ESSnet Big Data II

Grant Agreement Number: 847375-2018-NL-BIGDATA

https://webgate.ec.europa.eu/fpfis/mwikis/essnetbigdata
https://ec.europa.eu/eurostat/cros/content/essnetbigdata_en

Workpackage I
Mobile Network Data

Deliverable I.2 (Data simulator)
A simulator for network event data

Final version, December 12, 2019

Prepared by:

Bogdan Oancea (INS, Romania)

Marian Necula (INS, Romania)

David Salgado (INE Spain)
Luis Sanguiao (INE, Spain)

Workpackage Leader:

David Salgado (INE, Spain)
david.salgado.fernandez@ine.es
telephone : +34 91 5813151
mobile phone : N/A

Contents

1

Introduction

1.1. A short introduction to discrete-event system simulation

1.1.1. Discrete-event simulation benefits

In December 1968, took place the second edition of the Conference on Application of Simulation using the General Purpose Simulation System(GPSS), a discrete-event simulation programming language and software implementation developed by IBM. Besides presentations regarding the subject matter, a special section of the conference was dedicated on the topic of "Difficulties in convincing the Top Management". A critical aspect in decision making process, either for scientific or business related purposes, is to provide the necessary and sufficient arguments in favor or against pursuing a certain outcome, given a set of very specific constraints. Even in the early stages in the development of event system simulation, the researchers recognized the huge potential of system simulations to provide strong arguments upon which decisions can rely. Therefore, some of the major benefits of using system simulation, as presented in (Banks et al., 2010) and reformulated in terms of the ESSnet WPI package objectives, are explicitly stated below:

- system simulation can be used to gain a more or less general level understanding on the statistical nature of the process which generates new data sources, which many, if not all, are outside of the scope of traditional statistics;
- system simulation can be used to experiment with new methodological designs before implementation;
- system simulation can be used to generate data, otherwise very hard to access or inaccessible;
- system simulation can be, in many cases, cheaper to use than piloting implementation projects;
- system simulation can be used to inform top management on the advantages and disadvantages on using different approaches in integrating new data sources into statistical production;
- system simulation can provide expressive and very informative animations on the data generation process, otherwise difficult to extract from tens or hundreds of thousands of documentation pages;
- system simulation can be used in training new experts.

Since its inception system simulation was used successfully in many scientific research areas such as physics, chemistry, biology, economics or in different type of economic activities - aerospace, transportation and infrastructure, construction or financial industries, to name only a few.

1.1.2. Brief history of simulation software and a description of discrete-event system simulation

The spark of event system simulation was World War II, in 1946, trying to perfect the design nuclear fission devices, John Von Neumann and Stanislaw Ullam used an electronic computer to simulate neutron diffusion and multiplication rates in fission reactions, using a novel statistical approach called Monte Carlo sampling method. By 1960, system simulation attracted the attention of IBM, which developed and released as a programming language and commercial application a general purpose system simulator. One of the first applications was developed by the United States Federal Aviation Administration used to deliver reliable weather forecasts. Between 1960 and 1980 a plethora of computer programming languages dedicated to discrete-event system simulation were developed. SIMSCRIPT and Simula, the first truly object-orientated language, overarch that period by the sheer number of users and applications. Also in this period, discrete-event system simulation became accessible to a larger audience with the advent of spreadsheet applications. From 1980 to 2010 the efforts were concentrated in providing integrated development environments for system simulation, providing mainly tools to reduce the burden of programming simulation applications and visualize the results. By today, simulation software should support functionality such as:

- modularity - new components should be easily added, replaced or discarded to better suit the needs of users;
- scalability - the software should event system simulation at small, medium or large scale. This feature is proportional with the available computing resources.
- verification and testing functionality - the software should let the users create multiple scenarios and answer 'what-if' questions;
- visualization - the software should have a graphics component for visualizing the results;

Discrete-event system simulation evolved along with simulation programming languages and software. The shift from endless days of programming and debugging brought the necessary freedom to focus on robust simulation techniques. Discrete-event system simulation focuses on problems where the state of a system changes in a discrete sequence of events, changes activated by the state of the system or sub-systems, the time advance function and the state trajectory function. The changes are also called the state transition function. Modelling a queue process is a classical example in which discrete-event system simulation is appropriate, where we are interested to count the number of occurrences on a set of predefined events and track how the system or sub-systems evolve from one state to another. In order to model a system, we need to define the atomic components which can be, in a very broad description, reduced to:

- entities or agents - the real world counter-part of physical object, animate or inanimate;
- attributes - a set of intrinsic properties pertaining to entities;
- activities - a set of predefined activities performed by entities;
- events - endogenous and/or exogenous triggers for activities;
- states - a set of variables needed to describe the system or sub-systems at any given time.

Modelling a system simulation involve a series of steps that should be carried out(Nutaro, 2011):

- correctly state the problem, usually based on the least harmful assumptions;
- set the objective or what results to expect and/or not to expect from the simulation;
- collect information about the system to be modelled;
- conceptualize or extract the essential features of the real-world system to be modelled;
- identify the necessary software tools to implement the simulation;
- verify and validate if the simulation model is logically correct and accurate;
- design multiple run scenarios;
- provide some measures of performances, based on the initial sets of objectives;
- document and report.

The next sections and chapters will present in great detail these steps for our case.

1.2. Mobile network discrete-event system simulation

Mobile network data represent a new and very promising data source for producing official statistics in different domains such as population statistics, tourism, transportation to name only few of them, and several pilot studies have been already conducted in this areas.

There are already some interesting results obtained by the research community around Mobile Network Operators and some leading National Statistical Institutes have clearly shown the immense potential to produce official statistics using mobile phone data. Besides the classical statistical indicators, mobile network data could be used to tackle novel statistical products of social interest with the incorporation of network science methods (**?**).

But, despite the regulatory support in National and European Statistical Acts, data access is proving to be a major challenge for most of European NSIs. Accessing mobile network data from telephone operators is a very difficult task even for research purposes. At the time of writing this report, as far as we know, no NSI from EU has a long term and sustainable access to mobile network data. Nevertheless, some NSIs have limited access to mobile network data for some specific projects with interesting results (UN Global Working Group on Big Data for Official Statistics, 2017), but it seems that no one has been able to incorporate this new data source in the current statistical production.

The main goal of the Mobile Network Data Workpackage is to develop a general framework for the production of official statistics using data generated by mobile networks. In order to avoid potential blockings due to the current obstacles in accessing real data, one of the research tracks proposed within this workpackage is to develop a tool that will enable us to generate instrumental data. Thus, we propose the development of a framework to run mobile network data micro-simulations. These micro-simulations will provide us with synthetic data, that will allow us not only to immediately proceed with the development of the general framework but also to test the models included in this framework. Checking the real performance of a model is simply not possible in real life, because there is no way we can know the real positions of the people at different time instants. 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.

While a simulation is 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.

First, we started by defining the main features of the synthetic data generator:

- it should support loading different maps (geographical areas) as a basis for the simulation; these maps could be real maps (cities, districts) or could be synthetically generated with an appropiate software;
- it should have support to define a reference grid overlapped on the map as a basis for computing location probabilities and population densities;
- it should support multiple mobile network operators to be involved in a simulation;
- it should support defining the configuration of the mobile network(s): antennas locations, antennas parameters;
- it should support a flexible modeling of the interaction between mobile devices and antennas;
- it should support a flexible modeling of how people move around the map;
- it should support defining some basic characteristics of the population involved in a simulation;
- it should produce output files with enough information to be able to compute at least:
  1. an estimate of the location probability of each mobile device;
  2. the true location of each mobile device during a simulation;
  3. the movement pattern of the population;

4. the true location of each individual involved in a simulation, either carrying a mobile devices or not;
5. the number of devices in a well delimited area;
6. the true population in a well delimited area;

- it should be fast enough to run simulations with large populations ($10^4 \dots 10^6$ individuals);

Before starting to develop an entirely new tool for synthetic data genation we've made an inventory of the existing tools in this area, checking if there is one that can be used for our purposes.

The *cdr-gen* project (Bordin, M.V., 2017) is a very simple Call Detail Record (CDR) generator written in Java that allows the user to configure up to a certain extent the parameters of the calls (duration distribution, type of call, etc.) but has no support for defining the geografical coordinates of the mobile devices, the movement of the people carrying mobile devices or the parameters of the network.

Another CDR generator (Real Impact Analysis, 2017) written in Scala allows users to generate CDRs data with different models or with a mix of models. A simulation implies several steps: generate the cells, the mobile operator, the users, the social network of users and eventually generate the interaction between users. However, the capabilities to run complex simulations lacks, the cells of the mobile network are generated randomly with a fixed shape. There is no support to define our own maps and the (at least some) technical parameters of the antennas.

NetSim (Tetcos, 2019) is a software that enables users to simulate a network comprising of devices and links, and study the behavior of this network. While this is a complex software that includes a user-friendly GUI and capabilities to simulate several real mobile network communications protocols, it is a commercial product with a limited version for academic institutions. An important drawback of this simulation software comes from the fact that it is oriented on producing data needed for mobile networks optimizations and it does not provide the kind of data that matches our statistical needs.

Another network simulation software that we've tested is OPNET Network Simulator (Zheng and Hongji, 2012). Besides being a commercial software, it is also oriented on producing data needed for mobile network optimization and it does no output the information that we need to produce population estimates.

The traffic simulation packages SUMO (Krajzewicz et al., 2012) and MATSim (Horni et al., 2016) are more similar to our needs of modeling the population mobility but unfortunately they don't have any support for mobile devices and networks.

Considering the minimal set of features that we defined for our synthetic data simulator we couldn't find any pre-existing software to entirely fullfil our needs, and we proceeded to develop our own simulation software.

Since the first attempts in early 70's and 80's made by Thomas Schelling (Schelling, 1971) and Robert Axelrod (Axelrod, 1997) Agent Based Modeling has proved to be a powerful simulation tehcnique in a variety of fields: biology, business, economics, technology, social sciences. Agent-based models are made up of multiple dynamically interacting rule-based agents that ussually have spatio-temporal coordinates in an environment that can create the real-world complexity. These characteristics make agent-based modeling approach an ideal candidate for our simulation tool.

Thinking in terms of algorithmic representation of agents and their interactions we found that object-oriented programming paradigm is the natural choice for micro-simulations, and therefore an object-oriented language should be used for this purpose. We considered several OO languages: Java, Scala, C++, and even R (it has some limited capabilities of object-oriented programming).

When we analyzed these languages in order to choose one, we considered the following criteria:

- it should have at least one free, open source compiler/interpreter;
- the compiler/interpreter should be available at least on the most widespread operating systems nowadays: Windows, Linux, MacOSX;
- it should have libraries needed to implement different features of the simulation software;

- the execution speed and memory requirements of the final executable program should allow users to run simulations with relatively large populations even on normal desktops/laptops;
- the resulting software should be portable on different computing platforms, including here cloud computing environments;
- free, open source tools like an IDE, a debugger, a profiler should be available;
- the time to production should be small enough considering the time-frame of the project;
- it should have a large community of programmers.

After a careful examination of all choices and taking into consideration our previous programming experience we decided to use C++. Even if all the analyzed languages largely meet the criteria taken into account, what we considered to be decisive in our choice was the speed of execution and the memory requirements (Stroustrup, 2013), (Stroustrup, 2018). Mobile phone datasets could be very large, they could have sizes from TB to PB, that's why we need implementations that have a reasonable runtime even for very large datasets.

A second choice that we had to take before designing any methodology and writing any piece of software was to decide what kind of sythetic data we want to obtain: event (signalling) data, CDRs, or both? Ideally, the answer is both. But we had to consider the limited resources that we have at our disposal and we decided to design and implement a synthetic data simulator that produces event / signalling data. The reason why we took this decision is based on the intrinsic characteristics of these data: they are produced with a high frequency, allowing us to calculate more precisely the statistical indicators of interest (location probabilities, number of devices, target population). Moreover, if we also need CDRs, they can be obtained by taking a sample from the signalling data sets. Nevertheless, in the near future we do intend to produce CDRs as an output of the data simulator.

In the following chapters we describe the methodological aspects involved in the generation of the synthetic data, the general structure and internal details of the software implementation, the structure of the input and output data. We also provide instructions on how to download, build and run the software using the example datasets provided together with the source code. The report ends with further directions of improvement that we intend to follow in the near future. We don't provide here details on technical concepts that underlie the functioning of mobile networks since these type of information were given in previous reports (WP5.3, 2018), (WP5.2, 2017).

We emphasize here that simulation software is only a part of the whole. Figure 1.1 shows the modular structure of our proposed end-to-end framework approach and the software packages that implement parts of this framework. Some of them are already developed, others are to be developed within the timeframe of the ESSnet project. We differentiate two approaches: a static and a dynamic one. From a static point of view the *mobloc* package Martijn Tennekes (2018) was already developed during the last ESSnet with the aim of computing the location probabilities of mobile devices starting from the network configuration parameters. Considering the dynamic behavior, we proposed a set of software tools that can be interfaced and that covers different parts of the methodological framework:

- *data simulator*: this is the subject of the present report. It simulates a population with mobile devices, generating the event data while the persons move around a map;
- *destim*: this is a prototype R package used to compute the location probabilities of the mobile devices from a dynamic point of view, which is based on Hidden Markov Models. This package was already tested feeding it with the synthetic data sets produced by the simulator;
- *pestim*: this is an R package already developed in the previous ESSnet that implements a hierarchical model in a Bayesian approach, with the purpose of estimating the target population starting from the numbet of the mobile devices provided by the aggregation and filtering module;

- *visualization software*: this is a set of software modules used to visualize in a user friendly way all the data that flow through the different modules in a dynamic fashion. It will start with the generation of network events, going to location probabilities of the mobile devices, the aggregated number of mobile devices and the final figures for the target population. These software modules will be developed during this project, part of them being already used to produce some visualizations included in this report;

- aggregation and filtering software: this piece of software (intended to be developed) will apply specific algorithms to detect devices of target population and their displacement patterns, allowing us to identify home and work locations, trips, commuting routes etc. It will use as a input the information on location probabilities produced by *destim* and the output will be passed to *pestim* package.

To obtain a coherent chain of tools, we kept the same concepts regarding the network configuration and the signal propagation as they were initially defined in the *mobloc* package, although enhancing them as the project advances.

Given the objectives of ESSnet on Big Data and the strict constraints on data access, the proposed approach and implementation could be considered without any reasonable doubt a scientific and practical breakthrough for official statistics. Also, the simulator could be the seed of inspiration for novel and innovative approaches in modern official statistics, as a boundary-breaking achievement.

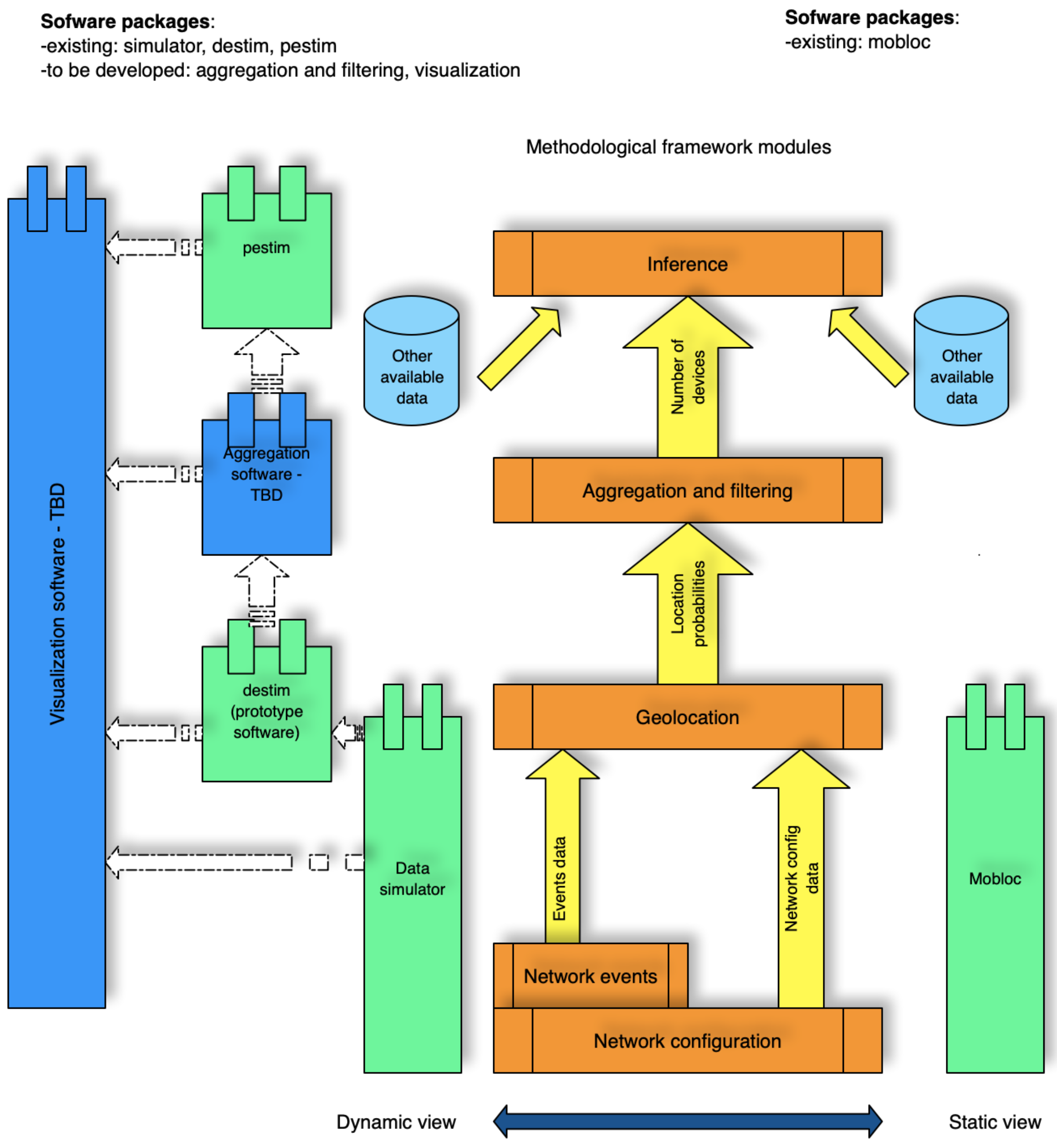


Figure 1.1: The software packages already developed or to be developed

2

The simulation software: design and implementation details

This chapter is devoted to provide a generic description of the proposed synthetic data generator. We start by presenting some methodological aspects upon which our synthetic data generator is based, then we describe the general structure of the software implementation together with some internal details. We emphasize the modular structure of the data generator tool which is a key element in any software implementation. Following a modular approach allows us to easily add new features or to change the implementation details of the already existing features without affecting the whole software (Bass et al., 2012).

The synthetic data generation software includes only the main functionalities needed to produce the desired output. Details such as, e.g., visualization techniques for the generated output datasets are implemented in other complementary tools that will be presented separately.

This tool is focused on the main steps that drive us from a synthetic population and the displacements of each individual in a certain time interval to a simple form of raw network event data and a set of measures/indicators computed using the raw data. From a logical point of view, our simulator is based on the following important modules, namely, (i) a GIS module, (ii) an agent based simulation module, and (iii) a computational module responsible for producing some measures of the interactions between mobile devices and network.

We emphasize again that this tool produces only a very simple form of raw network events data like a connection event or a disconnection event but they are enough for our statistical needs. In fact, the very complicated structure of the real mobile network events should be decomposed and transformed to such a simple form to be used for statistical purposes. The decomposition and transformation is a de facto requirement of any system simulation in order to capture only the essence of the process, i.e. the stochastic process of data generation by a mobile network and while the raw network events are simple, they are not simplistic, taking into account that they can be very easily modified to produce different datasets according to user needs.

In section 2.1 we provide some methodological aspects of the data generation process while in section 2.2 we describe the software implementation.

2.1. Methodological aspects

This section is dedicated to methodological details involved in the generation of the synthetic mobile network data. The following main aspects will be discussed:

- generation of the synthetic population;
- the model of the radio signal propagation used to describe the interaction between mobile devices and antennas;
- the displacement pattern of the individuals during a simulation;
- the generation of the network events;
- the general algorithm of a simulation cycle.

2.1.1. The synthetic population generation

A simulation involves a number of individuals composing a reference population. We start our simulation approach by considering only closed populations, i.e. the number of individuals moving inside the map during a simulation will be constant. Later developments will include also the possibility of having a varying number of individuals during the simulation process: some individuals will leave the map of the simulation (for example they get in an airplane simply dissapear from the map), some individulas will enter the map.

The number of the individuals in the synthetic population is provided by the user that conduct the simulation process using a configuration file. Together with the number of individuals, other personal characteristics will be provided too: gender, age.

Each individual will move from a location to another during the simulation with a certain speed, therefore, the (average) speed of individuals will be an input for the simulator. Nevertheles, specifying the exact speed for each individual could be cumbersome for simulations with a large number of individuals. A more realistic approach that we propose is to specify an average speed and then to generate the actual speed of each individual using a probability distribution based on the average speed. The method implemented in the software product uses in fact two different speeds, one that emulates walking, another one that emulates car transport and set, on the average, half of individuals to move with the lower speed and the other half with the higher speed. Based on these two average values, the speed of each individual is generated as a random value drawn from a normal distribution.

The displacement from one location to another during the simulation is modeled as a sequence of time intervals when the individuals move around the map and time intervals when they stay in the same location. Again, specifying the exact values for these intervals for each individual is not realistic for large populations, therefore they are generated from probability distributions: the stop interval from a normal distribution with the mean given by the user as an input and the interval between two stops from an exponential distribution with the mean provided also as an input.

Each individual involved in a simulation can own 0, 1, or 2 mobile devices. The exact number of mobile devices each individual has is decided using the value (0/1) of a random variable drawn from a Bernoulli distribution (a so-called Bernoulli trial). The parameters of the Bernoulli distributions will be specifiyed as general parameters of the simulation. A simulation can support one or two Mobile Network Operators (MNO). The actual number of MNOs is provided as an input.

The pseudocode for the proposed algorithm to generate the population involved in a simulation is given in Algorithm 1.

2.1.2. The radio signal propagation model

The radio signal propagation model is very important to describe the interaction between mobile devices and antennas. In mobile networks, two main types of antennas can be used: omnidirectional and directional antennas Panwar et al. (2016).

The pattern of the radio signal propagation differs: while for an omnidirectional antenna the signal propagates isothropic, having the same pattern in all directions, for a directional antenna the signal propagates through a preffered direction, with an angular aperture of aproximately 120 degrees.

We propose to use two measures of the radio signal:

- signal strength;
- signal quality;

We mention that we used all the concepts introduced by the *mobloc* R package Martijn Tennekes (2018) WP5.4 (2018) and the same mathematical formulas to compute the signal strength and quality in order to maintain a certain degree of compatibility with the previous results from the ESSnet project.

