A framework for mobile network data micro-simulation

Overview

Gathering mobile network data from telephone operators, has been proved to be a difficult task even for research purposes (let alone for production!). Moreover, checking the *real* performance of a model is simply not possible, because there is no way we can know for real the position of the people. Of course, some basic quality checks can be made, but if we must choose between two models, chances are that those tests give us no clue about which one performs better.

Thus, we propose the development of a framework to run mobile network data micro-simulations. These simulations will provide us with synthetic data, useful to test the models. Being a micro-simulation, the estimations can be compared to the real data, unavailable in real life. While a simulation is (no matter how sophisticated) always different from real data, there is really no reason to expect that a model would perform worse for synthetic data than for real data. On the contrary, dealing with real data would be expected to be even more problematic, so a good performance for simulated data should be demanded anyway.

In this document we contribute to the framework with some design ideas. They are intended to be more guidelines than design requirements, because some better ideas might arise. We think that object-oriented programming paradigm is the natural choice for micro-simulations, and therefore an object-oriented language should be used, though any of them would fit.

Some preliminary considerations

As usual, we will start by defining the most general abstract classes/interfaces and continue with more refined classes that will inherit from them. The framework alone should be enough to run simulations, so at least a complete set of basic concrete classes should be implemented too. Moreover, the more realistic we want the model, the richer variety of classes we need. Thus, factory methods might come handy for object construction. On the other hand, the users might want to derive and use their own classes, so an abuse of final classes should be avoided.

The framework should be thread safe or directly multi-threaded, because simulations are expected to be slow. Of course one option would be a cluster (Spark, Hadoop, . . .) based implementation, but we do not find it indispensable.

A very rough approach to the simulator would be: the agents (people, for instance) perform its actions (move, call, . . .), the clock ticks, and repeat till time is up. During the simulation the data needed for estimation will be written to disk. This involves an exact count of the population for each cell, a Telephone Operator account, and an administrative account. The implementation of different methodologies to get the account by the Telephone Operator is interesting to test their effect in the final estimations. Several methodologies might similarly be implemented for the administrative accounts.

The class World

The simulation takes place in an object of class World. The main program will just consist in the creation of an object of class world. Some fields suggested are:

- agentsCollection: an object of class AgentsCollection that contains the array/list of agents. Agents include people, mobile phones, telephone operators, . . .
- map: a map object of class Map for the people to move.
- clock: a clock object of class Clock to keep track of the time.

We will start with a superclass World that contains little more than a clock. Next we may define a subclass GeoWorld that includes the map and some agents that just move around. And so on. Obviously new methods are included too. The class World fields are always the most abstract classes, so we don't need to modify the code of the class. Agents might be implemented as array/list directly but it is useful to encapsulate them in a class for flexibility (we might need pass forwarding though), which is our suggestion.

The methods of class World would be:

- The contructor: Basically initializes the objects and calls runSimulation
- runSimulation: this method is called at the end of the constructor. It will start the simulation. Mainly it allows agents to perform actions, dumps the state of the simulation to file and updates the clock.
- doStaff: this method is virtual but mostly a pass forward to allow the agents to perform actions.
- dumpState: again a virtual method and mostly a pass forward to write to disk the data generated.
- getCurrentTime: since clock should be protected from other objects this is a pass forward for the agents get the time.

The clock is updated when all people finish their actions. A more realistic simulation would let run both clock and people in different threads. This approach may add more complexity to the framework than the profits we get, but may be not we have to take a decission. More methods may be added in the subclasses, for example a mapFill method to populate the map with people in GeoWorld subclass.

The class AgentsCollection

This class is basically a collection of the agents of the World. Agents can, in principle, perform actions but they might be programmed to behave passively (public emergency services' switchboard for example). Some fields:

- world: The object of class World where the collection lives in.
- agents: A list-like structure of agents. In order to improve performance parallel and parallel distributed collections are interesting options to be considered.

Some examples of agents are:

- People: They wander around the map, call, send SMS messages, . . .
- Mobile phone devices: They connect to the antennas (and thus to mobile phone networks), allow people to call, . . .
- Antennas: They connect to the mobile phone devices, allow them to make calls, send data to telephone operators, . . .
- Telephone operators: They collect data from antennas and build the CDR, or even the estimation of the number of devices by cell (the method can be overriden to define subclasses with different methodologies). They also manage the calls/SMS's between different networks.

The methods of this class are mainly methods to manage the list (add, delete, count, . . .) and a method to allow agents to perform actions.

The class Map

This class represents the map of the World. Its main role is to control the movement of the agents. This control might include speed limit (physical issues, traffic issues), unreachable points, . . . Thus it must contain information about streets, roads, buildings, and more.

The most important method of this class would be getClosestAllowed. When an agent wants to go somewhere, it specifies its target and the map returns where the agent moves to. The target would be for example agent's home, so as time passes by the agent gets closer and closer and eventually arrives.

The Agent subclasses

As has already been stated, agents are objects that perform actions. They are expected to be by far the abstract class with more subclasses, and more different between them. We are going to describe some possible subclasses of this family:

- Agent: This is the most abstract class and all agent classes have this one as ancestor. An abstract class/interface/trait with a method that specifies how to perform actions is enough.
- Locatable: It is still an abstract class, it adds a method that returns the location of the agent. Even in the few examples there are non locatable agents: Telephone operators.
- Movable: Locatables that can move, mostly people. A variable field position is included.
- Immovable: Locatables that can not move, like antennas. A constant field position is included.
- Holdable: Locatables that move with another agent. The most obvious is the mobile device, moving with its holder.

The example

An example with scala code and its scaladoc is included too. Note that we are not suggesting to use scala for the framework, the code and documentation are included as an illustrative example of the suggestions of this document. It is also no optimized so for older computers you are warned to decrease the number of people. Moreover no calls are generated so it is just people randomly moving in the plane. Two cells are defined, $x < 0$ and $x \geq 0$, so a similar amount of people in both cells is expected. The cell is a function that maps each point into a String, and is a field of the map object. In practice this function should be a field of the agents that represent telephone operators, as it is an important part of its methodology.