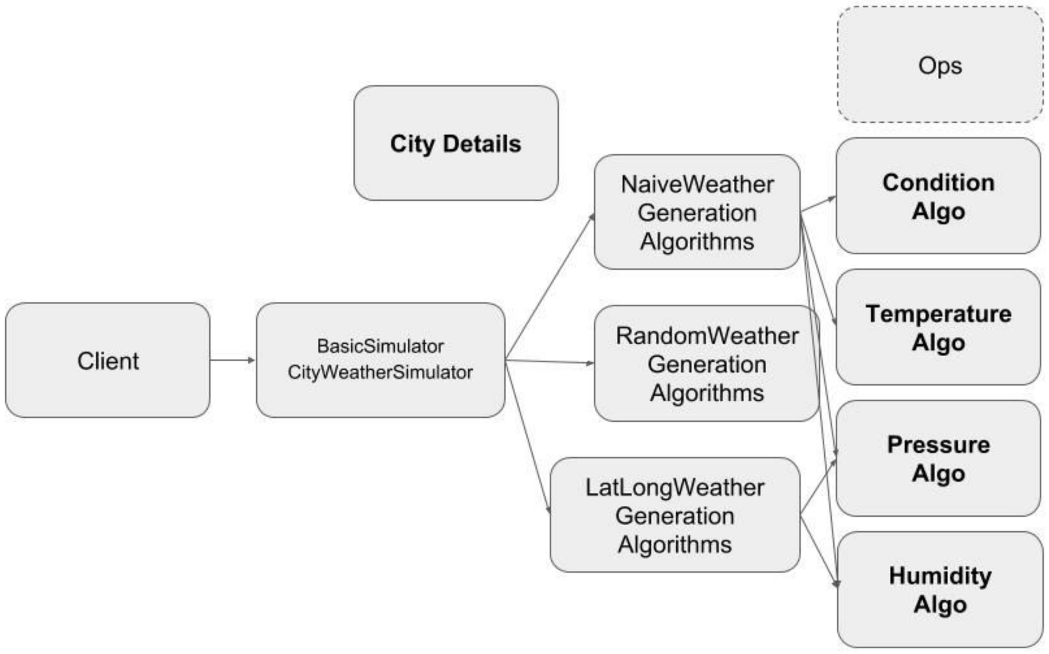
If one sees “Phoenix|33.44838” are separated by “|” while 33.44838,-112.074043 by “,” if future one needs to add any other separator it is flexible to do so.

This document presents a class diagram which gives the class diagram design of Simulator implementation. The principle design philosophy used were

The Single Responsibility Principle
The Open Closed Principle
The Interface Segregation Principle
The Dependency Inversion Principle


«interface» SimulatorFactory Interface + get_simulator
SimulatorFactory + conf: Config + get_simulator
«interface» SimulatorInterface + activate + simulate
WeatherSimulationClient + _simulation: Config + run
BasicSimulator + num_iterations: int + activate + simulate
Extends
CityWeatherSimulator + num_iterations: int + activate + simulate
«interface» WeatherModelFactory + get_model
WeatherModelFactoryImpl + algo_name: String + get_model
«interface» WeatherGenerationAlgorithm + init + run + cleanup
«interface» SimulatorFactory ReporterInterface + get_reporter
SimulatorFactoryReporterInterface + conf: Config + get_reporter
«interface» SimulatorReporter Interface + save_state
FileBasedReporter + conf: Config + save_state
«interface» FormatterInterface Interface + __call__
PipeFormatter + conf: Config + save_state
NaiveWeatherGenerationAlgorithm + city: CityInformation + init + run + cleanup
RandomWeatherGenerationAlgorithm + city: CityInformation + init + run + cleanup
LatLongWeatherGenerationAlgorithm + city: CityInformation + init + run + cleanup
«interface» WeatherProps Interface + next_value
NaiveTemperature + conf: Config + next_value
NaivePressure + conf: Config + next_value
NaiveHumidity + conf: Config + next_value
NaiveCondition + conf: Config + next_value
RandomTemperature + conf: Config + next_value
RandomPressure + conf: Config + next_value
RandomHumidity + conf: Config + next_value
RandomCondition + conf: Config + next_value
LatLongTemperature + conf: Config + next_value
LatLongPressure + conf: Config + next_value
LatLongHumidity + conf: Config + next_value
LatLongCondition + conf: Config + next_value
«interface» SimulationTickInterface + get_time + get_hour
SimulationSystemTime + get_time + get_hour
Expects Interface
Implements Interface
Composition (or has instance)
Extends For Functionality

The overall architecture looks like this


Ops
City Details
Client
BasicSimulator CityWeatherSimulator
NaiveWeather Generation Algorithms
RandomWeather Generation Algorithms
LatLongWeather Generation Algorithms
Condition Algo
Temperature Algo
Pressure Algo
Humidity Algo

Simulator BasicSimulator or CityWeatherSimulator contains tasks or algorithms namely NaiveWeatherGenerationAlgorithm or RandomWeatherGenerationAlgorithm and LatLongWeatherGenerationAlgorithm.

Each task then runs Condition, Temperature, Pressure and Humidity algorithms. Like NaiveWeatherGenerationAlgorithm contains NaiveCondition, NaiveTemperature, NaivePressure, NaiveHumidity algorithms to run.

The algorithms were implemented with “yield” python functionality which preserves the state of object on every iteration. The idea is that since weather condition, temperature, pressure and humidity depend on subsequent days such functionality can easily be captured.