For omnidirectional antennas, the signal strength at a distance d from an antenna is given by:

$$S(d) = S_0 - S_{dist}(d) \tag{2.1}$$

Algorithm 1 Algorithm for synthetic population generation

```
Read $N$ - number of individuals
Read $speed\_walk$, $speed\_car$ - average speed for walking and for car transportation
Read $stop$, $interval$ - the average time interval for a stop and the average time interval between two stops
Read $age\_distribution$ - type and parameters of age distribution
Read $male\_share$ - the share of males in the final population
Read $numMNOs$ - the number of Mobile Network Operators
Read $p1$, $[p2]$ - the market share of the first (and optional the second) mobile phone operator
Read $p\_sec$ - the share of the individuals having 2 mobile devices
for i in 1:N do
    Generate $p$ - a Person object
    Generate $loc$ (a random location on the simulation map), set the initial location of $p$ to $loc$
    Generate $age$ using $age\_distribution$, set the age of $p$ to $age$
    Generate $gender$ - drawn from a Bernoulli distribution with parameter $male\_share$
    Set the gender of $p$ to $gender$
    Generate $s$ from a Bernoulli distribution with parameter $0.5$
    if $s = 1$ then
        Generate $sc$ - the speed of an individual from a normal distribution with the $mean = speed\_car$
        Set the speed of $p$ to $sc$
    else
        Generate $sw$ - the speed of an individual from a normal distribution with the $mean = speed\_walk$
        Set the speed of $p$ to $sw$
    end if
    Generate $interval$ - a random value from an exponential distribution, the time interval between two stops
    Set the time interval between two stops of $p$ to $interval$
    Generate $stop$ - a random value from a normal distribution, the time interval of a stop
    Set the time of a stop of $p$ to $stop$
    if $numMNO = 1$ then
        Generate $has\_phone$ - a random value from a Bernoulli distribution with parameter $p1$
        if $has\_phone = 1$ then
            Generate $m$ - a MobilePhone object
            Assign $m$ to $p$
        end if
        Generate $has\_sec\_phone$ - a random value from a Bernoulli distribution with parameter $p\_sec$
        if $has\_sec\_phone = 1$ then
            Generate $m2$ - a second MobilePhone object
            Assign $m2$ to $p$
        end if
    else
        Generate $has\_phone1$ - a random value from a Bernoulli distribution with parameter $p1$
        if $has\_phone1 = 1$ then
            Generate $m1$ - a MobilePhone object connected to the first MNO
            Assign $m1$ to $p$
        end if
        GenerAte $has\_phone2$ - a random value from a Bernoulli distribution with paramter $p2$
        if $has\_phone2 = 1$ then
            Generate $m2$ - a MobilePhone object connected to the second MNO
            Assign $m2$ to $p$
        end if
        if $p$ could have the a second mobile phone then
            Generate $m\_sec$ - another MobilePhone Object
            Assign $m\_sec$ to $p$
        end if
    end if
end for
```

where S_0 is the signal strength at $d = 1$m from the antenna and it can be computed using the power of the antenna (the power is an input parameter for the antenna under consideration):

$$S_0 = 30 + 10\log_{10}(P) \tag{2.2}$$

The second term in eq. 2.1 is a component describing the loss of the signal strength with the distance d from antenna and can be modeled as:

$$S_{dist}(d) = 10\log_{10}(d^{\gamma}) = 10\gamma\log_{10}(d) \tag{2.3}$$

where γ is called the path loss exponent or the attenuation factor and it normally has values between 2 and 6, depending on the environment: rural, urban or indoor. The value of this exponent is taken as an input parameter for our simulation software and in real life is determined by experimental measurements carried out by the Mobile Network Operator.

The signal for directional antennas has a more complicated model of propagation. It has the highest values of the strength along a predefined direction but the signal can also propagates along other directions too. We describe the propagation pattern using the following parameters (they as considered input parameters for the antenna under consideration):

- the azimuth or the direction angle ϕ_a is the angle in which the antenna is directed and it is measured from the North, clockwise;
- the tilt angle (or the elevation angle) θ_a;
- the horizontal beam width α_a, which specifies the angular aperture where the signal loss is 3dB or less;
- the vertical bean width β_a, which specifies the angular aperture in the vertical plane where the signal loss is 3dB or less;
- $azim\_dBback$ the signal strength at the back of the cell in the azimuth plane;
- $elev\_dBback$ the signal strength at the back of the cell in the elevation plane;

All these parameters are normally known by the Mobile Network Operator and could be provided to an interested user.

The signal strength at distance d from antenna is given by Tennekes et al. (2019):

$$S(d) = S_0 - S_{dist}(d) - S_{azimuth} - S_{elev} \tag{2.4}$$

where S_0 and S_{dist} are the same as in eq. 2.1. For $S_{azimuth}$ and S_{elev} we use the same model of computation as is implemented in *mobloc* package Martijn Tennekes (2018). We will not replicate here the formulas used in this R package to compute the signal strength for directional antennas, an interested reader can find all the detailes in Martijn Tennekes (2018), and in the source code of the the *mobloc* R package which is freely available.

The signal quality is a measure proposed by M. Tennekes in Martijn Tennekes (2018) and is defined as:

$$q(d) = \frac{1}{1 + exp(-S_{steep}(S(d) - S_{mid}))} \tag{2.5}$$

where S_{steep} and S_{mid} are two parameters that describe the steepness and the midpoint of the curve $qvs.S$. They will be considered as input parameters for our simulator and should be provided by the Mobile Network Operator that owns the antenna under consideration.

The simulator is configured with an input parameter to use either the signal strength or the signal quality for modeling the handover mechanism. When a mobile device tries to connect to an antenna, it will choose the one that provides the signal with the highest value for the strength/quality. In case that the selected antenna cannot connect the device (for example because it operates at its maximum capacity and it cannot accept another connection), the device will try to connect to the antenna with the second highest value for the strength/quality and so on.

2.1.3. The displacement pattern of the individuals

The displacement pattern of the individuals during a simulation is an input parameter of the simulator. Currently, it supports only two types of displacement patterns:

- random walk in a closed map;
- random walk with drift in a closed map.

Both patterns assume that the individuals can move only inside the borders of the map, i.e. the population is closed. This means that the number of individuals inside the boundaries of the map remains constant during the simulation.

The first pattern is a classical random walk where at each time step the direction is generated as a random value uniformly distributed in $[0, 2\pi]$ interval and the length of the step is detemined by the speed of each individual. If the next step would be outside the map, the person remains in the same position and at the next time instant a new displacement direction will be generated. The person will move only if the new final position along this new direction is inside the map, otherwise remains again in the same lcation. These steps repeat until a new position inside the map will allow the corresponding person to move. Algorithm 2 describes the procedure for calculating the next location at each time step of the simulation.

Algorithm 2 Random walk

```
procedure NEWLOCATION
    Generate a random value for the direction angle $theta$ from a uniform distribution
    $newX = x + speed \cdot \cos(theta) \cdot \Delta t$
    $newY = y + speed \cdot \sin(theta) \cdot \Delta t$
    if $(newX, newY)$ is within map then
        Set new location to $(newX, newY)$
    end if
end procedure
```

The second pattern contains a predefined trend direction of displacement ($theta$). During the first half of the simulation the direction of the displacement is generated as a random value from a normal distribution with the mean equals to $theta$ while during the second half the distribution mean changes to $theta + \pi$. $theta$ is a constant an can be seen as a parameter of the displacement pattern. The length of each step is determined as in the previous pattern by the speed of an individual. If the final location would be outside the map this step is not taken but a new direction is computed by adding a random value (drawn from a uniform distribution) to the trend direction angle $theta$ and the destination point is computed again. Algorithm 3 describes this procedure.

Either one or another pattern will be used, the individuals will take a sequence of steps for a time period then they will stop for another period of time. The actual time duration for the stops is generated from a normal distribution whose parameters are provided as input information. After each stop the duration of the next stop is generated from the same normal distribution. The time interval between two stops is generated from an exponential distribution whose parameter is provided as an input data. After each sequence of steps the new value for the next interval is generated from the same exponential distribution.

Algorithm 3 Random walk with drift

```
$change\_direction = false$
procedure NEWLOCATION
    if $current\_time \leq \frac{TS}{2}$ then
        Generate a random value for the direction angle $\alpha$ from a normal distribution with mean $theta$
    else
        Generate a random value for the direction angle from a normal distribution with mean $theta + \pi$
    end if
    if $change\_direction = true$ then
        Add a random value to $\alpha$
    end if
    $newX = x + speed \cdot \cos(theta) \cdot \Delta t$
    $newY = y + speed \cdot \sin(theta) \cdot \Delta t$
    if $(newX, newY)$ is within map then
        Set new location to $(newX, newY)$
    else
        $change\_direction = \neg change\_direction$
    end if
end procedure
```

2.1.4. Generation of the network events

The network events are generated at each step of the simulation when an individual changes his/her location and they are determined by the handover mechanism. Algorithm 4 describes how network events are triggered. The $TRYCONNECT$ procedure described in this algorithm is called by each mobile device at each time step of the simulation.

Each event generated during the interaction between the mobile device and an antenna is registered in a file by the antenna responsible for that event. At the end of the simulation, all the event files, belonging to all antennas of a mobile network operator are merged in a single file for later processing.

2.1.5. The general algorithm of a simulation cycle

Assembling all the pieces described so far, we can build an algorithm that describes a simulation cycle. Algorithm 5 gives a simple overview of the simulation process. During the simulation process, the events generated by the interaction between mobile devices and antennas are saved as well as the exact positions of all individuals.

2.2. The software implementation

2.2.1. The structure of the software

All the algorithms described in section 2.1 were implemented in a software package developed using the C++ language. We already exposed our rationale behind the decision of choosing this language in chapter 1. Another decision taken when we started the implementation was whether to use a GIS library to perform the geometric operations or to implement them directly in our software. We identified several operations that have to be performed with geometric shapes (objects) :

- build and work with different geometric objects such as points, polygons, circles;
- compute the distance between two points;
- check if a point is located inside a polygon;
- compute intersection between several polygons;

Algorithm 4 Generation of network events

```
procedure TRYCONNECT
    $p$ - the current location of the mobile device
    if device is already connected to an antenna then
        if the mobile phone is no longer in the coverage area of this antenna then
            disconnect the phone
            Register event: disconnection
        end if
    end if
    Identify the antenna with the highest value for signal strength/quality - A
    Try to connect to A
    if connection was succesful then
        if the device was connected to the same antenna A in the previous time step then
            Register event: connection to the same antenna
        else
            Register event: new connection
        end if
    else
        Register event: device in coverage area but not connected
        Build an ordered list of antennas (by signal strength/quality) that covers the device location
        Iterate through the list of antennas and try to connect to the antenna from the head of the list
        if the connection was succesful then
            Register event: new connection
        else
            Remove the antenna from the head of the list and take the next antenna from the list.
        end if
    end if
end procedure
```

- reading and saving the geometric shapes in standard file formats (coverage areas, maps);
- loading real maps that are available in standardized formats;

Considering the complexity of some geometric operations and the interoperability with other specialized software when working with the maps of the simulation, we chose to use a GIS library and evaluated several open source libraries: *GDAL* GDAL/OGR contributors (2019), *GEOS* GEOScontributors (2019) , *GRASS* GRASS Development Team (2017). All these libraries provide the features that we need for our simulator, but some of them are very complex and difficult to use. We decided to use the *GEOS* C++ library because it is (relatively) easy to compile, install and use comparing to *GDAL* or *GRASS*. *GEOS* is in fact a limited C++ port of the Java Topology Suite (JTS) library (see `https://www.osgeo.org/projects/jts/`)and offers a C and C++ API.

The simulation software was implemented in a modular approach, following a layered architecture where each layer provides specific functionalities to the other layers above it. The architecture of the software is depicted in Figure2.1 and is made up of five main layers. The first layer **Basic libraries and utilities** is composed by the standard C++ runtime libraries and some other utility libraries or classes:

- STL - the Standard Template Library used throughout the entire software;
- XML Parser - provides methods to parse XML files;
- CSVParser - provides methods to read and write `.csv` files;
- RandomNumberGenerator - provides methods used to generate random number according to specific distributions.

Algorithm 5 Simulation process

```
Initialize the map of the simulation
Overlap a grid on the map
Generate a synthetic population according to algorithm 1
Read the technical parameters of antennas and place the antennas on the map
Save the coverage areas of all antennas
Compute and save the signal strength / quality for each tile of the grid
for ($t = t_0; t \leq t_f; t = t + time\_step$) do
    for each individual $p$ in the population do
        Move $p$ to a new location according to algorithms 2 or 3
        Call TryConnect() procedure for each mobile device of $p$
        Register the exact time and location of $p$
    end for
end for
```

Next, it comes the **GIS layer** which is built around the *GEOS* C++ library and is responsible for implementing all the geometric operations and to offer support for working with maps. The third layer called **Data Encapsulation Layer** provides encapsulation for the main objects of our simulation software: antennas, individuals, mobile network operators, mobile devices. They act as agents and they are created based on some input configuration files described in chapter 3. The logic of the handover mechanism and the displacement patterns are also encapsulated in these objects. The **Simulation Layer** is responsible with running the actual simulation: each individual with or without a mobile device will move inside the map of simulation according to a pattern and at each time step of the simulation the interaction between mobile devices and antennas will generate network events that are saved for later processing. The top layer (**Computations Layer**) implements a simple method to compute the location probabilities of each mobile device during the simulation.

The data flow of the simulation process is presented in Figure2.2. The simulation software reads a series of input files and produces several output files. Except with the map where the simulation take place, all the input files are *XML* files. The map is a *WKT* (Well Known Text) file, a standard format used by the GIS software and supported by the *GEOS* library used in our simulator. The output files are all saved in `.csv` format. We decided to use this format having in mind that these files are intended for a later processing step and the `.csv` format can be easily read by any programing language that will be used in this stage. In the following we give a short overview of each input and output file, a detailed description of the structure of each file being provided in chapter 3.

There are five input files that should be provided to the simulation software:

- a Map file - it is the map where the simulation take place. Currently this is a *WKT* file that contains the external boundary of the geographical area where we run the simulation;
- a file with the general parameters of the synthetic population - named `persons.xml` in Figure2.2. This file contains information like the number of persons in the synthetic population, the age distribution, the share of males/females, the speed of the displacement, the share of the population starting from the same initial position in each simulation;
- a file with the location and the technical parameters of the antennas;
- a file with parameters for the simulation process. This file contains general parameters of the simulation: the initial and the final time instant of the simulation as well as the time increment, information about the mobile network operators, the type of the displacement of persons, the type of the handover mechanism, the dimensions of a tile in the grid and the name of some of the output files;
- an optional file that provides information necessary to compute the location probabilities of the mobile devices.

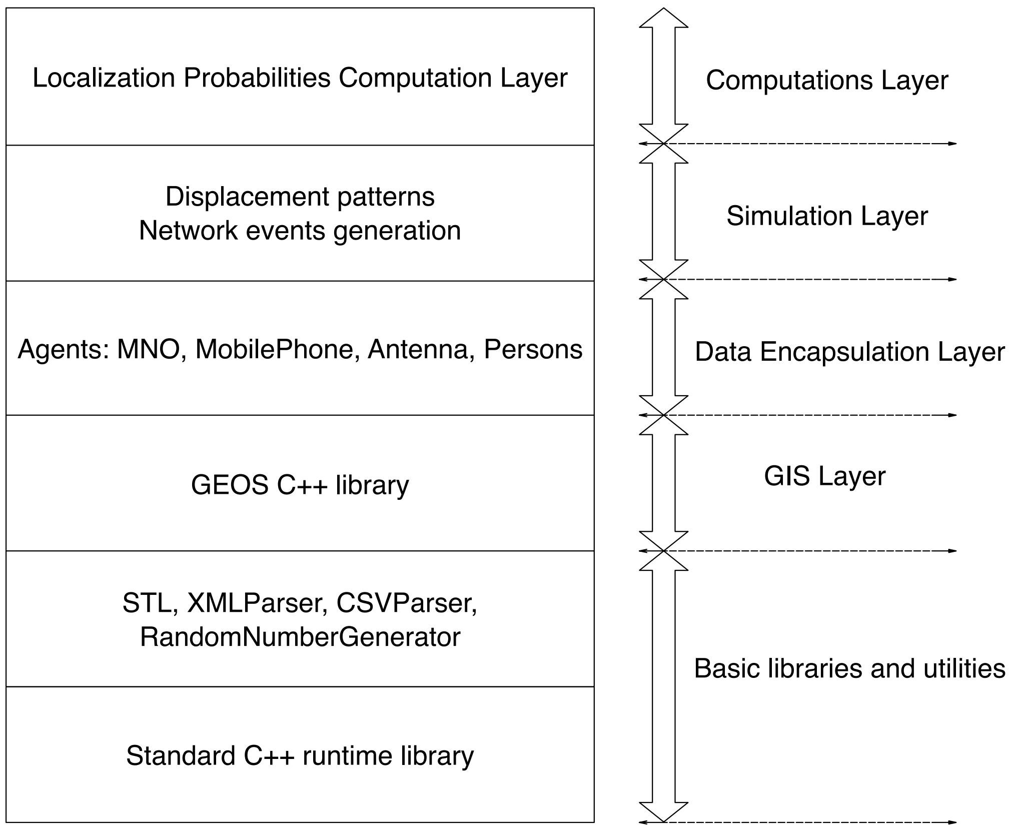

Localization Probabilities Computation Layer
Displacement patterns
Network events generation
Agents: MNO, MobilePhone, Antenna, Persons
GEOS C++ library
STL, XMLParser, CSVParser,
RandomNumberGenerator
Standard C++ runtime library
Computations Layer
Simulation Layer
Data Encapsulation Layer
GIS Layer
Basic libraries and utilities

Figure 2.1: The layered structure of the simulation software

The software outputs the information generated during the simulation in six files:

- a grid file the contains the full description of the rectangular grid overlapped on the map, named `grid.csv` in Figure2.2;
- a file that stores the antennas location on the map and grid at the initial time instant, named `antennas.csv` in Figure2.2;
- a file that contains the exact position on the map and grid of each person, at each time instant of the simulation, named `persons.csv` in the same figure;
- a file where is saved the signal strength or quality, depending on the handover mechanism selected for the center of each tile of the grid;
- a file that contains the coverage areas for each antenna, named `Antenna_cells.csv` in Figure2.2; item a file that contains the events generated by the interaction between mobile devices and antennas, together with the exact location of the mobile devices that generated the events, named `AntennaInfo.csv` in Figure2.2.

2.2.2. The component classes

A list of all classes is presented in table 2.1 together with a brief description of each one. The most important classes of the simulation software are detailed in the following.

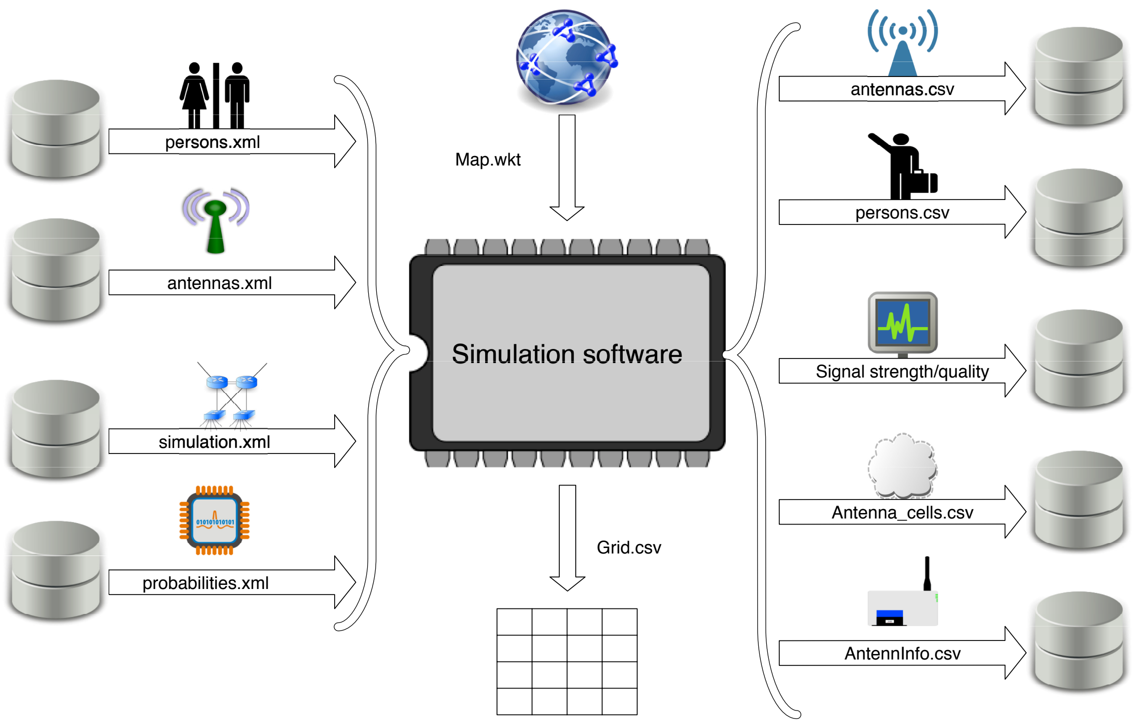


Figure 2.2: Data flow diagram

Agent

This is an abstract class, the base class for all agents involved in a simulation: persons, antennas, mobile devices, mobile network operators. Figure2.3 shows the inheritance diagram for this class and Figure2.4 shows the collaboration diagram. The constructor of this class takes a pointer to the map and the clock objects of the simulation and it has an unique ID that identifies the agent throughout the entire simulation. It also declares a method `toString()` implemented by all subclasses that gives a human readable string representation of the class. Detailes about all subclasses are provided below.

MobileOperator

This class represents a Mobile Network Operator. A mobile network operator object owns a set of antennas and has a set of mobile phones subscribed to it. It also has name and a market share read from the general simulation configuration file and passed to the constructor of the class.

The `MobileOperator` class is responsible to open the file where the coverage area of all the antennas that belong to this MobileOperator object are saved and closes it at the end of simulation. The name of this file is built concatenating `AntennaCells\_` with the name of the Mobile Network Operator. A line of this file contains the antenna ID followed by a *WKT* text representing the coverage area of antenna. It also opens the file where the signal strength / quality values computed in the center of each tile of the Grid object for all antennas that belong to this MobileOperator object are saved and closes it at the end of simulation. The name of this file is built concatenating `SignalQuality\_` with the name of the Mobile Network Operator. A line of this file contains the antenna ID followed by a set of values for the signal strength / quality computed in the center of each tile of the grid. Currently a simulation can be run with 1 or 2 mobile network operators.

Class name	Description
Agent	an abstract class, the base class for all agents involved in a simulation
AgentsCollection	a container for all the agents involved in a simulation
Antenna	this class simulates an antenna of the mobile phone network
AntennaInfo	it encapsulates all the information about an event generated by an antenna
Clock	the clock used to synchronize all activities during the simulation
Constants	defines some constants used in the process of the simulation
CSVParser	read and parse a csv file, write a csv file.
EMField	singleton class is used to compute different measures of the electromagnetic field radiated by an antenna
Grid	a grid of rectangular tiles overlapped on the map of the simulation.
HoldableAgent	the superclass for all agents that represent a device that can be held by a person
IDGenerator	a singleton class used to generate unique identifiers for all agents in the simulation
ImmovableAgent	an agent that can have a location on map but it cannot move
InputParser	it parses the command line and extract the parameters and their values.
LocatableAgent	it extends the Agent class and defines an object with a location on a map
Map	the map where the simulation takes place
MobileOperator	it represents a Mobile Network Operator company
MobilePhone	it represents a mobile phone.
MovableAgent	represents an Agent that can move inside the map
Person	it represents a person that can have 0,1 or 2 mobile phone(s)
RandomNumberGenerator	a singleton class used to generate random numbers according to different distributions
Tablet	it represents a Tablet with a SIM card - not yet implememted
TinyXML2	parses XML files
World	the class where the simulation process takes place

Table 2.1: A list of the classes

LocatableAgent

This class extends the `Agent` class and defines an object with a location on the map of the simulation. It provides a method `dumpLocation()` inherited by all its subclasses that produces a human readable string representing the location.

ImmovableAgent

This class extends the `Agent` class and represents an agent that can have a fixed location on map. Currently, the only subclass of it is `Antenna`.

Antenna

This class simulates an antenna of the mobile phone network. The collaboration diagram `Antenna` is presented in Figure2.5

An `Antenna` object receives all the technical parameters and the location on the map from the configuration files and it is responsible to register all the events generated during the interaction with the mobile devices. It has methods to compute the signal strength / quality as a function of distance and the coverage area which is saved in a file. An `Antenna` object keeps a pointer to the `MobileOperator` object that owns it. The constructor of the class also receives a pointer to the `Clock` object of the simulation and use it to

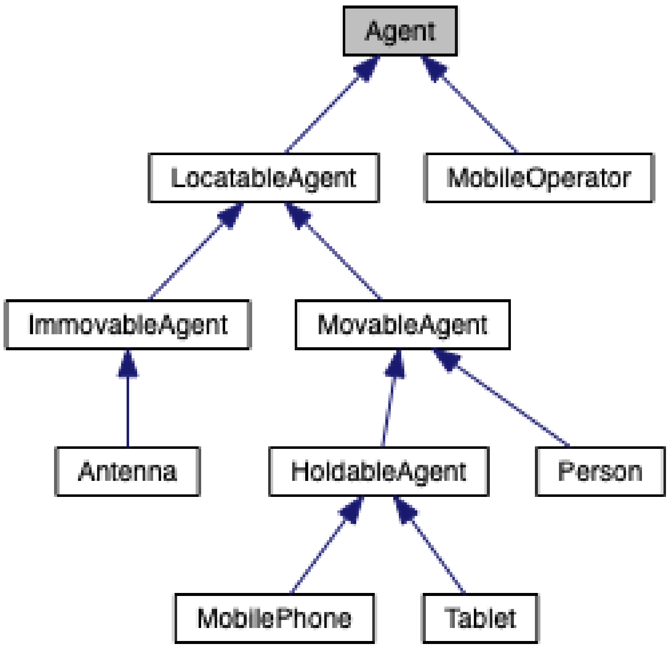


Figure 2.3: Agent class inheritance diagram

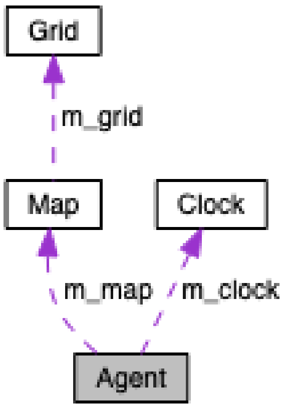


Figure 2.4: Agent class collaboration diagram

save the timestamp of each event. There are two different types of antennas supported by the software, omnidirectional and directional, with different patterns of the radio signal propagation.

MovableAgent

This is an abstract class and it represents an `Agent` that can move inside the map. It is used as a base class for all the agents that can move. Currently `Person` and `MobilePhone` classes inherits it. The speed of displacement is kept as a member of this class and it declares a pure virtual function `move(MovementType type)` used to displace the agent to a new location on the map. The parameter of this method defines the displacement pattern. Currently, two values are accepted for this parameter: `MovementType::RANDOM_WALK_CLOSED_MAP` and `MovementType::RANDOM_WALK_CLOSED_MAP_WITH_DRIFT` In section 2.1.3 we explained the pattern of the displacements for both cases: pure random walk and random walk with drift.

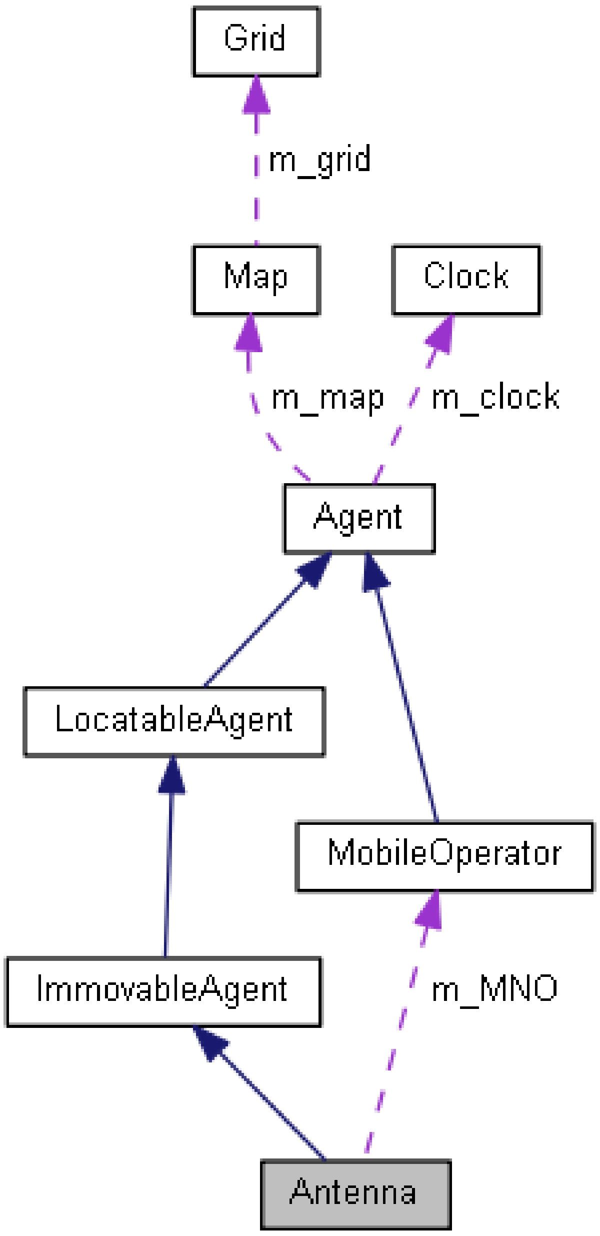


Figure 2.5: Antenna class collaboration diagram

Person

This class represents an individual that can have 0, 1 or 2 mobile device(s). During the simulation the person move around the map, carrying his/her mobile devices. The displacement on the map is a implemented as a sequence of time intervals when the individual move followed by a stop in a fixed position. The constructor of this class takes a pointer to the `Map` and `Clock` objects of the simulation, an initial position on the map, age, gender, the initial speeed of displacement, the average time of a stop and the average time interval between two consecutive stops. It implementes the `move()` method inherited from `MovableAgent` class as described previously and provides methods to add mobile devices, to get the number of these devices.

HoldableAgent

This is the superclass for all agents that represent a device that can be held by a person and it is responsible to set the handover mechanism of the device. There are two types of handover mechanisms already supported: based on the maximum signal strength or on the maximum signal quality provided by the surrounding antennas. It declares a pure virtual function `tryConnect()` that is called every time an agent sets the location on map. An object of class `HoldableAgent` stores a pointer to its owner (a `Person` object for example) and a pointer to the `Antenna` object where the agent is connected.

MobilePhone

This class represents a mobile phone and is a subclass of `HoldableAgent`. A mobile phone is owned by a `Person` object and it moves on the map together with its owner. While moving, at every time step of the simulation it calls `tryConnect()` and tries to connect to an antenna. The event generated by this interaction is saved by the corresponding `Antenna` object in a file. The constructor of this class receives pointers to the `Map` and `Clock` objects of the simulation, a pointer to the `Agent` object that owns it, the initial location on map (which is the same with the location of the owner), the type of handover mechanism to use, and the minimum value of the signal strength / quality that can be used to connect to an antenna. This class also has a method to set the `MobileOperator` object where the mobile phone has a subscription.

AgentsCollection

This is actually a container for all the agents used in a simulation. An agent could be an object of one of the derived classes of `Agent`. The `Agent` objects are kept internally in an `unordered_multimap` as pairs `<string, Agent*>` where the first element of the pair is the name of the concrete agent (a person, a mobile device, an antenna, a mobile network operator, etc.) and the second element is a pointer to the actual object. This class has methods to add and delete agents to the collection, to retreive an agent by its ID, to iterate through agents, to extract a list of specific agents (for example all the `Antenna` objects or all the `Person` objects etc.) The collaboration diagram of the `AgentsCollection` class is shown in Figure2.6.

Map

This is the map where the simulation takes place. It could be as simple as a rectangle or any kind of geometry object(s) read from a *WKT* file. The map defines the external boundary of the geographic area where the simulation takes place and it keeps internally a `Geometry` object, a factory object of `GeometryFactory` type used to create other geometric objects and a `Grid` object overlapped on the map. All geometric and geographic features of the simulator uses the *GEOS* C++ library. The collaboration diagram of this class is presented in Figure5.1.

Grid

This class implements a grid of rectangular tiles overlapped on the map of the simulation. The grid is used to compute the *location probability* of a mobile phone. This means that we compute the probability of a mobile device to be located in specific tile of the grid using the event data recorded by each antenna during the simulation. A finer grid will give a more accurate location but the computational cost increase when the size of the tiles decrease. The tiles of the grid are indexed starting with 0 for the tile in the bottom left corner of the grid in a row-major ordering. The last tile, with the biggest index, is the tile in the upper-right corner of the grid. The dimensions of a tile are given in the simulation configuration file.

Clock

This is the clock used to synchronize all the activities during a simulation. All the agents and all other objects involved in a simulation use the same `Clock` object. The `Clock` is initialized with the value of the starting and ending time of the simulation given in the configuration file and keeps the current time during the simulation. At each step of the simulation the current time is increased by an increment also read from the configuration file. Starting time, ending time, current time and the time increment are only conventional units and they do not depend in any way on the real clock of the computer.

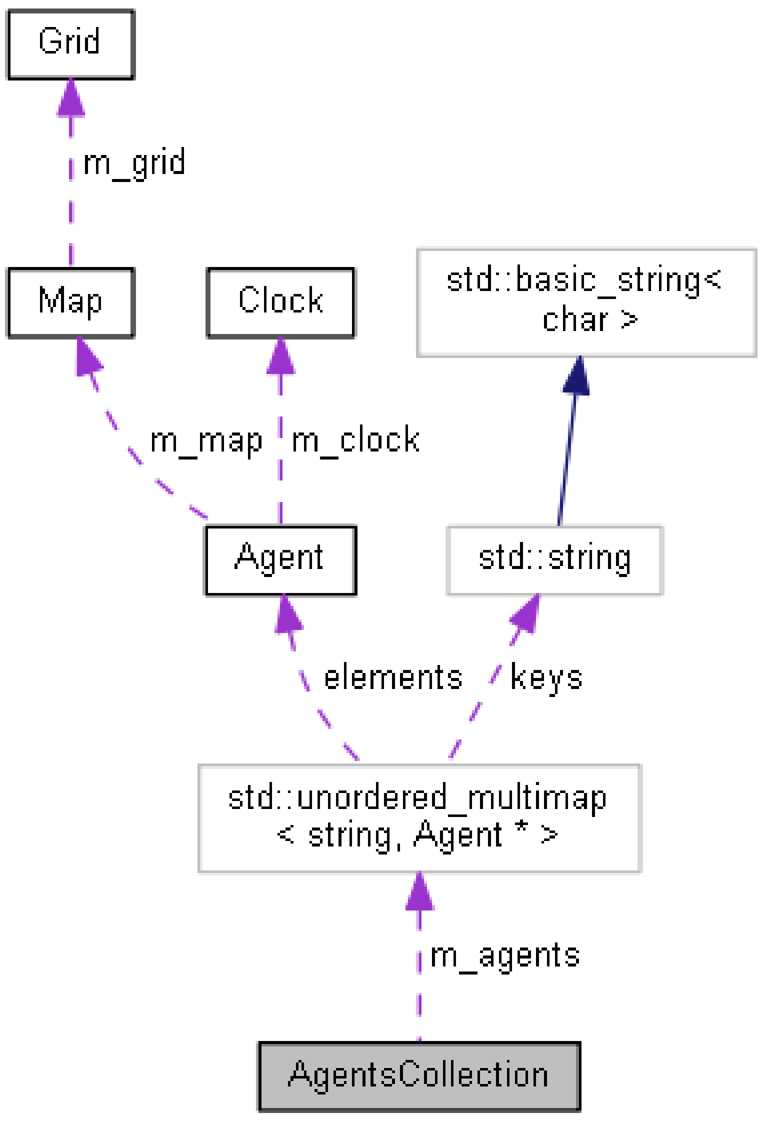


Figure 2.6: AgentsCollection class collaboration diagram

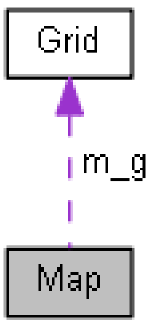


Figure 2.7: Map class collaboration diagram

World

This is the class where the whole simulation process takes place. A `World` object has a `Map`, a Clock, a set of `Agents` than can be persons, mobile phones, antennas, mobile operators etc. The constructor of this class builds a new `World` object, reading the parameters for the persons and antennas from *XML* configuration files provided in the command line of the simulator. The general parameters of the simulation (duration, the displacement pattern, the handover mechanism, etc.) are also read from an *XML* configuration file. The main method of this class is `runSimulation()`. It starts the clock and at every time step iterates over all persons, computes the new location and change their location to the new one. Once the new location is set, each mobile device calls `tryConnect()` which in turn generates a set of network events saved in output files by antennas. The collaboration diagram of this class is presented in Figure2.8

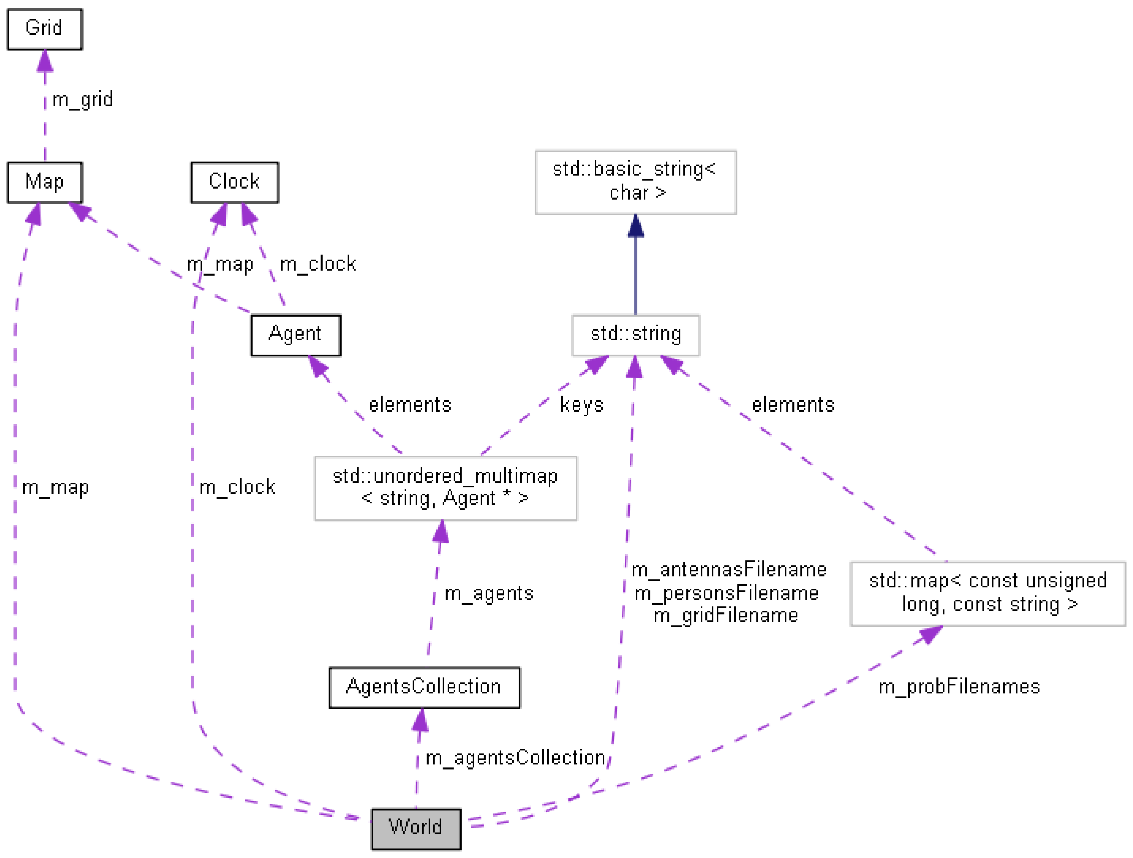


Figure 2.8: World class collaboration diagram

AntennaInfo

This class is used to encapsulate all the information about a network event generated by the interaction between a mobile device and an antenna: the timestamp, antenna ID, event code, the device ID, the exact location of the event, the tile index of this location.

EMField

This utility singleton class is used to compute different measures of the electromagnetic field radiated by an antenna (signal strength, signal quality) in given locations on the map. It also has methods to build decreasing ordered lists of antennas by signal strength / quality to be used in the handover process, to check if a location is within the coverage area of an antenna and to build lists with all the antennas that cover a specific location. All these methods are used either in the handover procees, when a mobile phone connects to an antenna or in the computation of event locations.

In summary, the simulator follows very closely the current paradigm of carrying discrete-event system simulation and we can easily identify our entities(world, individuals, mobile devices, antennas), their attributes(id's, coordinates of the position, speed, gender, age, signal strength, signal direction), activities carried by entities (moving), the events (tryConnect), the states(connected or disconnected) and the clock. Also,providing the modular and scalable approach needed for further developments. Is relatively easy to implement new classes, by expanding the ones already present, and introduce complexity by designing new algorithms to model individuals and network sophisticated patterns of behavior.

3
A description of the input and output files

In this chapter we provide a detailed description of each input and output files of the simulator. Example input files can be found in `data` directory of the source code tree.

3.1. Input files

There are five input files needed by the simulation software, one of them being optional:

- a map file;
- a configuration file with the technical parameters and location of each antenna;
- a general simulation configuration file;
- a configuration file for the synthetic population used for simulation;
- an optional configuration file needed to compute the location probabilities;

These files are provided to the simulation software using command line parameters, as it will be shown in chapters 4 and 5. We decided to split the input parameters in several files, each concerning one aspect of a simulation to keep the level of modularity high.

3.1.1. The map file

A simulation takes place in a certain geographical area which is specified using a map. The input map is a file in *WKT* (Well-Known-Text) format. *WKT*(Well-Known-Text) is a text markup language used as a standard method for representing geometric shapes in vectorial format and is used by most of the geographical information systems. It also supports specifying the spatial reference systems of spatial objects. While there is also a binary equivalent, known as well-known-binary format (*WKB*) we decided to use WKT because it is human readable and easy to manipulate, even with a very simple text editor. *WKT* allows users to define different geometric shapes such as: Point, MultiPoint, LineString, MultiLineString, Polygon, MultiPolygon. The *WKT* format was initially defined by the Open Geospatial Consortium (OGC) and described in their Simple Feature Access Open Geospatial Consortium (2011) and the current standard format is defined by ISO/IEC 13249-3:2016 ISO/IEC (2016).

In the current stage of development, the simulation software uses only the external boundary of the geographic area considered which is defined as a POLYGON object in the map file. Below we list the content of the `map.wkt` file which can be found in `data\dataset1` directory in the source code tree:

```
POLYGON ((0.0 1000.0, 1000.0 0.0, 3000.0 2000.0, 5000.0 0.0, 7500.0 2000.0,
9000.0 0.0, 10000.0 1000.0, 10000.0 10000.0, 0.0 10000.0, 0.0 1000.0))
```

This a a very simple polygon created with a text editor. More complex polygons that bounds real world regions can be created using `https://www.openstreetmap.org/` set of free maps and tools. The map file provided in `data\dataset2` directory is such a map that covers a region of Madrid and it was extracted from `https://www.openstreetmap.org/`.

3.1.2. The general simulation configuration file

A simulation is defined using a set of parameters which are provided by the means of an input file in *XML* format W3C (2019). In all example datasets provided with the source code, this file was named `simulation.xml`. Below we listed the content of the `data\dataset1\simulation.xml` file and we will use it to describe each input parameter.

```
<simulation>
    <start_time>0</start_time>
    <end_time>200</end_time>
    <time_increment>1</time_increment>
    <time_stay>10</time_stay>
    <interval_between_stays>25</interval_between_stays>
    <mno>
        <mno_name>MNO1</mno_name>
        <prob_mobile_phone>0.35</prob_mobile_phone>
    </mno>
    <mno>
        <mno_name>MNO2</mno_name>
        <prob_mobile_phone>0.35</prob_mobile_phone>
    </mno>
    <prob_sec_mobile_phone>0.15</prob_sec_mobile_phone>
    <movement_type>random_walk_closed_map_drift</movement_type>
    <connection_type>strength</connection_type>
    <conn_threshold>-80</conn_threshold>
    <grid_file>grid.csv</grid_file>
    <grid_dim_tile_x>500</grid_dim_tile_x>
    <grid_dim_tile_y>500</grid_dim_tile_y>
    <persons_file>persons.csv</persons_file>
    <antennas_file>antennas.csv</antennas_file>
    <random_seed>12</random_seed>
</simulation>
```

The input parameters are enclosed in `<simulation>` and `</simulation>` tags. The first three parameters defines the timeframe of a simulation:

- the value enclosed by `<start_time>` and `</start_time>` specifies the starting moment of the simulation;
- the value between `<end_time>` and `</end_time>` specifies the time instant when the simulation ends;
- the value enclosed by `<time_increment>` and `</time_increment>` defines the time increment of the simulation.

This means that the simulation clock is initialized with the `start_time`, at every time step of the simulation the current value of the clock is incremented with `<time_increment>` and when the current value of the clock reaches the `<end_time>` the simulation stops. The units for these values of time are seconds by default and the units of all other geographical coordinates and speeds of movement are m/s.

These units could be changed with the provision that the user should correlate the values of time, coordinates, and speeds in order to produce some realistic simulations.

The movement pattern of the individuals involved in a simulation is composed of a sequence of displacements and stops. This means that an individual will move for a period of time (i.e. he/she will change position on map during this time interval) and then he/she will stop in a fixed position for another period. The average value of the stop period for the individuals in a simulation is specified using `<time_stay>` tag. This value has the same units as `<start_time>`, `<end_time>`, and `<time_increment>`. The actual value for the stop time for each individual is a random value drawn from a normal distribution with the mean equals to `time_stay` parameter and the standard deviation equals to $0.2 \times$ `time_stay`. The time interval between two stops, i.e. the time interval while an individual moves is also random value but this is drawn from an exponential distribution with the mean given by `interval_between_stays` parameter.

The simulation software supports one or two mobile phone operators. We are aware that there are more than two mobile network operators in almost every European country, but we support only two not to complicate very much the simulation process. A set of parameters should be provided in this configuration file for each of the mobile network operator involved in the simulation. `<mno>` and the corresponding ending tag `</mno>` enclose the required input information for a mobile phone operator which means that we will have to repeat `<mno>` and `</mno>` tags as many mobile operators as we want to introduce in our simulation (in our case, we can repeat them maximum two times). Two information are required for each mobile network operator. Firstly, a name is mandatory to identify the operator and this name is provided using `<mno_name>` and `</mno_name>` tags. This name will identify the operator throughout all simulation scenario. The second parameter of a mobile network operator is the probability for an individual to have a mobile phone with a subscription to it. This is equivalent to the mobile network operator market share and is provided using `<prob_mobile_phone>` and `</prob_mobile_phone>` tags. For example, 0.35 in the example file means that on the average, 35% of the individuals from the whole population will own a mobile phone with a subscription to one mobile network operator.

An individual involved in a simulation could have 0, 1, or 2 mobile phones. While the ownership of the first mobile phone is controled by the `prob_mobile_phone` parameter with a different value for each mobile network operator, the ownership of the second mobile phone is determined by the `prob_sec_mobile_phone` parameter which is common for the entire population. The value of this parameter gives the probability of an individual that already has a mobile phone to have a second mobile device. In the example file, 0.15 means that 15% of the individuals with a mobile phone will have a second mobile phone too. In real life it is possible, but very unlikely, for an individual to have more than two mobile phone, but we ignore this case to simplify our simulation process.

With the `<movement_type>` tag, the user can set the type of the displacement pattern. Currently, two values are allowed here:

- `random_walk_closed_map`
- `random_walk_closed_map_drift`

The first value, `random_walk_closed_map`, means that the displacement of each individual is described as a pure random walk, i.e. at each time step of the iteration a new angle of displacement is computed, and the next step is along this direction. The length of the step is determined by the speed of the individual, $step\_length = speed \times time\_increment$. The direction is generated as a random value drawn from a uniform distribution with the limits 0 and $2 \times \pi$. The individuals are confined to move only inside the boundary of the map. When the next location is computed for each individual, if some individuals have positions that would be outside the map, they will not move to this position but stay in the same (old) position until the next time step of the simulation when a new direction of displacement will be generated. The second value, `random_walk_closed_map_drift` is a modified version of the previous method where at each time step the direction of displacement is a random value drawn from a normal distribution with a predefined mean and a standard deviation equals to 0.1 radians. The mean of this distribution is $\frac{3 \cdot \pi}{4}$ during the first half of the simulation period and $\frac{5 \cdot \pi}{4}$ during the second half. If a step would be outside the map, the oposite direction is taken and the coresponding individual moves only inside the map.

The values for the parameters defined by `connection_type` and `conn_threshold` tags are to be interpreted in close connection to each other. `connection_type` defines the handover mechanism and it could take 2 values:

- `strength`;
- `quality`.

The first value of this parameter means that a mobile device will use the value of the signal strength to connect to an antenna while the second value means that the signal quality is used to connect. Both measures of the radio signal where defined in chapter 2. The `conn_threshold` tag is used to define the minimum value of the signal strength or signal quality that is considered usable by a mobile device to establish a connection. A the value below this limit is considered only noise and cannot be used by a mobile device to connect to an antenna. The concrete value is interpreted in relation to the value specified for the handover mechanism. If this is based on signal strength, then the value defined by `conn_threshold` should be read as the minimum signal strength that can be used to establish a connection and if the handover mechanism is based on signal quality the value of `conn_threshold` should be read as the minimum signal quality below which the radio signal is considered noise and cannot be used by a mobile device for connection purposes.

To reduce the computational complexity of our procedures, we use a rectangular grid of tiles overlapped on the map to compute the location probabilities of mobile devices. It is not be feasible from a computational point of view to work in a continuos time-space approach, that's why we base our simulation on a discrete time and space approach. The location probabilities of the mobile devices are computed only for the center of the tiles composing the grid. The dimensions of the tiles on OX and OY axes are defined using `grid_dim_tile_x` and `grid_tile_dim_y` tags. The same measurement units as for the map boundary should be used for tile dimensions too. Some of the parameters of the grid are computed by the simulator based on the bounding box of the map boundary. All these parameters, together with the tile dimensions are saved in a `.csv` Y. Shafranovich (2016) file for later processings. The name of this file is given using `grid_file` pair of tags. A finer grid means a better location probability but it also means a higher computational demand put on the computer that runs the simulation.

The `persons_file` tag is used to define the name of an output file, written by the simulator in `.csv` format where the exact position of each individual is saved during the simulation period. The exact structure of this file is described in section3.2.

`antennas_file` tag also defines the name of an `.csv` output file where the exact location of each antenna together with other parameters are saved. A complete description of the structure of this file is given in section 3.2.

The last tag in this configuration file, `random_seed`, is used to specify a random seed to initialize the random number generator needed by the simulator. We provide a constant value for this seed since some of the random initial locations generated for each individual should be reproductible across several simulations.

3.1.3. The antennas configuration file

The technical parameters and the exact location of each antenna are specified in a configuration file in XML format. In the example input files that we provide, the antennas configuration file is named `antennas.xml` in all datasets.

Below we list a part of the `data\dataset1\antennas.xml` file where we retained the technical parameters for 2 antennas: the first one is an omnidirectional antenna, the second one is for a directional antenna. In the following we provide a description of each paramter.

```
<antennas>
    <antenna>
        <mno_name>MNO1</mno_name>
        <maxconnections>100</maxconnections>
        <power>10</power>
        <attenuationfactor>3.55</attenuationfactor>
        <type>omnidirectional</type>
        <Smin>-80</Smin>
        <Qmin>0.5</Qmin>
        <Smid>-92.5</Smid>
        <SSteep>0.2</SSteep>
        <x>0</x>
        <y>7500</y>
    </antenna>
...
     <antenna>
        <mno_name>Orange</mno_name>
        <maxconnections>100</maxconnections>
        <power>10</power>
        <attenuationfactor>3.55</attenuationfactor>
        <type>directional</type>
        <Smin>-80</Smin>
        <Qmin>0.5</Qmin>
        <tilt>5</tilt>
        <azim_dB_back>-30</azim_dB_back>
        <elev_dB_back>-30</elev_dB_back>
        <beam_h>65</beam_h>
        <beam_v>9</beam_v>
        <direction>135</direction>
        <Smid>-92.5</Smid>
        <SSteep>0.2</SSteep>
        <x>5000</x>
        <y>5000</y>
        <z>10</z>
     </antenna>
</antennas>
```

The `mno_name` pair of tags encloses the name of the mobile network operator that owns the current antenna. It should match one of the names given in the general simulation configuration file (see subsection 3.1.2) with the same pair of tags, otherwise the simulator software will raise an exception and the simulation will stop.

An antenna could connect a number of mobile phones up to a maximum value, known as the antenna capacity. When this maximum number of connections is reached, no other incoming requests for connection received from mobile devices could be accepted. The value of the maximum number of connections supported by an antenna is defined using `maxconnections` pair of tags.

`power` specifies the power of the antenna in Watts.

The external environment of an antenna can be described using an attenuation factor of the radio signal. This factor is used to compute the decrease of both the signal quality and signal strength with the distance from the antenna location. The value of the attenuation factor is usually computed by the mobile network operator using field measurements and it has values between 2 in open field and 6 inside buildings. In the antenna configuration file, the attentuation factor is given using the `attenuationfactor` pair of tags.

The simulation software supports two types of antennas:

- omnidirectional;
- directional.

The exact type of an antenna is given by the value of the `type` parameter (tag). Here the user could provide either `omnidirectional` or `directional`.

Smin parameter defines the minimum value of the signal strength that can be used by mobile devices to connect to an antenna and Qmin defines the minimum value of the signal quality used for the same purpose. We provide both parameters here, but only one is used in the actual computations, the one that matches the handover mechanism set in the general simulation configuration file by the connection_type parameter. Smin and Qmin are also used to compute the coverage area of an antenna as the area where the signal strength or the signal quality (depending on the handover mechanism used in simulation) is greater than the minimum value set by these parameters. The coverage area is then saved using a *WKT* representation in an output file described in section 3.2.

Smid and SSteep are the parameters involved in the computation of the signal quality (see 2). If the handover mechanism is based on the signal strength, these two parameters are not ignored.

x, y andz are used to specify the location of the antenna on map. If z is missing, it is considered by default to be zero. The units of these coordinates should be the same as the ones used for the map.

For directional antennas, some suplementary parameters are needed:

- direction defines the main direction of the radio signal propagation in sexagesimal degrees, measured from north, clockwise;
- tilt defines the tilt of the antenna given also in sexagesimal degrees;
- azim_dB_back and elev_dB_back defines the difference in the signal strength between the propagation direction and the opposite direction in the azimuthal and elevation planes in dB;
- beam_h specifies the horizontal beam width, which is the angle around the main direction of the antenna where the signal loss is less than or equal to 3 dB.
- beam_v specifies the vertical beam width, which defines the aperture in the vertical plane with a signal loss less than or equal to 3dB.

The parameters for directional antennas presented above are the same as the ones used in *mobloc* package. For a detailed overview on them an interested reader can consult Martijn Tennekes (2018). We emphsize the importance of specifying realistic values for these parameters in order to obtain a simulation closed to real life.

3.1.4. The population configuration file

The main characteristics of the synthetic population generated by the simulator are defined using a specific configuration file, also in XML format. In the example input files that we provide, the population configuration file is named population.xml in all datasets. Below we listed the content of the data\dataset1\population.xml file and we will use it to describe each input parameter.

```
<persons>
    <num_persons>50</num_persons>
    <min_age>10</min_age>
    <max_age>87</max_age>
    <age_distribution>
        <type>normal</type>
        <mean>42</mean>
        <sd>25</sd>
    </age_distribution>
    <male_share>0.48</male_share>
    <speed_walk>200</speed_walk>
    <speed_car>1000</speed_car>
    <percent_home>0.10</percent_home>
</persons>
```

The most important characteristic of the population is number of individuals used for simulation and it is given using the `num_persons` pair of tags. Each individual has some descriptive attributes: age, gender, speed of displacement. Although values for age distribution and gender are specified in this configuration file, they are not currently used but are intended for future developments, when the simulator will also generate CDRs.

The age of individuals is a random value drawn from a truncated normal distribution or a uniform distribution. In each case, the minimum age is determined by the value of the `min_age` parameter and the maximum value by the value of `max_age`. The type of the age distribution is specified inside the `age_distribution` pair of tags. They enclose the actual type given using `type` pair of tags. Two values can be used here: `normal` which means a truncated normal distribution or `uniform` which means the uniform distribution. If `normal` is used, two other parameters follow: `mean` - the mean value of the age and `sd` - the standard deviation of the age. For the uniform distribution there is no need to specify other parameters here, `min_age` and `max_age` being the limits of the age values. The gender of each individual is also a random value, the share of the males being determined by the value enclosed within the `male_share` tags. The actual value for the gender is drawn from a Bernoulli distribution.

During a simulation, the individuals can move with two speeds: one has a lower value, the other one has a larger value. They are introduced to simulate individuals moving by cars or other means of transport and individuals walking. These two values of the speed are defined by the `speed_walk` and `speed_car` parameters. The actual value of the speed of an individual is a random value drawn from a normal distribution with the mean given by the value of the two above mentioned parameters and the standard deviation 10% of the mean value.

The initial location of each individual at the begining of a simulation is generated randomly. To add a touch of reality to our simulation we force a share of the total population to have the same initial location on the map in every simulation. This share of population is defined using the last parameter in the configuration file, `percent_home`. The value given in this example 0.1 means that 10% of the entire population will have the same initial location in every simulation, the rest 90% having locations that differ from one simulation to another.

3.1.5. The probabilities configuration file

While the location probabilities of the mobile devices during the simulation are intended to be computed by another software, that will use the events generated by the network, following a complex methodology, we also provide a simple method to compute them in this simulation software. This computation is based on a Bayesian approach and follows the method described in Tennekes et al. (2019). Described shortly, the posterior location probability is given by $P(j|a) =\propto P(j)P(a|j)$, where $P(j|a)$ is the probability of a device to be located in tile j provided that the device is connected to antenna a, $P(j)$ is the prior location probability and $P(a|j)$ is the *event-location* probability (or the *location likelihood*) i.e. the probability of a device connected to antenna a to be in the center of tile j which can be computed based on the simulator software outputs. Currenlty, the simulation software supports two types of prior probabilities: uniform prior probability, $P(j) = \frac{1}{nTiles}$ for all tiles j in the grid and the network prior probability given by $P(j) = \frac{\sum_{a\in A} s(j,a)}{\sum_{a\in A}\sum_{j\in 1:nTiles} s(j,a)}$. Here A is the set of all antennas, $nTiles$ the total number of tiles in the reference grid, and $s(j,a)$ is the signal strength or signal quality received from antenna a in the center the tile j.

The computation of the posterior location probabilities is an optional feature of the simulation software and it is activated only if the software is run with `-o` command line option. An example configuration file, `data\dataset1\probabilities.xml`, included in our test data sets is listed below.

```
<probabilities>
    <prior>network</prior>
    <prob_file_name_prefix>probabilities</prob_file_name_prefix>
</probabilities>
```

There are only two parameters passed to the simulation software through this file. The first one is the prefix of the output file name where these probabilities are to be saved. This prefix is given using prob_file_name_prefix tags. The file name is then composed by concatenating this prefix with the name of the mobile network operator for which the computation is performed. We mention here that these probabilities are computed separately for each mobile network operator using the event data for that operator. The file is saved in .csv format.

The second parameter is the type of the prior probability used to compute the posterior location probabilities and is defined using prior tags. Only two values are supported for this parameter: uniform which means that an uniform prior location probability is used and network which means that the prior location probability is computed using the mobile network characteristics as described above.

3.2. Output files

During the simulation and after it is finished, the simulation software outputs a series of information splited into several files. All output files are in the .csv format and they are intended to be used in further processings, mainly to compute the posterior location probabilities for the mobile devices and to have the *real data* to compare the results of the statistical estimations with them. This is in fact one of the main strengths of the simulation software because in real life it is absolutely impossible to have real data about population counts in certain geographical areas, at certain time instants to compare them with the results of the inference methods that we propose. Without this comparison no one can decide if an inference model is better than other or not.

In the following subsections we describe the structure of each output file of the simulator.

3.2.1. The antenna file

Although the exact position of each antenna is given in an input configuration file (see subsection 3.1.3), the simulation software also writes an output file named by default antennas.csv in the curent directory where it is run, that contains location information for each antenna. The name of this file could be changed using <antennas_file> parameter in the general simulation configuration file (see subsection 3.1.2). One row in this file corresponds to an antenna, which means that the number of rows of this file should be equal to the number of antennas defined in the antennas configuration file. The content of file is generated at the initial time instant of the simulation. Below we list the first 6 lines of this file after a simulation was performed.

```
t,Antenna ID,x,y,MNO ID,Tile ID
0,2,0.000000,7500.000000,0,280
0,3,1000.000000,2500.000000,0,81
0,4,2000.000000,7500.000000,0,283
0,5,3000.000000,2500.000000,0,85
0,6,4000.000000,7500.000000,0,287
....
```

The first line is the header line, indicating the columns meanings:

- **t** - the time instant when the information on a row was generated. One can note that the value on the first column on each row of the file is 0. This is because we assume that atennas are fixed during the simulation, therefore the information recorded at the initial time instant will not change during the simulation and there is no reason to output this information at later time instants;
- **Antenna ID** - an internal ID of the antenna generated by the simulation software. Each row of the file contains information about one antenna identified by its ID. The values of these IDs are unique for a simulation. One cannot assume anything else about the IDs, excepting their uniqueness;
- **x** - the coordinate along OX of an antenna;
- **y** - the coordinate along OY of an antenna;

- **MNO ID** - the unique ID of the mobile network operator that owns this antenna. This ID is generated by the simulation software and it uniquely identifies a mobile network operator object during the simulation;
- **Tile ID** - the ID of the tile where the antenna is located. This ID is needed to uniquely identify each tile of the grid and it is in fact its index. The tiles' indexes of the grid are liniarized following a row-major order: it starts with 0 for the bottom left tile and goes up to $nTiles - 1$ for the upper-right tile. Figure3.1 depicts a grid with 25 tiles, showing the index (ID) of each tile;

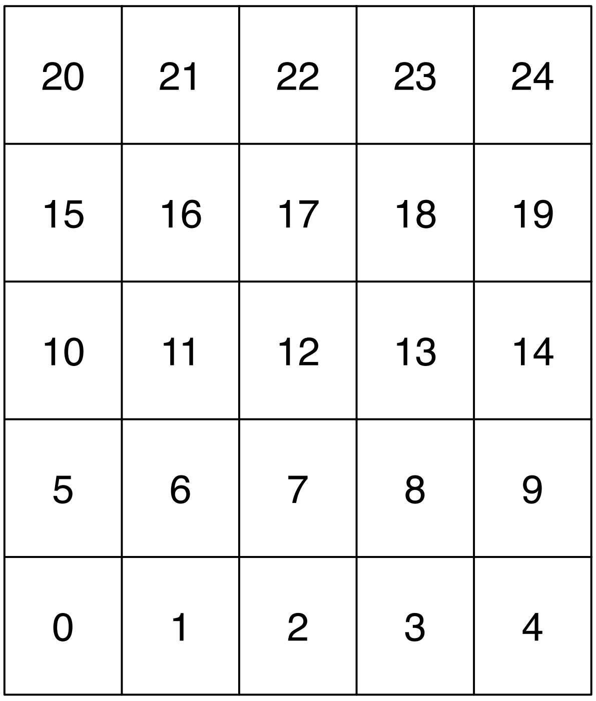


Figure 3.1: The tiles IDs for a grid with 25 tiles.

This file could be used for example by a visualization software to place the antennas in their exact position on the map.

3.2.2. The network event files

The network events are the most important data sets generated by the simulatior and they are stored in several files during their generation. Each antenna saves the events that it generates in a `.csv` file. The name of the file storing the events registered by an antenna is made up by concatenating the following strings: "Antenna", the internal antenna ID (the same ID found in `antennas.csv`) , "_MNO_", and the name of the mobile network operator as it was specified in the general simulation configuration file. Each antenna will use the name of the mobile network operator that owns it. For example, if we follow the configuration file given in subsection 3.1.3 in conjunction with the mobile network operator name specified for each antenna in antennas configuration file, the file names where the events are saved look like: `Antenna2_MNO_MNO1.csv`, `Antenna3_MNO_MNO1.csv`, ... `Antenna22_MNO_MNO1.csv`, etc.

To allow easy processing of the network event files, after the simulation is finished, all the files belonging to a mobile network operator are merged into a single large file. The name of this file is `AntennaInfo_MNO_` plus the name of the operator. Running a simulation with the test data files provided in `data/dataset1` will produce two event files: `AntennaInfo_MNO_MNO.csv` and `AntennaInfo_MNO_MNO2.csv`.

We list below the first and the last 6 lines from an event file and explain the meanings of each column. Each row of such a file contains an event recorded during the simulation.

```
t,Antenna ID,Event Code,Device ID,x,y, Tile ID
```

```
0,2,0,52,897.078045,5832.890081,221
0,4,0,90,1510.729201,6618.276457,263
0,5,0,47,3988.458704,2130.753835,87
0,6,0,60,4828.685926,8181.805414,329
0,8,0,79,6792.816925,7967.925558,313
...
199,18,2,69,2732.140804,9097.350172,365
199,3,2,38,957.345382,4255.426410,161
199,10,2,47,7803.511449,7231.533290,295
199,3,2,92,693.619802,4594.840112,181
199,6,2,60,4828.685926,8181.805414,329
199,2,0,90,184.837113,8135.300545,320
```

There are seven columns in this file:

- **t** - is the time instant when the event was generated;
- **Antenna ID** - is the internal ID of the antenna that recorded this event. It corresponds to IDs found in `antennas.csv`;
- **Event Code** - is an integer the represents the code of the network event. Currently, four types of events are generated and recorded:
  - 0 - a device connects to the antenna under consideration. This device generating the connection event was not connected to this antenna in the previous time instant - either it was connected to another antenna or it was out of the coverage area of any antenna;
  - 1 - a device disconnects from the antenna under consideration;
  - 2- a device connects to the antenna under consideration and it was also connected to the same antenna in the previous time instant;
  - 3 - a device is inside the coverage area of the antenna but the connection failed from different reasons (for example, the antenna reached its maximum capacity);
- **Device ID** - is the internal ID of the device that generated the event interacting with the antenna under consideration. This ID uniquely identifies each mobile device during a simulation and it is assigned to the device by the simulation software;
- **x** - is the exact coordinate on OX where the mobile device was located when it generated the current event;
- **y** - is the exact coordinate on OY where the mobile device was located when it generated the current event.;
- **Tile ID** - is the ID (index) of the tile where the point given by (x,y) is located on the grid. For a description of this ID and how the tiles are indexed see subsection 3.2.1.

We mention here two important things. We save the exact location of the mobile device when it generated the event in order to be able to compare the outputs of the inferential model that we will use to estimate the device count with the real number of mobile devices in a specific geographic area. The event codes 0 and 2 are equivalent from a connection/disconnection prespective. The only thing that differs is that in the previous time instant the mobile device was connected or not to the same antenna. We've decided to save this event with two distinct codes to ease the data processing from a temporal trajectory perspective. The same stands for values 1 and 3: they both mean a disconnection/failed connection event.

The final merged file with all the events generated during the simulation by all antennas could be very large, depending on the number of antennas, the number of mobile devices at the duration of the simulation. The size of this file can reach several gigabytes for large simulations. Such large files requires special processing technics common to big data technologies.

3.2.3. The persons file

This is an output file where information about all the individuals involved in a simulation are saved, regardless of whether they have and carry a mobile device or not. The default name of this file is `persons.csv` but it can be changed using `persons_file` parameter in the general simulation configuration file.

We list below the first 6 and last 6 lines from such a file generated during a simulation. While the other output files have a fixed structure this file has a different number of values on different rows: the number of values on a row could be 5, 6, or 7 depending on how many mobile devices an individual has (0, 1, or 2). Special care should be taken when this file is used in later processings to correctly read each row regardles of the number of values that it contains.

```
t,Person ID,x,y,Tile ID,Mobile Phone(s) ID
0,24,2328.508167,5762.076273,224
0,25,2381.660150,9317.205014,364
0,26,6224.073484,1656.513468,72
0,27,1735.187816,7482.781718,283
0,28,9625.113310,4374.041617,179
...
199,83,4497.675127,5387.393715,208
199,84,3353.085762,3771.063791,146
199,85,8943.705380,7715.859722,317
199,86,502.724584,2155.261842,81,87,88
199,89,184.837113,8135.300545,320,90
199,91,693.619802,4594.840112,181,92
```

The columns of this file are:

- **t** - is the time instant for which the information on a row was recorded. Values for **t** starts with initial time of the simulation and ends with the final time, increasing by the time increment value given in the general simulation configuration file;
- **Person ID** - is an internal ID which uniquely identifies each individual during a simulation. This ID is generated by the simulation software;
- **x** - is the exact coordinate on OX of the location where the individual was at time instant **t**;
- **y** - is the exact coordinate on OY of the location where the individual was at time instant **t**;
- **Tile ID** - is the ID (index) of the tile where (x,y) is located on the grid;
- **Mobile Phone(s) ID** - represents the internal IDs of the mobile devices an individual could own during a simulation. Because we allow individuals to have 0, 1, or 2 mobile devices, we could have no value on this column in the case that person doesn't have a mobile device or 1 or 2 values in case the person has 1 or 2 mobile devices. This specificity should be taken into consideration when the file is processed;

Using the information saved in this file, one can easily compute the exact number of mobile devices and persons within a specified geographical area at a time instant, an esential information used to choose the best estimation model. We emphasize that this information is not available in real life, but it is an essential information to be able to decide which is the most accurate model from a set of proposals.

3.2.4. The grid file

The full description of the grid overlapped on the map is saved in the `grid.csv` output file. This is the default name of the file, but it can be changed using the `grid_file` parameter from the general simulation configuration file. The coordinates of the origin of the grid and the number of tiles on OX and OY axes are computed by the simulation software, based on the dimensions of the tiles provided by the user and the bounding box of the map. We list below an example file:

```
Origin X,Origin Y,X Tile Dim,Y Tile Dim,No Tiles X,No Tiles Y
0.000000,0.000000,500.000000,500.000000,20,20
```

Origin X and **Origin Y** are the coordinates of the origin of the grid, i.e. the coordinates of the bottom left corner, **Tile Dim X** and **Tile Dim Y** are the dimensions of a tile on OX and OY axes and **No Tiles X** and **No Tiles Y** are the number of tiles along the OX and OY axes. **No Tiles X** and **No Tiles Y** are the minimum number of tiles on OX and OY needed to completly cover the map and they are computed by the simulation software. The information in this file is intended to be used by the visualization software and also by the statistical inference models.

3.2.5. The coverage area files

The coverage area of each antenna is computed during the building process of the antenna objects. It is defined as the geographical area where the signal strength or the signal quality is greater than a predefined minimum value. The type of the physical measure used to compute this area (strength or quality) is determined by the handover mechanism defined in the general simulation configuration file (see subsection 3.1.2) and the minimum value is selected accordingly from `Smin` or `Qmin` antenna parameters.

For omnidirectional antennas this area is a circle while for directional antennas it has a shape similar to an ellipse. In another chapter (see chapter 5) we provide a visual representation for the coverage areas of several antennas built using an R script. The name of the file containing the coverage areas is composed by concatenating `AntennaCells_` with the mobile network operator name. Below we show few lines from such a file.

```
AntennaId,Cell Coordinates
2,POLYGON ((2400.5 7500.0, 2395.5 7650.5, 2381.5 7801.0, 2358.0 7950.0,
2325.0 8097.0, 2283.0 8242.0,  2232.0 8383.5, 2172.0 8522.0, 2103.5 8656.5,
2026.5 8786.0, 1942.0 8911.0, 1849.5 9030.0, 1750.0 9143.0, 1643.0 9250.0,
1530.0 9349.5, 1411.0 9442.0, 1286.0 9526.5, 1156.5 9603.5, 1022.0 9672.0,
...

2395.5 7349.5, 2400.5 7500.0))

3,POLYGON ((3400.5 2500.0, 3395.5 2650.5, 3381.5 2801.0, 3358.0 2950.0,
3325.0 3097.0, 3283.0 3242.0, 3232.0 3383.5, 3172.0 3522.0, 3103.5 3656.5,
3026.5 3786.0, 2942.0 3911.0, 2849.5 4030.0, 2750.0 4143.0, 2643.0 4250.0,
2530.0 4349.5, 2411.0 4442.0, 2286.0 4526.5, 2156.5 4603.5, 2022.0 4672.0,
...

3358.0 2050.0, 3381.5 2199.0, 3395.5 2349.5, 3400.5 2500.0))
```

The coverage area is saved using the *WKT* format. The geometric object used to represent this area is a polygon which covers both cases: omnidirectional and directional antennas. For convenience, we erased some points from the polygon defining this area. Each row of the file contains the internal antenna ID and a *WKT* string that specifies the coverage area.

3.2.6. The signal strength/quality files

The signal strength or the signal quality is a key information used to compute the location probability of the mobile devices and it is saved in an `.csv` output file after the simulation ends. The strength or quality of the signal is computed in the center of each tile of the grid, thus, we save $nTiles$ values for each antenna. When we designed the structure of this file we took into consideration the size of the resulting dataset which is directly proportional to the number of antennas and the number of tiles. We decided to save these information using a file format that results in the smallest possible file, avoiding redunancy. Below we list a small part of such a file. It contains the value of the signal strength for a grid with 400 tiles. The name of the file is built by concatenating `SignalMeasure_` with the name of the mobile network operator. The numerical values in this file are determined by the handover mechanism: if this is based on the signal strength then the values represent the strength of the signal computed in the center of each tile, otherwise they are the values of the signal quality in the same points.

```
Antenna ID,Tile0,Tile1,Tile2,..., Tile397,Tile398,Tile399
22,-103.317,-102.673,-102.01,...,-91.3519,-91.0388,-90.8773
21,-106.209,-105.819,-105.434,...,-67.408,-67.408,-70.2426
20,-104.683,-104.321,-103.97,...,-77.4238,-80.4711,-83.1097
...
2,-97.1963,-97.2678,-97.4089,...,-100.529,-101.326,-102.086
```

The **Antenna ID** is the internal ID of the antenna and the rest of the columns represents the IDs of the tiles of the grid: **Tile0** means the tile with ID = 0, **Tile1** the tile with ID = 1 and so on. The tile indexing mechanism follows the same convention described in subsection 3.2.1: Tile0 is the tile from the bottom left corner while Tile 399 is the tile from the upper right corner.

3.2.7. The probabilities files

This file contains the location probabilities for each mobile device, at each time instant of the simulation, computed in the center of each tile of the grid. This is an optional file, generated only if the user runs the simulation software with `-o` command line parameter. We made it optional because the simulation software only implements a simple method to compute the location probabilities of the mobile devices. It is intended to develop a specific methodology for this purpose which will implement more complex algorithms. The default name of the file is made up concatenating `probabilities_` with the name of the mobile network operator. The prefix of the file name can be changed using the `prob_file_name_prefix` parameter from the probabilities configuration file. The location probabilties are computed separately, for each mobile network operator, i.e. we will have as many files as the number of mobile network operators involved in a simulation (one or two).

Below we list an excerpt from this file.

```
t,Phone ID,Tile0,Tile1,Tile2,...,Tile397,Tile398,Tile399
0,74,0.00049590,0.00054434,0.00059678,...,0.00411431,0.00402433,0.00388522
0,67,0.00319929,0.00341594,0.00359041,...,0.00065885,0.00060353,0.00055150
0,59,0.00049590,0.00054434,0.00059678,...,0.00411431,0.00402433,0.00388522
...
199,59,0.00054349,0.00060950,0.00068477,...,0.00293726,0.00301839,0.00305995
199,50,0.00062803,0.00068343,0.00074079,...,0.00304963,0.00281111,0.00255230
199,38,0.00199288,0.00224990,0.00251169,...,0.00084366,0.00078644,0.00072901
199,34,0.00069465,0.00071789,0.00073739,...,0.00246280,0.00212505,0.00182935
199,32,0.00119860,0.00123544,0.00126097,...,0.00132840,0.00115201,0.00099991
```

Each of these files (in case of several mobile network operators) contains the follwing information:

- **t** - is the time instant when the location probabilities are computed;
- **Phone ID** - is the internal ID of the mobile device for which the location probabilities are computed;
- **Tile0**, **Tile1**, ... **Tile399** - are the location probabilities in the center of the tiles identified by their IDs (indexes);

In this example we used a grid with 400 tiles and the simulation has 200 time steps. The indexing method of the tiles was described in subsection 3.2.1.

4

How to build and run the simulator

Being distributed at the level of source code, in order to build the simulation application, a C++ compiler compliant with C++17 standard (ISO/IEC, 2017) is needed. During the development process we used the GNU C++ compiler as well as the LLVM C++ compiler. However, the source code is written in a portable manner, using only standard features of the C++ language providing thus a high level of portability. Any other C++ compiler compliant with the C++17 standard can be used to build the application.

In the following sections we describe how to setup an environment that allows one to compile the application, we describe the steps needed to effectively build the application and we show how this software can be run. We provide detailed instructions for the main operating systems on the software market nowadays: Windows, Linux and MacOS. We emphasize again that we used only open source tools (compilers, IDEs and other software tools).

4.1. Build the application under the Windows operating system

Building the simulation application on Unix-like systems (Linux, MacOS, etc.) is straightforward but on Windows this is not a very easy task. That's why we provide here detailed information about the necessary steps to follow when a user wants to build the application on Windows. We don't target a specific version of the Windows operating system, the software was tested and works on Windows 7 and all other newer versions.

4.1.1. Prerequisites

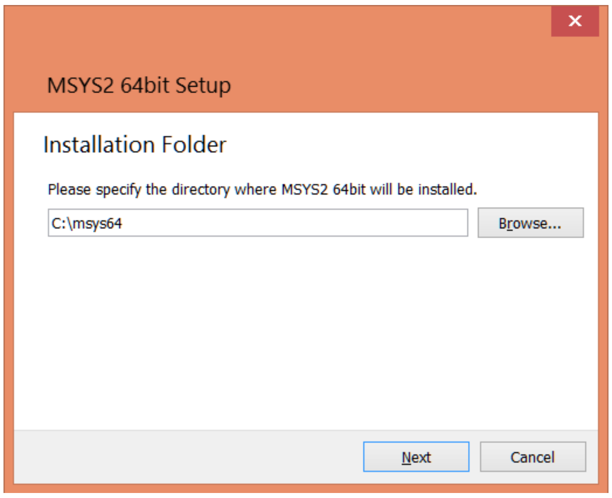
4.1.1.1. MSYS2 environment

Since the simulation software is distributed at the level of source code, the user must have a software development environment that allows him/her to compile and build the application. Because Windows does not provide by default such an environment, the first step before building the application is to setup a development environment. There are several such environments, some of them being very complex (like *Cygwin*) others being out of current maintenance (like *msys*). We chose to work with *MSYS2* environment which is a software distribution and building platform for Windows and it can be freely downloaded from `https://www.msys2.org/`.

MSYS2 provides a limited POSIX compatibility layer which makes it a Unix-like programming environment on top of the Windows. Together with the compiler provided by *MinGW-w64* package, it ensures a good interoperability with native Windows software. Among other utilities it provides a *bash* shell, *autotools*, *revision control systems* and combined with the *MinGW-w64* toolchains allows users to build native Windows applications. It also has a package management system `pacman` which provides easy installation and maintenance of aditional software packages.

Installing *MSYS2* is straightforward and detailed instructions are provided on the project's web page `https://github.com/msys2/msys2/wiki/MSYS2-installation`. We give here only a brief summary of installing instructions. The first step is to download the *MSYS2* installer. There are 32 and 64 bit versions of the installer software but we recommend using the 64 bit version. The link to download it is: `http://repo.msys2.org/distrib/x86_64/msys2-x86_64-20190524.exe`. Please note that MSYS2 can be installed on Windows 7 or newer versions and it requires an *NTFS* partition, it cannot be installed on *FAT** partitions.

After downloading the *MSYS2* installer, run it and select the installation directory, for example `C:\msys64` (see Figure4.1).



MSYS2 64bit Setup

Installation Folder

Please specify the directory where MSYS2 64bit will be installed.

C:\msys64

Figure 4.1: Installing the *MSYS2* environment

Once *MSYS2* is installed, one should use the `pacman` package manager to update all the software packages installed by default. First, start an *MSYS2* shell by selecting the *Start* button, then select *MSYS2 64 bit* and then *MSYS2 MSYS* to start the shell (see Figure4.2).

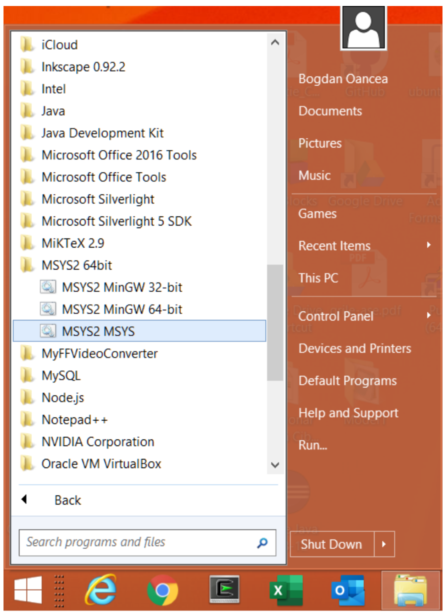
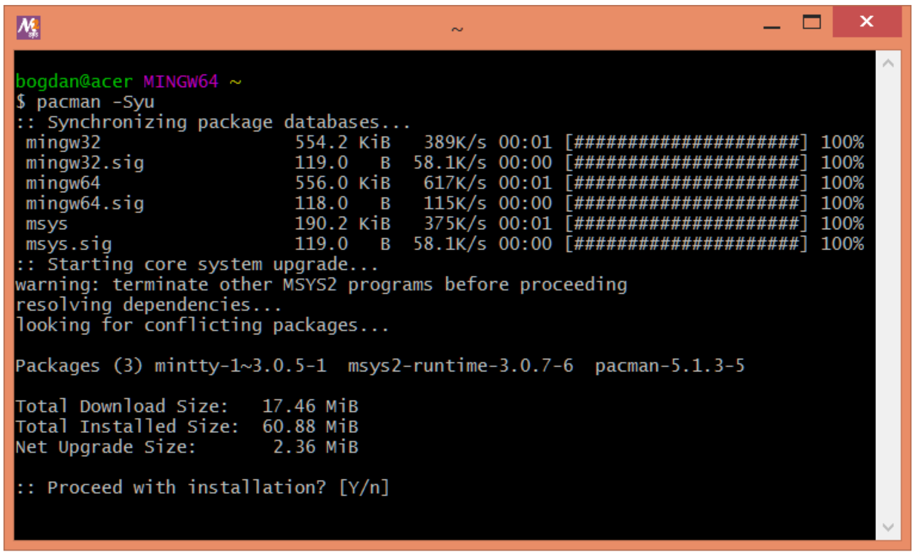


Figure 4.2: Updating the MSYS2 environment

In the shell window run the following command (see Figure4.3):

```
$pacman -Syu
```

Answer with `Y` to the request to proceed with the installation.



```
bogdan@acer MINGW64 ~
$ pacman -Syu
:: Synchronizing package databases...
 mingw32                 554.2 KiB   389K/s 00:01 [#####################] 100%
 mingw32.sig             119.0   B  58.1K/s 00:00 [#####################] 100%
 mingw64                 556.0 KiB   617K/s 00:01 [#####################] 100%
 mingw64.sig             118.0   B   115K/s 00:00 [#####################] 100%
 msys                    190.2 KiB   375K/s 00:01 [#####################] 100%
 msys.sig                119.0   B  58.1K/s 00:00 [#####################] 100%
:: Starting core system upgrade...
warning: terminate other MSYS2 programs before proceeding
resolving dependencies...
looking for conflicting packages...

Packages (3) mintty-1~3.0.5-1  msys2-runtime-3.0.7-6  pacman-5.1.3-5

Total Download Size:   17.46 MiB
Total Installed Size:  60.88 MiB
Net Upgrade Size:       2.36 MiB

:: Proceed with installation? [Y/n]
```

Figure 4.3: Updating the MSYS2 environment

4.1.1.2. GNU C++ compiler on Windows with MSYS2

The next step is to install a C++ toolchain to build C++ applications. This consist in a C++ compiler and the associated utilities. While there are a variety of toolchains to build C++ applications on Windows operating system and the most natural choice seems to be the one provided by Visual Studio IDE, we wanted to avoid proprietary software and use only free, open source software. This was the main reason why we chose the GNU C++ compiler. Besides being open source, it is also available on the most used platforms: Windows, Linux, MacOS, etc. The Windows port of the GNU C/C++ compiler is distributed in a package called *Mingw* - Minimalist GNU for Windows. There are open source IDEs that contain a *Mingw* compiler like *CodeBocks* or *Qt Creator*, but in this report we will show how to build the simulator application from the command line.

After updating all packages, use `pacman` again to install the development toolchain and the C/C++ compiler. Run the command below in an *MSYS2* shell:

```
$pacman -S base-devel gcc
```

When `pacman` ask what base-devel applications should be installed, answer with Enter to select all and answer with Y to the next question to install the required dependencies.

Then, update the user or system `PATH` environment variable and add the path to the C++ compiler and `make` utility. In the following we assume the default location for *MSYS2* 64 bit version which is `C:\msys64` folder. One can update the `PATH` variable using the standard *Control Panel* application provided by Windows. In *Control Panel* select *System*, then *Advanced System Settings*. A dialog box like the one shown in Figure4.4 should appear on the screen.

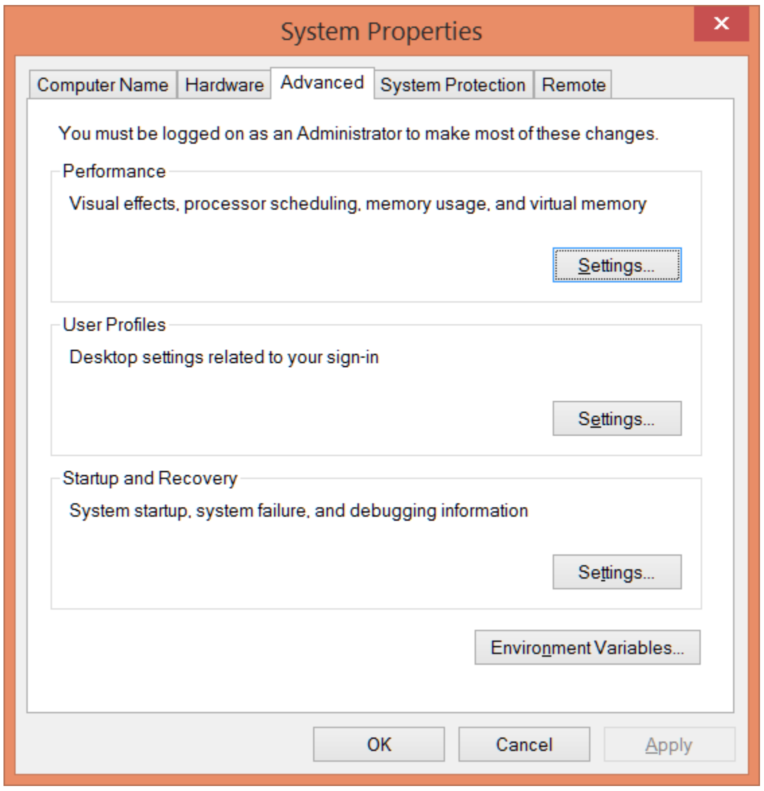
Next, press *Environment variables* button. In the dialog box shown in Figure4.5 select the *Path* variable either from *User Variables* list or from the *System Variables* list and then press the *Edit* button.

In the *Edit User/System Variable* text box add a semicolon at the end of the already existing text and then add the path to the C++ compiler, for example `C:\msys64\mingw64\bin`.

Alternatively, the system `PATH` variable could be updated using the `CMD` utility program that all Windows versions has by entering the following command:

```
>setx path C:\msys64\mingw64\bin;C:\msys64\usr\bin;"%path%
```

Instead of using the standard `CMD` program to update the `PATH` variable, one can use the shell provided by *Mingw MSYS2* by running `C:\msys64\mingw64.exe` application which has all the required environment variables already set.



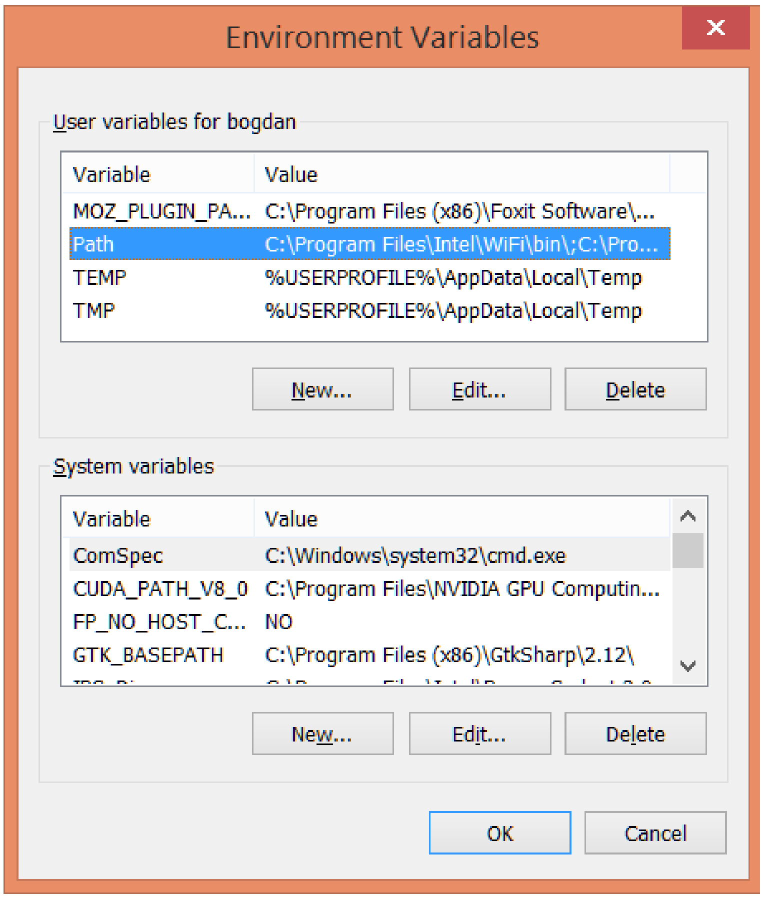
You must be logged on as an Administrator to make most of these changes.

Performance
Visual effects, processor scheduling, memory usage, and virtual memory

User Profiles
Desktop settings related to your sign-in

Startup and Recovery
System startup, system failure, and debugging information

Figure 4.4: Control Panel - Advanced Settings dialog box



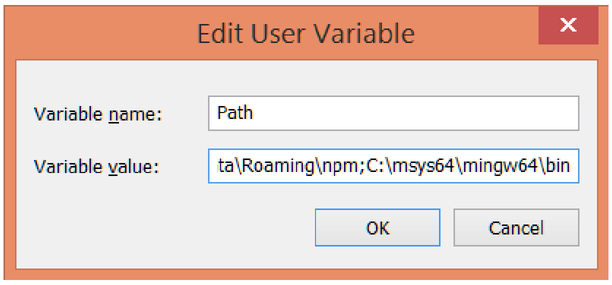
User variables for bogdan

Variable	Value
MOZ_PLUGIN_PA...	C:\Program Files (x86)\Foxit Software\...
Path	C:\Program Files\Intel\WiFi\bin\;C:\Pro...
TEMP	%USERPROFILE%\AppData\Local\Temp
TMP	%USERPROFILE%\AppData\Local\Temp

System variables

Variable	Value
ComSpec	C:\Windows\system32\cmd.exe
CUDA_PATH_V8_0	C:\Program Files\NVIDIA GPU Computin...
FP_NO_HOST_C...	NO
GTK_BASEPATH	C:\Program Files (x86)\GtkSharp\2.12\

Figure 4.5: Selecting the variable



Variable name: Path
Variable value: ta\Roaming\npm;C:\msys64\mingw64\bin

Figure 4.6: Updating the PATH variable

4.1.1.3. GEOS C++ library

The simulation application uses a GIS library - *GEOS*. Before being able to build the simulator, the *GEOS* library should be downloaded, compiled and installed.

GEOS is a C++ port of *JTS* - Java Topology Suite, an open-source library that provides an implementation of the Euclidean planar linear geometry operations and is intended to be used as a component of vector-based geographical information systems. The C++ port *GEOS* contains only a subset of functions from the original *JTS* but they are sufficient for the simulator needs.

The current version of the simulator was developed using *GEOS* ver. 3.7.1 but we also tested it against *GEOS* ver. 3.7.2. The source code of the library can be downloaded from `https://trac.osgeo.org/geos`. Detailed instructions on how to build the *GEOS* C++ library are provided at `https://trac.osgeo.org/geos/wiki/BuildingOnUnixWithAutotools`. In the following, we shortly describe this process. After unpacking the archive with the source code of the library, for example in `C:\data-simulator\geos-3.7.1` directory, one has to open a shell by running `C:\msys64\mingw64.exe`, change the directory to the source code location and then type the following commands:

```
$ ./configure
$ make
$ make check
$ make install
```

Compilation of the *GEOS* library could take several minutes depending on the computer performance. The `make check` command is optional, it runs some self-tests that comes with the source code distribution. If the user wants to remove the object files created during the compilation he/she can type the following command:

```
$ make clean
```

More, the files created by `./configure` can be also deleted with the command:

```
$ make distclean
```

By default, the libraries produced after compilation are installed in the `C:\msys64\mingw64\lib` directory. This can be checked with the following commads:

```
$ cd C:\\msys64\\mingw64\\lib
$ ls *geos*
libgeos.a       libgeos.la   libgeos_c.dll.a
libgeos.dll.a   libgeos_c.a  libgeos_c.la
```

Please note the double backslash used to separate the components of the path. It is necessary because *MSYS2* shell follows the UNIX convention and interprets a single backslash as the begining of an escape character.

If the output is different from the one above this means that something went wrong during the compilation process. The library can also be installed in another directory, by running `./configure` with the `--prefix=PATH` option.

4.1.1.4. The git version system

Git is one of the most popular free and open source version control systems, desgined to keep track of the changes in the source code during development. It that can be used for any type of project, from very simple to very complex software systems. It was created in 2005 by Linus Torvalds and since then it has become one of the most used tools for developing and sharing software projects. We used *git* together with the `https://github.com` site that provides free hosting for software development version control. The simulator software can be found at the following address: `https://github.com/MobilePhoneESSnetBigData/simulator`. To install, compile and run the simulator the user should first get the source code from the above mentioned address. For this purpose he/she needs to install the `git` tool which can be found at: `https://git-scm.com/download/win`. The installation process is very simple: after the installation kit is downloaded on the local computer, simply run the installer. It will install two shells and one GUI application. They are found under the *Git* group in the *Start* menu (see Figure4.7).

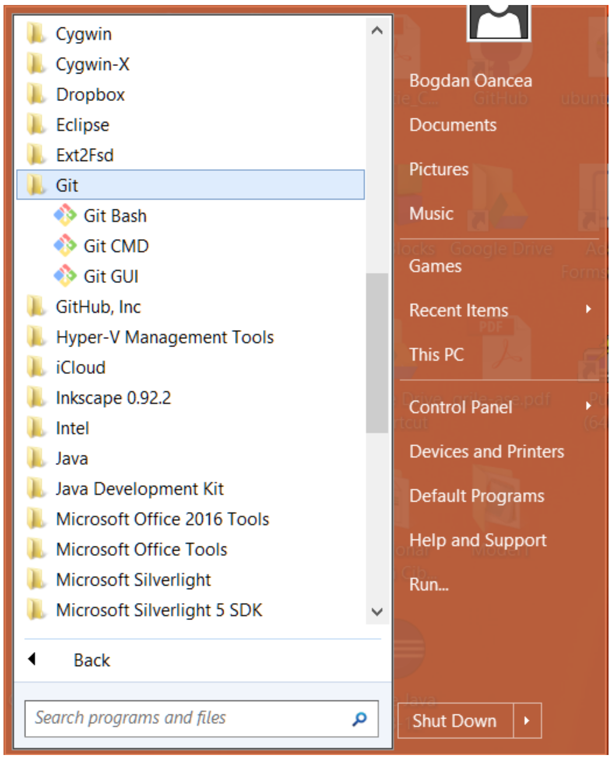


Figure 4.7: The git tool

4.1.1.5. Build and run the simulation software

Assuming that the user wants to download the source code of the simulator in `C:\data-simulator` directory, he/she should open this directory in *Windows Explorer* application, perform a right click with the mouse and then select *Git Bash Here* from the contextual menu that appear on the screen (see Figure4.8).

Then, the user should type the following command in the *Bash* shell:

```
$git clone https://github.com/MobilePhoneESSnetBigData/simulator.git
```

After running this command a new directory named `simulator` can be found in the current directory and it contains the source code of the simulation software. One should change the current directory to

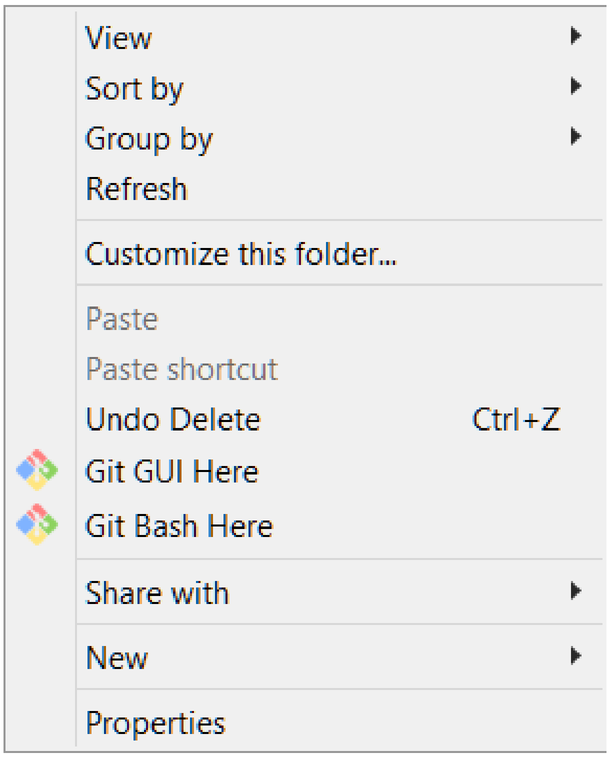


Figure 4.8: Run the git bash shell

the `simulator`, edit the `makefile.inc` file found here with any text editor and change the values of the following 3 variables:

```
PROJ_HOME
GEOS_HOME
MSYS_HOME
```

`PROJ_HOME` should point to the directory where the source code of the simulator was downloaded (for example `C:/data-simulator/simulator`), `GEOS_HOME` should point to the directory where the *GEOS* C++ library was installed and `MSYS_HOME` should indicate the directory where the *MSYS2* development environment was installed. If the default directories given in this installation instructions were used, the three variables should look like:

```
PROJ_HOME = C:/data-simulator/simulator
GEOS_HOME = C:/msys64/mingw64/lib
MSYS_HOME = C:/msys64/
```

After editing these values, save the file, open an *MSYS2* shell (by running `C:/msys64/mingw64.exe`), change the current directory to the one where the simulator source code was downloaded and type:

```
$ make
$ make install
```

The executable file is copied under the Release directory. To run a simulation type the following:

```
 $Release/simulator.exe -m ./data/dataset1/map.wkt -s ./data/dataset1/simulation.xml
-a ./data/dataset1/antennas.xml -p ./data/dataset1/persons.xml
-pb ./data/dataset1/probabilities.xml -v -o
```

where the following files provide input parameters:

- `map.wkt` is the map where the simulation takes place; it is specified using `-m` option;
- `simulation.xml` is a configuration file containing the general parameters of a simulation; it is specified using `-s` option;
- `antennas.xml` is the antennas configuration file and it contains the technical parameters of each antenna together with the locations of all antennas on the map; it is specified using `-a` option;
- `persons.xml` is the population configuration file containig the characteristics of the synthetic population used for a simulation; it is specified using `-p` option;
- `probabilities.xml` is the file containing the parameters needed to compute the posterior location probabilities for mobile devices; it is specified using `-pb` option; the probabilities are computed only if the `-o` option is present in the command line;

If `-v` is given in the command line, the simulator will output on console the set of persons, mobile phone operators, antennas and mobile phones.

We provide sample input configuration files under the `data` directory located in the root directory of the simulator source code. Some of them are artificially generated others use real maps and antenna information, freely available on Internet (`https://www.openstreetmap.org`, `https://www.opencellid.org`).

4.2. Build the application under the Linux operating system

Building the simulation software under Linux is similar. If the Linux distribution used does not have a C++ compiler and the related tools needed to build an application from source already installed, the first step is to download and install them. We provide here installation instruction for Ubuntu version of Linux. Simply, the user should open a terminal and type the following commands:

```
$ sudo apt update
$ sudo apt upgrade
$ sudo apt install build-essential
```

Instead, one can run also:

```
$ sudo apt-get update
$ sudo apt-get upgrade
$ sudo apt-get install build-essential
```

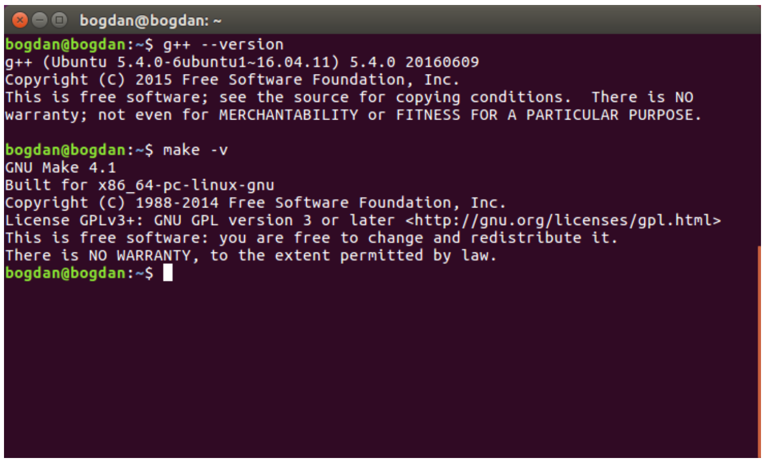
The `sudo` command will require the user to enter the his/her password. The sequence of instructions install a set of packages which are essential for building applications on Ubuntu including *gcc/g++* compiler, *make* and other required utilities. To verify that the installation went OK, one should check it with:

```
$ g++ --version
$ make -v
```

A sample output of these two commands are provided in Figure4.9.

Downloading and building the *GEOS* C++ library follows the same steps described in section 4.1.1.3. After the download was completed, change the current directory to the root of the source code tree, open a terminal and type:

```
$ ./configure
$ make
$ make check
$ make install
```



```
bogdan@bogdan:~$ g++ --version
g++ (Ubuntu 5.4.0-6ubuntu1~16.04.11) 5.4.0 20160609
Copyright (C) 2015 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

bogdan@bogdan:~$ make -v
GNU Make 4.1
Built for x86_64-pc-linux-gnu
Copyright (C) 1988-2014 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
bogdan@bogdan:~$
```

Figure 4.9: Checking the GNU compiler installation on Ubuntu

If `git` version control system is not already installed, in the same opened terminal type:

```
sudo apt update
sudo apt install git
```

Checking the installation can be done with the following command:

```
git --version
```

The output should be like (the version of *git* may differ):

```
git version 2.17.1
```

The next step is to download the source code of the simulation software:

```
$git clone https://github.com/MobilePhoneESSnetBigData/simulator.git
```

Then, the user should open `makefile.inc` with any text editor at his/her dissposal and change the values of the following 2 variables:

```
PROJ_HOME
GEOS_HOME
```

On Linux the `MSYS_HOME` variable is not used and it can ignored.

`PROJ_HOME` should point to the folder where the user downloaded the source code of the simulator and `GEOS_HOME` should point to the folder where GEOS library was installed, usually `\usr\local\lib`. After changing these values, check that the current working directory is the directory where the simulator source code is located and type the following commands in a terminal:

```
$ make
$ make install
```

The executable file is installed in the `Release` directory which was created in the root directory of the source code. To run a simulation, type the following command in a terminal:

```
$./Release/simulator -m ./data/dataset1/map.wkt -s ./data/dataset1/simulation.xml
-a ./data/dataset1/antennas.xml -p ./data/dataset1/persons.xml
-pb ./data/dataset1/probabilities.xml -v -o
```

The significance of each command line option is the same as presented in section 4.1.1.5.

4.3. Build the application under the MacOS

Building the simulation software under MacOS is similar to Linux. If the MacOS does not already have a C++ compiler and the related tools needed to build an application from source installed, the first step is to download and install them. There are several choices of free C++ compilers and related tools available on MacOS and we will shortly present here two of them.

The first solution is to install *Xcode*, the development environment provided by Apple to write applications for MacOS. Among other tools it contains the LLVM C++ compiler and an IDE. To install *Xcode* one should open the *App store* application, go to *Develop* section, search for *Xcode*, press the *Get* button and follow the steps required by the automatic installation procedure. When this procedure ends, the user could open a *Terminal* and check if all went OK by typing:

```
$xcode-select -p
```

If the following message appear on the screen this means that *Xcode* was properly installed.

```
/Applications/Xcode.app/Contents/Developer
```

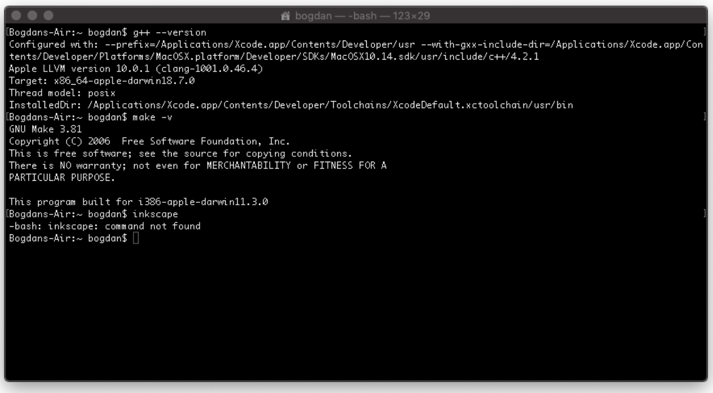
Next, the user should install the Xcode command line tools by typing:

```
$ xcode-select --install
```

in the *Terminal* window. Finally, check again that the compiler and the command line tools were installed by typing:

```
$g++ --version
$make -v
```

A result similar to the one in Figure4.10 shoulld appear in the *Terminal* window.



```
Bogdans-Air:~ bogdan$ g++ --version
Configured with: --prefix=/Applications/Xcode.app/Contents/Developer/usr --with-gxx-include-dir=/Applications/Xcode.app/Con
tents/Developer/Platforms/MacOSX.platform/Developer/SDKs/MacOSX10.14.sdk/usr/include/c++/4.2.1
Apple LLVM version 10.0.1 (clang-1001.0.46.4)
Target: x86_64-apple-darwin18.7.0
Thread model: posix
InstalledDir: /Applications/Xcode.app/Contents/Developer/Toolchains/XcodeDefault.xctoolchain/usr/bin
Bogdans-Air:~ bogdan$ make -v
GNU Make 3.81
Copyright (C) 2006  Free Software Foundation, Inc.
This is free software; see the source for copying conditions.
There is NO warranty; not even for MERCHANTABILITY or FITNESS FOR A
PARTICULAR PURPOSE.

This program built for i386-apple-darwin11.3.0
Bogdans-Air:~ bogdan$ inkscape
-bash: inkscape: command not found
Bogdans-Air:~ bogdan$
```

Figure 4.10: Checking the C++ compiler and command line tools installation on MacOS

A second option is to install the GNU C++ compiler but this is not provided by the official *App store*. Instead, the user should first install the *homebrew* package manager typing the following command in a *Terminal* window:

```
$/usr/bin/ruby -e
"$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install)"
```

Then, enter the following two commands:

```
$brew install gcc
$brew install make
```

in the same *Terminal* window to install the GNU C++ compiler and `make` tool.

If *Xcode* command line tools were installed, they already provide `git` version control system. Otherwise, `git` can be installed using:

```
$brew install git
```

Downloading and building the *GEOS* C++ library follows the same steps described for Windows and Linux operating systems. The source code of the simulation software can be downloaded as explained in the previous sections, first by choosing a directory where to download it and then by typing:

```
$git clone https://github.com/MobilePhoneESSnetBigData/simulator.git
```

command in a *Terminal* window.

As for Linux, the user should open `makefile.inc` file with any text editor and change the values of the following 2 variables:

```
PROJ_HOME
GEOS_HOME
```

`PROJ_HOME` should point to the directory where the user downloaded the source code of the simulator and `GEOS_HOME` should point to the folder where *GEOS* library was installed, usually `\usr\local\lib`. In a *Terminal* window change the current directory to the one where the simulator source code is located and type the following commands:

```
$ make
$ make install
```

The executable can be found now in `Release` subdirectory of the current directory.

5

Examples of usage

In this chapter we provide an example how to run the simulator using one set of the predefined configuration files, distributed together with the source code and how to understand the results/outputs. While looking at the bare numbers saved in the output files is not very attractive, we used some custom made R scripts to visualize the results of a simulation in order to explain and understand better the output data meaning.

The following examples assumes that the user works under the Windows operating system. After building the executable application (see section 4.1.1.1 for a description on how build the executable application), open an *MSYS2* shell, go to the root folder of the source code distribution and run a simulation with the following command:

```
$Release/simulator.exe -m ./data/dataset1/map.wkt -s ./data/dataset1/simulation.xml
-a ./data/dataset1/antennas.xml -p ./data/dataset1/persons.xml
-pb ./data/dataset1/probabilities.xml -v -o
```

It uses the input files provided in `data/dataset1` directory of the source code distribution. This dataset has an artificial generated map bounded by a 10 km x 10 km box and we overlapped a grid of square tiles with the dimensions 500 m x 500 m, which means that we have a total number of 400 tiles overlapped on the map. The data set has 23 antennas belonging to 2 mobile network operators, both with a 35% market share. 22 antenas are omnidirectional while one antenna is directional. The simulation involves 50 persons, 15% of which have 2 mobile phones. The individuals move around the map using a random walk with drift model of displacement, and the handover mechanism is based on the signal strength. The content of the configuration files as well as the map file are listed in appendix A for convenience. We emphasize that the simulation software use a random number generator to set the initial position of the individuals (with the exception of a small share whose initial positions are always the same provided that the random seed is the same) and other parameters of the displacement like initial speed, the direction angle etc. and therefore if one tries to run a simulation with the same input files the results may be different.

The map used in this example together with the antennas placed on it and the overlapped grid are shown in Figure5.1. The map is a simple polygon and it was created with a text editor. Each antenna has an internal ID which is displayed near to them. These IDs are genetated internally by the simulator, they cannot not be specified in the configuration file. Having the IDs generated automatically by the simulator, we are sure that they are unique during the simulation.

After the simulation is finished, the following output files could be found in the current directory:

- `persons.csv`
- `antennas.csv`
- `grid.csv`
- `Antenna1_MNO_MNO1.csv`...`Antenna22_MNO_MNO1.csv` (one file per antenna) and `Antenna23_MNO_MNO2`
- `AntennaInfo_MNO_MNO1.csv`

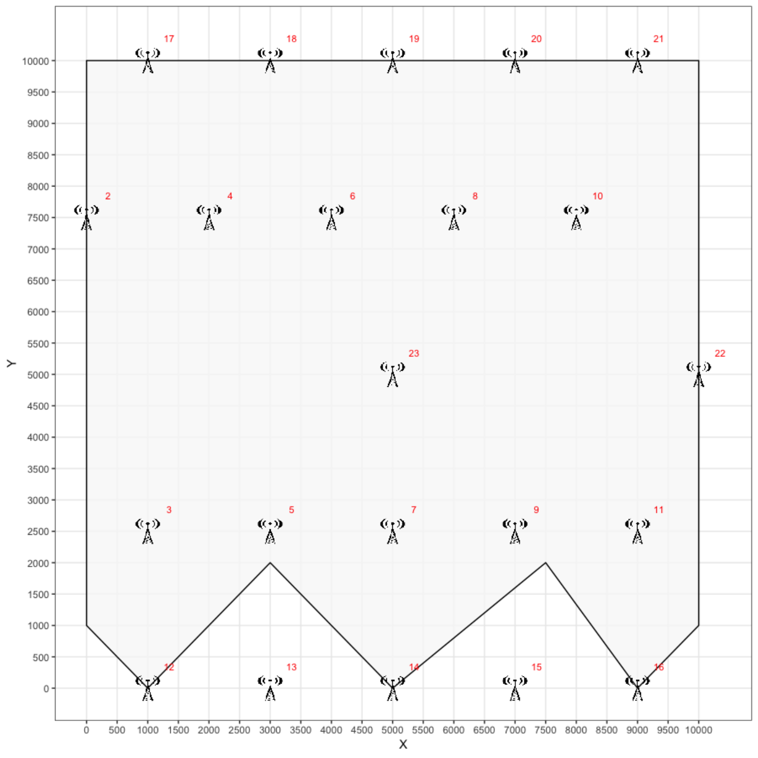


Figure 5.1: An example map and a set of antennas

- `AntennaInfo_MNO_MNO2.csv`
- `SignalMeasure_MNO1.csv`
- `SignalMeasure_MNO2.csv`
- `AntennaCells_MNO_MNO1.csv`
- `AntennaCells_MNO_MNO2.csv`
- `probabilities_MNO1.csv`
- `probabilities_MNO2.csv`

The content of each output file was described in detail elsewhere in this document (see chapter 3). Before stating the actual simulation, the software computes the coverage area for all antennas, i.e. the area where an antenna provides a radio signal good enough to be usable by mobile devices. Outside of this coverage area, a mobile device cannot connect to the antenna under consideration. The coverage areas are defined as polygons and they are saved in `AntennaCells_MNO_MNO1.csv` and `AntennaCells_MNO_MNO2.csv` files written in *WKT* format, together with the antennasIDs. These two files can b inspecteth any text editor.

In Figure5.2 we show the same map with the internal antenna ID displayed near each antenna symbol and we selected antennas with IDs 2, 6, 7, and 22 to show their coverage areas. All these selected antennas are omnidirectional , and their coverage areas are circles as expected. One can note that some coverage areas overlap: in the set of the four selected antennas,the coverage area for antenna 2 overlap the area for antenna 4. This is in fact a case often found in real life. We chose only a limited set of antennas to show their coverage areas to limit the overlapping between different coverages areas that would have been made the figure difficult to understand.

Figure5.3 presents the coverage area for antenna 23 which is a directional antenna and it is oriented at 135 degrees from North. The direction angle is read from the `antenna.xml` configuration file and it is easy to see that the direction of signal propagation and the shape of the coverage area is different from the ones of an omnidirectional antenna. The arrow in Figure5.3 was specialy added to highlight the directional character of signal propagation. The maximum distance and the area covered by the radio signal can be controlled by the *tilt* parameter.

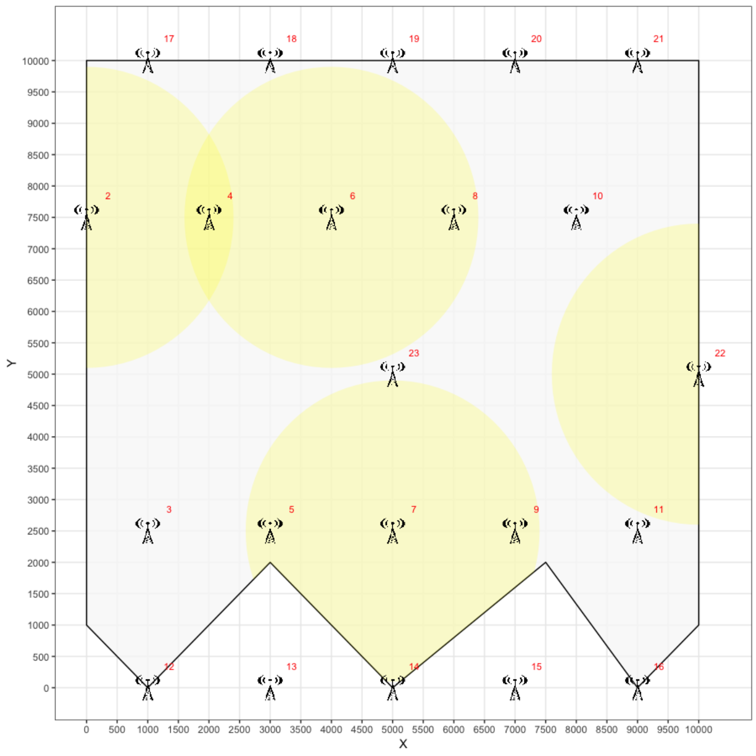


Figure 5.2: Coverage areas for a set of antennas

In this simulation the handover mechanism is based on the signal strength and its values are computed after the actual simulation ends in the center of each tile of the grid and saved in `SignalMeasure_MNO1.csv` and `SignalMeasure_MNO2.csv` files. We built two visual representations for the data in these files, one for omnidirectional antennas with IDs 2, 6, 7, 22 presented in Figure5.4 and another one for directional antenna with ID 23 presented in Figure5.5. Analysing these figures it can be observed how the signal strength decrease with the distance from antenna location and the patterns of the signal strengt or directional and omnidirectional antennas. Another observation that can be made analyzing from Figure5.4 is the overlapping between the radio signals of antenna 2 and 6, the same situation that we presented in Figure5.2 the coverage areas.

Based on the signal strength values we can compute the so-called *event-location* probabilities or *location likelihood* which gives us the observed probability of a device located in tile g to generate a connection event to antenna a. The *location likelihood* is defined as the ratio of the received signal strength from an antenna in the center of a tile and the sum of the received signal strength in the same point from all antennas of the same mobile network operator. We used the following formula to compute the *event-location* probability Tennekes et al. (2019):

$$P(a|j) = \frac{s(j,a)}{\sum_{a \in A} s(j,a)} \tag{5.1}$$

where $s(j|a)$ is the signal strength/signal quality received from antenna a in tile j and A is the set of all antennas. We computed this probability for each antenna and each tile of the grid using the values recorded in `SignalMeasure_MNO1.csv` that contains the signal strength in the center of each tile for each antenna for a simulation and built a visual representation of it presented in Figure5.6. It can be noticed how this probability decreases with the distance from the antenna location.

If the handover mechanism is set to use the signal quality, the same computations are performed only replacing $s(j,a)$ with the values of the signal quality which are saved in the same `SignalMeasure_MNO1.csv` file.

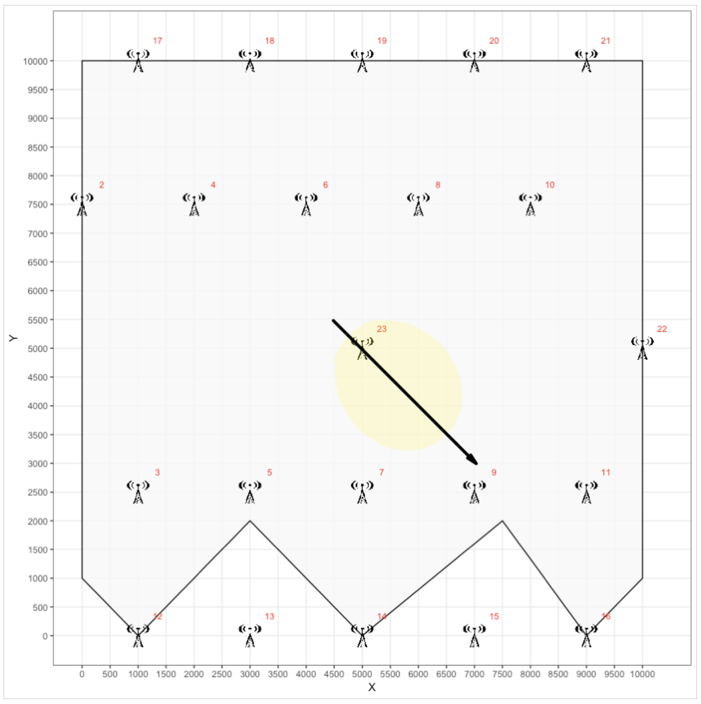


Figure 5.3: Coverage area for a directional antenna

During the simulation, the individuals in the synthetic population start to move according to a predefined pattern and their mobile devices will generate connection/disconnection events which are saved by each antenna. Using the *event-location* probability defined in equation (5.1) the simulator computes and outputs the posterior location probability using a Bayesian approach:

$$P(j|a) =\propto P(j)P(a|j) \tag{5.2}$$

where $P(j|a)$ is the posterior location probability of a device to be located in tile j provided that the device is connected to antenna a, $P(j)$ is the prior location probability at our disposal and $P(a|j)$ is the *event-location* probability (or the *location likelihood*). There are two types of prior probabilities implemented in the simulation software: uniform prior probability, i.e. the $P(j) = \frac{1}{nTiles}$ for all tiles j in the grid and the network prior probability given by $P(j) = \frac{\sum_{a\in A} s(j,a)}{\sum_{a\in A}\sum_{j\in 1:nTiles} s(j,a)}$ Tennekes et al. (2019). Here A is the set of all antennas, $nTiles$ the number of tiles in the grid, and $s(j,a)$ is the signal quality or signal strength received from antenna a in the center the tile j.

We built two visualizations of the posterior location probabilities, plotting these probabilities for a selected device at different time instants of the simulation. The first one presented in Figure5.7 is built using an uniform prior probability while the second in Figure5.8 used the network prior.

To see very clear the difference in the location probabilities in different tiles, in Figure5.7 we grouped the probability values in 4 intervals and presented different intervals with different shading intensity: the darkest area is the area with the highest location probability and the lightest area is the region with the smallest location probability. The blue point is the device and the red point is the antenna where the device is connected at a certain time instant, the rest of the antennas being depicted with black. As it is expected, the highest values for the location probability are around the antenna where the device is connected, but this area is quite large due to the uniform prior.

In Figure5.8 we used the network prior and represented the values in the top quartile of the posterior location probabilities values with a shaded area. It can be easily observed that the prior used in this example resulted in a much better localization of the device compared to the localization computed using the uniform prior.

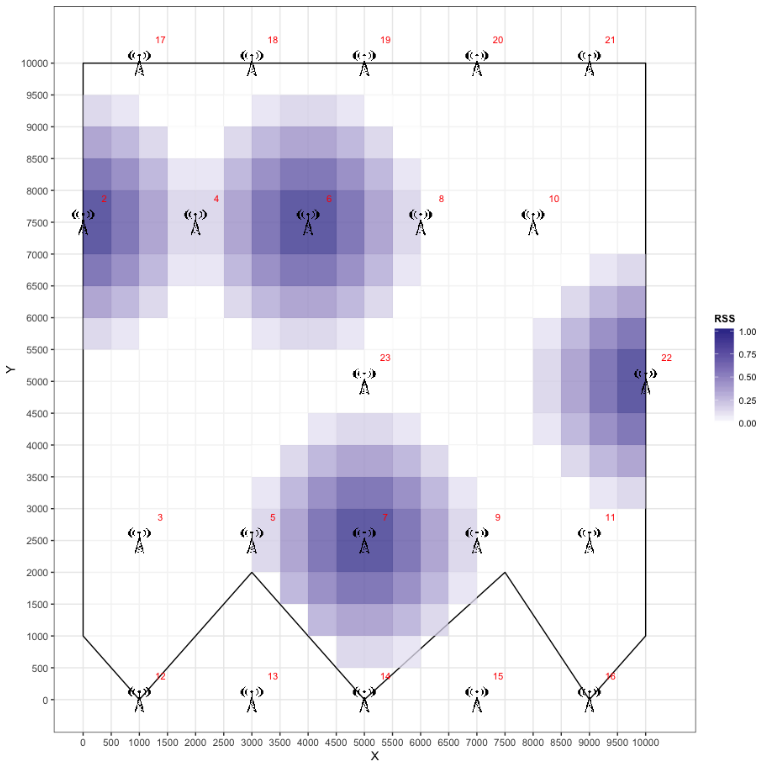


Figure 5.4: Signal strength for some omnidirectional antennas with IDs 2,6,7, 22

Using the posterior localization probabilities one can compute a point estimation (mean, mode etc.) of the number of devices inside a well-delimited area. This estimation will be then used to infer the population number which can be compared with the real number of individuals inside the concerned area computed easily from `persons.csv` file that contains the exact position of each individual at every time instant of the simulation. Using this comparison, one can decide how good is the inference model and choose among several models.

To exemplify how simulation results can be used, we run a simulation with a population made up of 5000 individuals, one single mobile network operator, the map and antenna configuration already presented in Figure5.1, computed the posterior location probabilities with the network prior for the devices located inside the area defined by $x \in [0, 5000]$ and $y \in [7000, 10000]$, and obtained the results presented in Table5.1 . This is a comparison between the number of the estimated number of mobile devices and the real number that can be used to evaluate the quality of the estimations. Of course, one can observe that there are differences between the real numbers of mobile devices and the estimated ones, but this difference comes from the very simple way of computing the posterior location probabilities. The algorithm implemented in the simulation software is only for exemplification, a more accurate algorithm will be provided in a separate software package.

Time instant	Estimated number of devices	Real number of devices
0	245	297
49	277	286
99	270	360
149	273	396
199	274	390

Table 5.1: A comparison between estimated and real number of devices

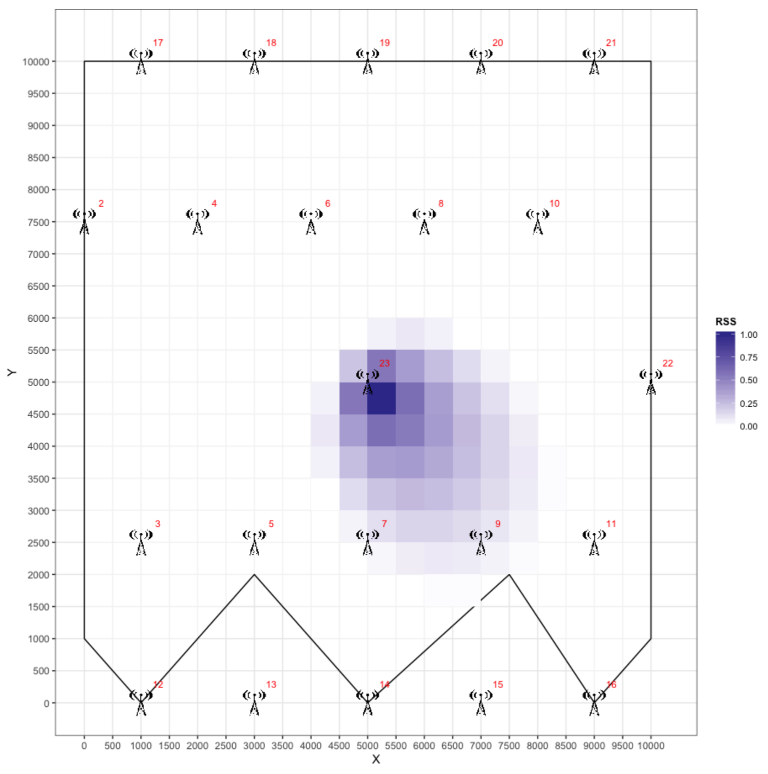


Figure 5.5: Signal strength for a directional antenna

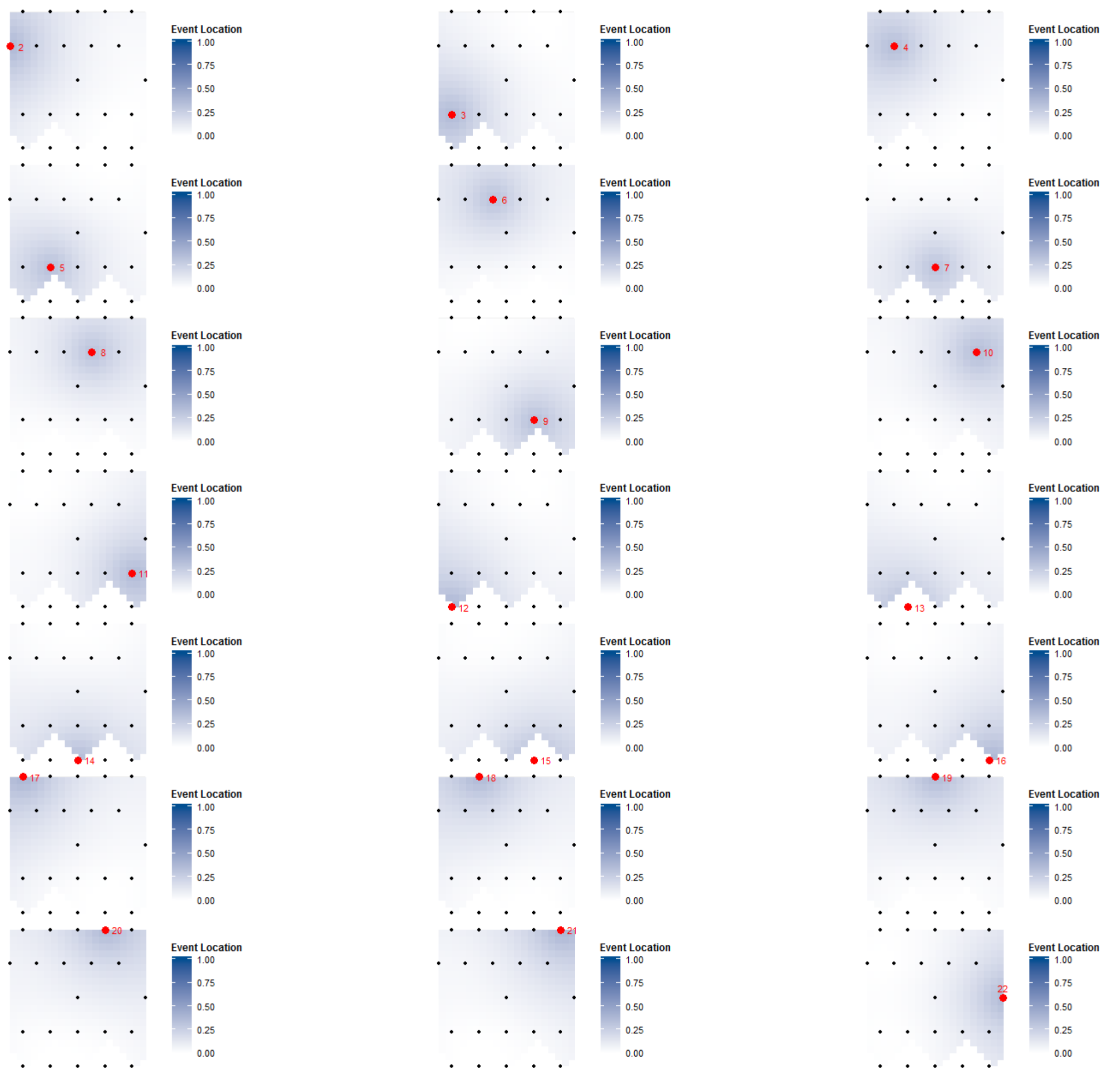
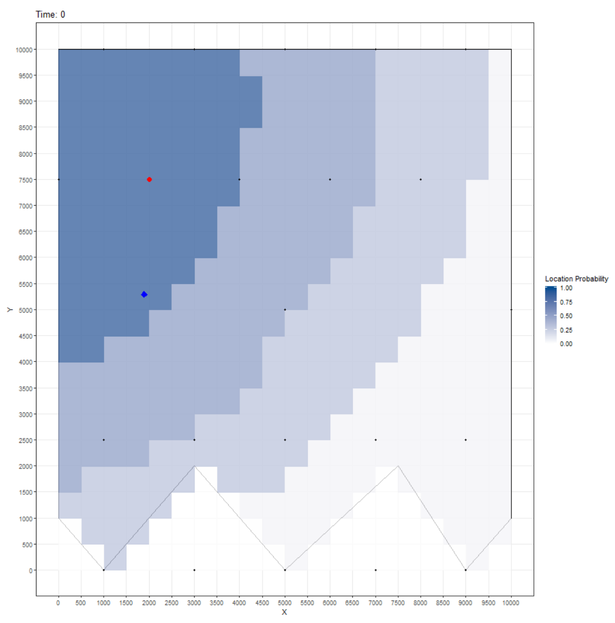
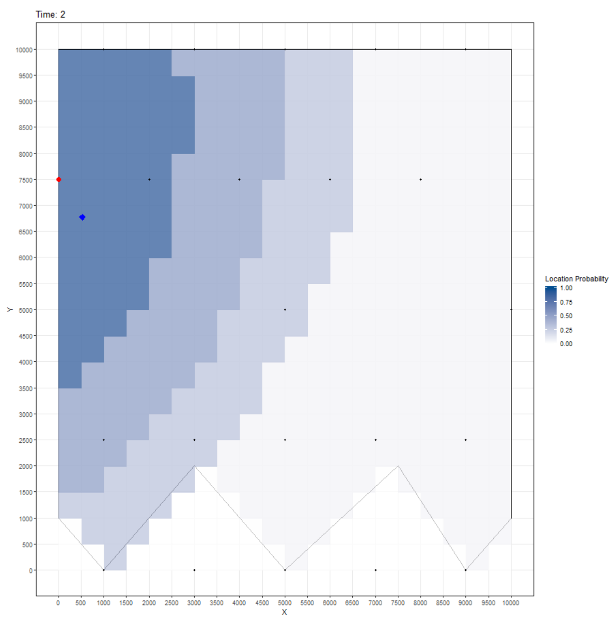
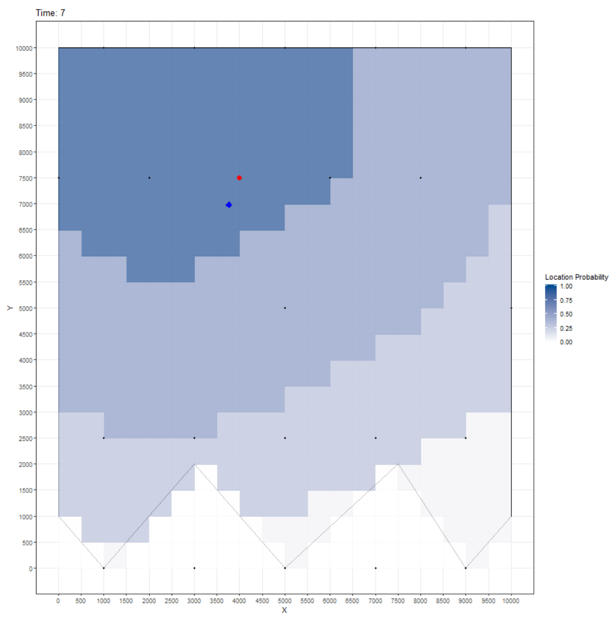
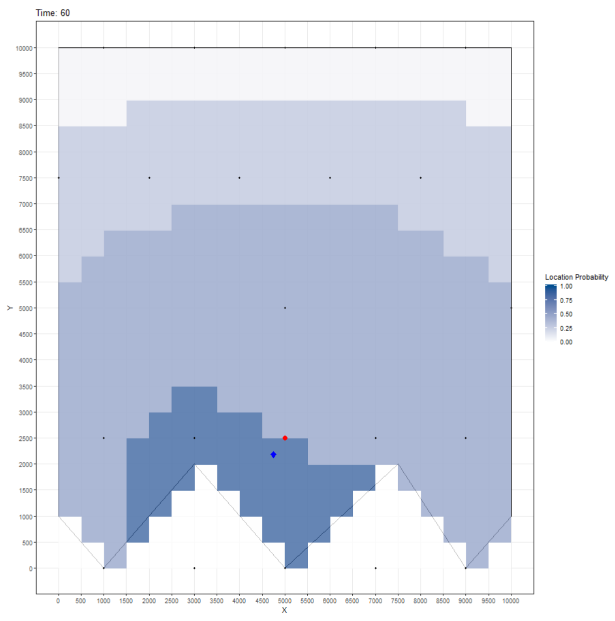


Figure 5.6: Signal strength for a directional antenna


Time: 0
Location Probability
1.00
0.75
0.50
0.25
0.00
X
Y


Time: 2
Location Probability
1.00
0.75
0.50
0.25
0.00
X
Y


Time: 7
Location Probability
1.00
0.75
0.50
0.25
0.00
X
Y


Time: 60
Location Probability
1.00
0.75
0.50
0.25
0.00
X
Y

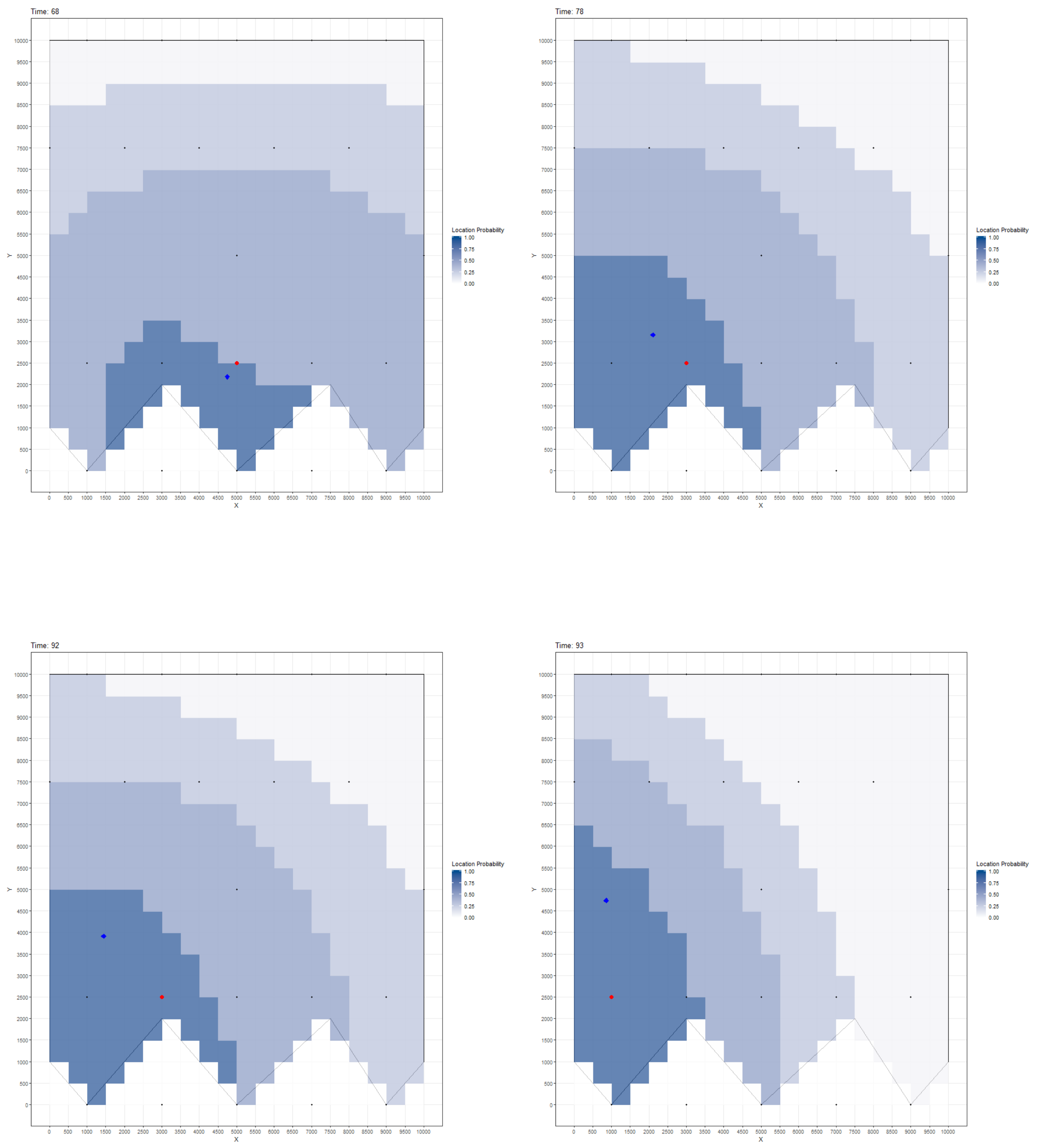
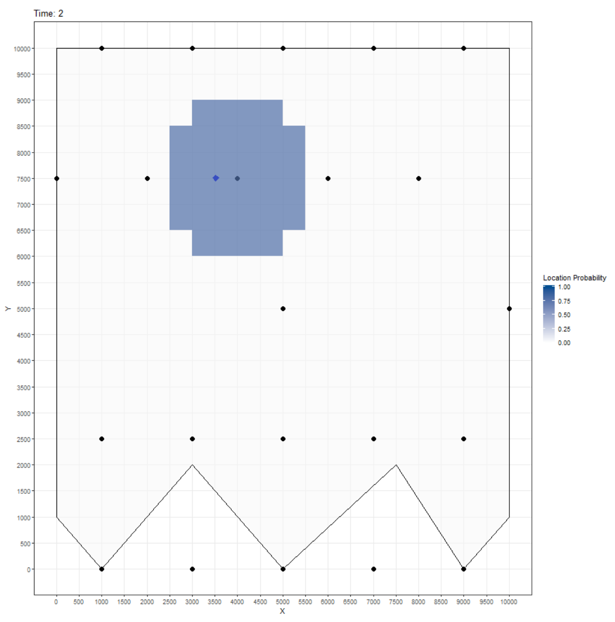
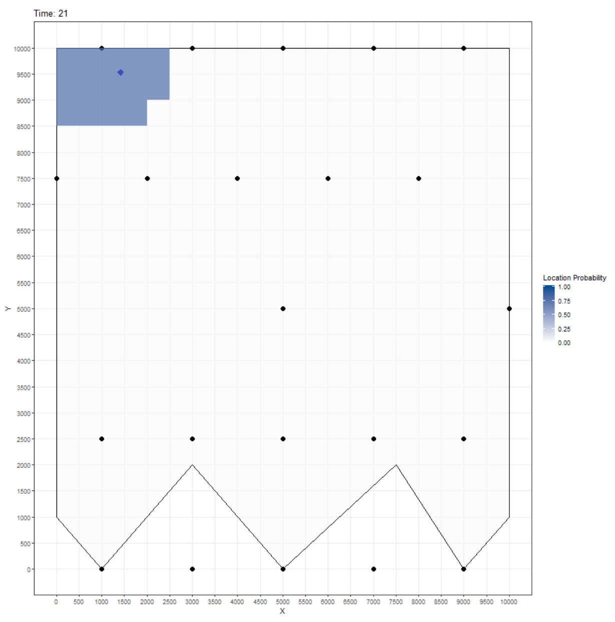
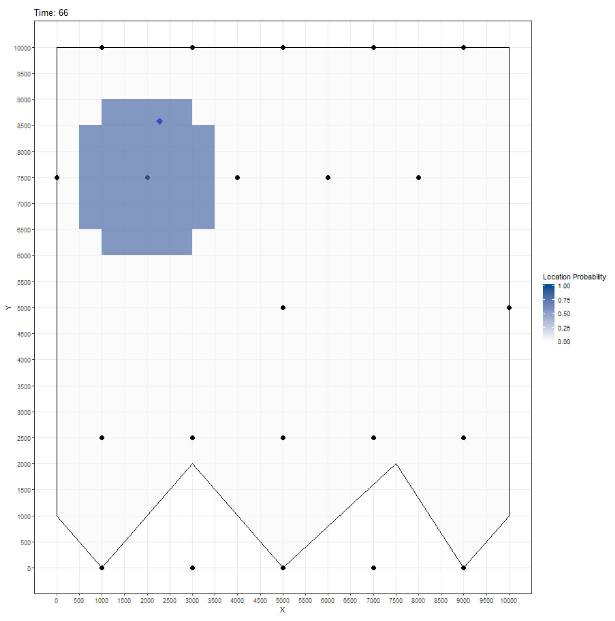
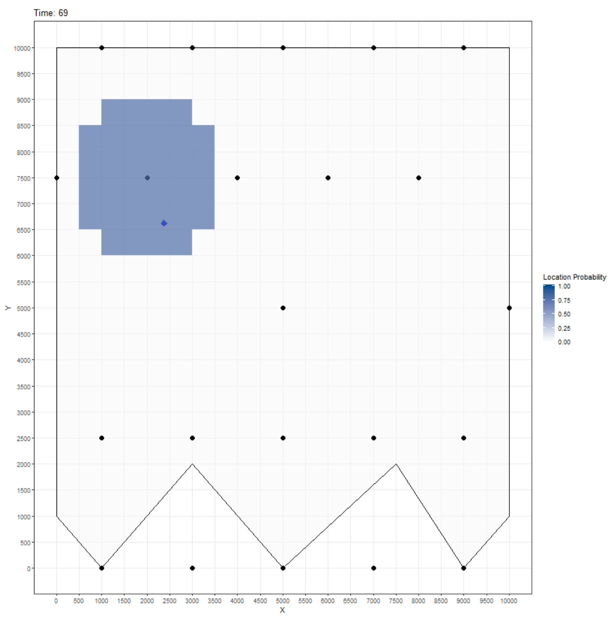


Figure 5.7: Posterior location probabilities for a device at different time instants using a uniform prior probability


Time: 2
Y
X
Location Probability
1.00
0.75
0.50
0.25
0.00


Time: 21
Y
X
Location Probability
1.00
0.75
0.50
0.25
0.00


Time: 66
Y
X
Location Probability
1.00
0.75
0.50
0.25
0.00


Time: 69
Y
X
Location Probability
1.00
0.75
0.50
0.25
0.00

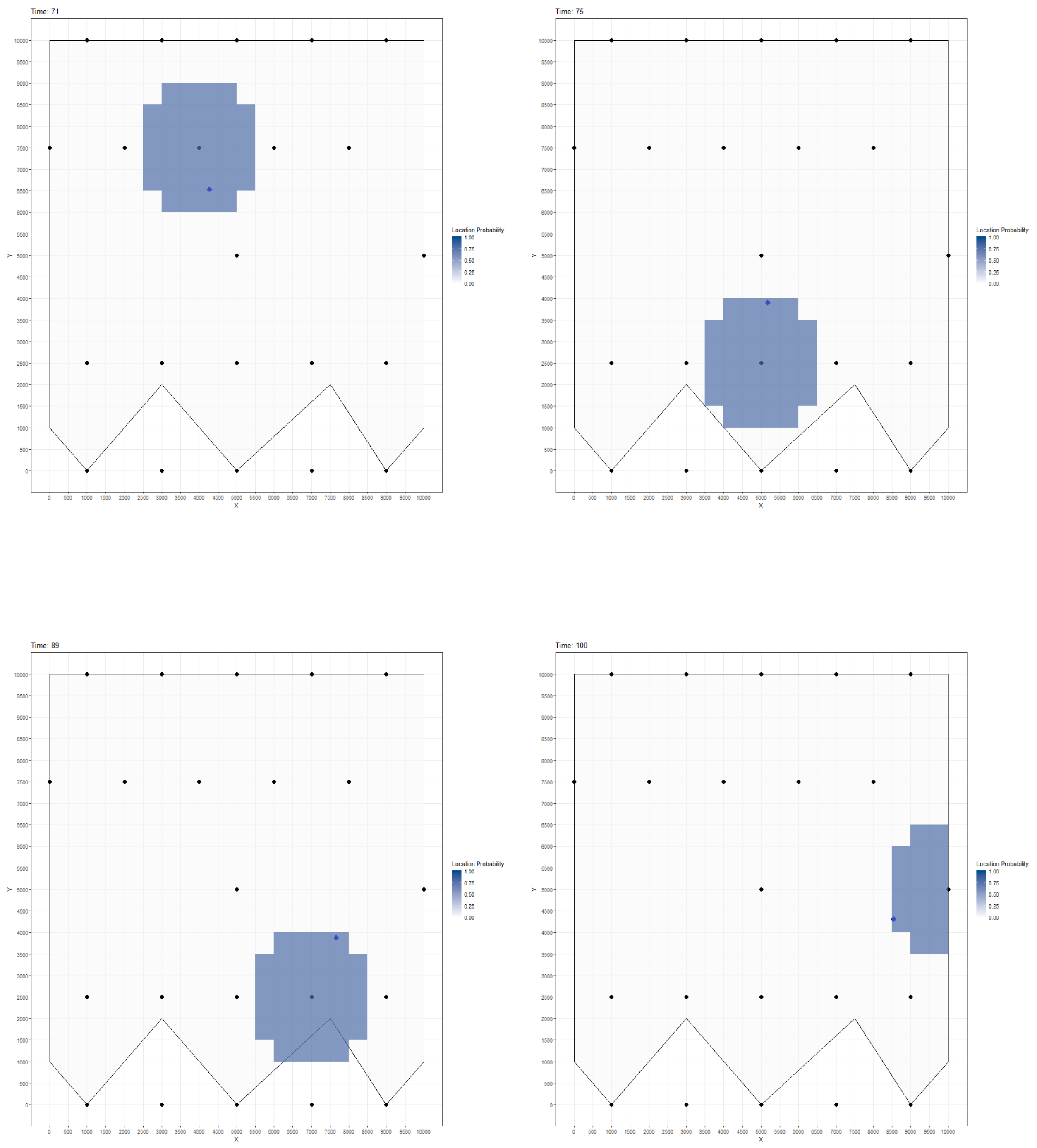


Figure 5.8: Posterior location probabilities for a device at different time instants using the network prior probability

6

Further developments

The simulation software is intended to be further developed to support the research on the methodology of estimating the population count using mobile phone data. As the main features that will be added in future versions we mention the following.

Generation of Call Detail Records
Call Detail Records (CDRs) are the other type of data produced by a mobile network, besides the network events. We intend to add a new functionality to the simulator to generate also CDRs. To be more realistic, the generation of CDRs with use the characteristics of the individuals: gender, age. Altghough network events provide a more accurate location probability, there are several existing applications that use CDRs and the simulation software can be used to check the accuracy of models underlying these applications.

Downtime periods for antennas
In real mobile networks, some antennas could be stopped from functioning for some periods of time for different reasons and the network traffic rerouted to neighbouring antennas. This feature will be added to our simulator in the future to support such events.

Mobile antennas
In the current version of the simulator, all antennas have fixed coordinates on the map. Nevertheless, we intend to add the possibility to change the location of some antennas during the simulation. This will emulate the case when a mobile operator brings new antennas to different locations to support an unusually high number of individuals in a small area, common to situations like sport events.

New technical parameters
One step further to improve the localization probabilities will be to introduce the use of Timing Advance which corresponds to the length of time a signal takes to reach the antenna from a mobile device. Other technical parameters of the antennas will be also introduced to make the signal propagation model to simulate as much as possible the real physical phenomena.

More complex maps
The maps used by the simulator will be enhanced to support roads and home locations. The roads will be introduced in conjuction with more complex displacement patterns while their home locations will define certain areas inside the map where individuals are most likely to be during some periods of the day (for example during the nights). The maps will also contain elevation information for each tile, now the map being flat.

Population displacements patterns
Besides the two patterns of displacement currently supported we intend to add more complex ones:

- Levy flights;
- Displacements along roads;

- Outgoing and incoming individuals: some of the individuals will be allowed to move outside the boundaries of the map, others will enter inside the map (from an outside region) at some time instants. This feature will create an open population, i.e. the number of individuals moving inside the map area will vary along the time;
- Sink points: they will be points (or very small areas) inside the map that will attract a number of individuals and at some time instants part of them will leave the map. They will emulate for example airports (when people dissappear from that points), or some social/sport events that attract a large number of persons).

Other improvements intended for the future are related to the software design, maintainability, performance and to installation and deplyoment of the software:

- Adding unit tests. They will allow us to have a control over the changes and newly introduced features to avoid introducing bugs in the code already running correct.
- Adding a logging library that will enable the software to log some events in external files. The log files will allow us to easilyl identify malfunctions of the software and correct them.
- Improving the documentation. This is already an ongoing effort to make the software easy to install and use.
- Transforming some computational intensive procedures to support a multithreading approach in order to improve the performance for very large populations and number of antennas. This will include a research among available multithreading libraries to choose one that best fits our needs: easy to use, scalable, freely available.
- Introducing an additional interface layer that will decouple the main part of the simulator from the current GIS library. This will allow us to replace GEOS C++ library in the future with other GIS libraries without affecting the rest of the software. The capability of replacing the GIS library without affecting the rest of the software is useful if other features unsupported by GEOS are to be implemented in the future.
- Creating an installation kit for Windows and Mac OS. This will ease the installation and will make the software accesible for statisticians without an IT background.
- Creating **docker images** that can be deployed in a container. Similar to the previous points, this will make our simulation software easy to distribute an use.

Appendix A

Example configuration files

map.wkt:

```
POLYGON ((0.0 1000.0, 1000.0 0.0, 3000.0 2000.0, 5000.0 0.0, 7500.0 2000.0,
9000.0 0.0, 10000.0 1000.0, 10000.0 10000.0, 0.0 10000.0, 0.0 1000.0))
```

simulation.xml:

```
<simulation>
    <start_time>0</start_time>
    <end_time>200</end_time>
    <time_increment>1</time_increment>
    <time_stay>10</time_stay>
    <interval_between_stays>25</interval_between_stays>
    <mno>
        <mno_name>MNO1</mno_name>
        <prob_mobile_phone>0.35</prob_mobile_phone>
    </mno>
    <mno>
        <mno_name>s</mno_name>
        <prob_mobile_phone>0.35</prob_mobile_phone>
    </mno>
    <prob_sec_mobile_phone>0.15</prob_sec_mobile_phone>
    <movement_type>random_walk_closed_map_drift</movement_type>
    <connection_type>strength</connection_type>
    <conn_threshold>-80</conn_threshold>
    <grid_file>grid.csv</grid_file>
    <grid_dim_tile_x>500</grid_dim_tile_x>
    <grid_dim_tile_y>500</grid_dim_tile_y>
    <persons_file>persons.csv</persons_file>
    <antennas_file>antennas.csv</antennas_file>
    <random_seed>12</random_seed>
</simulation>
```

probabilities.xml

```
<probabilities>
    <prior>network</prior>
    <prob_file_name_prefix>probabilities</prob_file_name_prefix>
</probabilities>
```

persons.xml

```
<persons>
    <num_persons>50</num_persons>
    <min_age>10</min_age>
    <max_age>87</max_age>
    <age_distribution>
        <type>normal</type>
        <mean>42</mean>
        <sd>25</sd>
    </age_distribution>
    <male_share>0.48</male_share>
    <speed_walk>200</speed_walk>
    <speed_car>1000</speed_car>
    <percent_home>0.10</percent_home>
</persons>
```

antennas.xml

```
 <antennas>
    <antenna>
        <mno_name>MNO1</mno_name>
        <maxconnections>100</maxconnections>
        <power>10</power>
        <attenuationfactor>3.55</attenuationfactor>
        <type>omnidirectional</type>
        <Smin>-80</Smin>
        <Smid>-92.5</Smid>
        <SSteep>0.2</SSteep>
        <x>0</x>
        <y>7500</y>
    </antenna>
    <antenna>
        <mno_name>MNO1</mno_name>
        <maxconnections>100</maxconnections>
        <power>10</power>
        <attenuationfactor>3.55</attenuationfactor>
        <type>omnidirectional</type>
        <Smin>-80</Smin>
        <Smid>-92.5</Smid>
        <SSteep>0.2</SSteep>
        <x>1000</x>
        <y>2500</y>
    </antenna>
    <antenna>
        <mno_name>MNO1</mno_name>
        <maxconnections>100</maxconnections>
        <power>10</power>
        <attenuationfactor>3.55</attenuationfactor>
        <type>omnidirectional</type>
        <Smin>-80</Smin>
        <Smid>-92.5</Smid>
        <SSteep>0.2</SSteep>
        <x>2000</x>
        <y>7500</y>
```

```
</antenna>
<antenna>
    <mno_name>MNO1</mno_name>
    <maxconnections>100</maxconnections>
    <power>10</power>
    <attenuationfactor>3.55</attenuationfactor>
    <type>omnidirectional</type>
    <Smin>-80</Smin>
    <Smid>-92.5</Smid>
    <SSteep>0.2</SSteep>
    <x>3000</x>
    <y>2500</y>
</antenna>
<antenna>
    <mno_name>MNO1</mno_name>
    <maxconnections>100</maxconnections>
    <power>10</power>
    <attenuationfactor>3.55</attenuationfactor>
    <type>omnidirectional</type>
    <Smin>-80</Smin>
    <Smid>-92.5</Smid>
    <SSteep>0.2</SSteep>
    <x>4000</x>
    <y>7500</y>
</antenna>
<antenna>
    <mno_name>MNO1</mno_name>
    <maxconnections>100</maxconnections>
    <power>10</power>
    <attenuationfactor>3.55</attenuationfactor>
    <type>omnidirectional</type>
    <Smin>-80</Smin>
    <Smid>-92.5</Smid>
    <SSteep>0.2</SSteep>
    <x>5000</x>
    <y>2500</y>
</antenna>
<antenna>
    <mno_name>MNO1</mno_name>
    <maxconnections>100</maxconnections>
    <power>10</power>
    <attenuationfactor>3.55</attenuationfactor>
    <type>omnidirectional</type>
    <Smin>-80</Smin>
    <Smid>-92.5</Smid>
    <SSteep>0.2</SSteep>
    <x>6000</x>
    <y>7500</y>
</antenna>
<antenna>
    <mno_name>MNO1</mno_name>
    <maxconnections>100</maxconnections>
```

```
        <power>10</power>
        <attenuationfactor>3.55</attenuationfactor>
        <type>omnidirectional</type>
        <Smin>-80</Smin>
        <Smid>-92.5</Smid>
        <SSteep>0.2</SSteep>
        <x>7000</x>
        <y>2500</y>
    </antenna>
    <antenna>
        <mno_name>MNO1</mno_name>
        <maxconnections>100</maxconnections>
        <power>10</power>
        <attenuationfactor>3.55</attenuationfactor>
        <type>omnidirectional</type>
        <Smin>-80</Smin>
        <Smid>-92.5</Smid>
        <SSteep>0.2</SSteep>
        <x>8000</x>
        <y>7500</y>
    </antenna>
    <antenna>
        <mno_name>MNO1</mno_name>
        <maxconnections>100</maxconnections>
        <power>10</power>
        <attenuationfactor>3.55</attenuationfactor>
        <type>omnidirectional</type>
        <Smin>-80</Smin>
        <Smid>-92.5</Smid>
        <SSteep>0.2</SSteep>
        <x>9000</x>
        <y>2500</y>
    </antenna>
    <antenna>
        <mno_name>MNO1</mno_name>
        <maxconnections>100</maxconnections>
        <power>10</power>
        <attenuationfactor>3.55</attenuationfactor>
        <type>omnidirectional</type>
        <Smin>-80</Smin>
        <Smid>-92.5</Smid>
        <SSteep>0.2</SSteep>
        <x>1000</x>
        <y>0</y>
    </antenna>
    <antenna>
        <mno_name>MNO1</mno_name>
        <maxconnections>100</maxconnections>
        <power>10</power>
        <attenuationfactor>3.55</attenuationfactor>
        <type>omnidirectional</type>
        <Smin>-80</Smin>
        <Smid>-92.5</Smid>
        <SSteep>0.2</SSteep>
        <x>3000</x>
        <y>0</y>
    </antenna>
    <antenna>
        <mno_name>MNO1</mno_name>
        <maxconnections>100</maxconnections>
        <power>10</power>
        <attenuationfactor>3.55</attenuationfactor>
        <type>omnidirectional</type>
        <Smin>-80</Smin>
        <Smid>-92.5</Smid>
        <SSteep>0.2</SSteep>
        <x>5000</x>
        <y>0</y>
    </antenna>
    <antenna>
        <mno_name>MNO1</mno_name>
        <maxconnections>100</maxconnections>
        <power>10</power>
        <attenuationfactor>3.55</attenuationfactor>
        <type>omnidirectional</type>
        <Smin>-80</Smin>
        <Smid>-92.5</Smid>
        <SSteep>0.2</SSteep>
        <x>7000</x>
        <y>0</y>
    </antenna>
    <antenna>
        <mno_name>MNO1</mno_name>
        <maxconnections>100</maxconnections>
        <power>10</power>
        <attenuationfactor>3.55</attenuationfactor>
        <type>omnidirectional</type>
        <Smin>-80</Smin>
        <Smid>-92.5</Smid>
        <SSteep>0.2</SSteep>
        <x>9000</x>
        <y>0</y>
    </antenna>
    <antenna>
        <mno_name>MNO1</mno_name>
        <maxconnections>100</maxconnections>
        <power>10</power>
        <attenuationfactor>3.55</attenuationfactor>
        <type>omnidirectional</type>
        <Smin>-80</Smin>
        <Smid>-92.5</Smid>
        <SSteep>0.2</SSteep>
        <x>1000</x>
        <y>10000</y>
```

```
</antenna>
<antenna>
    <mno_name>MNO1</mno_name>
    <maxconnections>100</maxconnections>
    <power>10</power>
    <attenuationfactor>3.55</attenuationfactor>
    <type>omnidirectional</type>
    <Smin>-80</Smin>
    <Smid>-92.5</Smid>
    <SSteep>0.2</SSteep>
    <x>3000</x>
    <y>10000</y>
</antenna>
<antenna>
    <mno_name>MNO1</mno_name>
    <maxconnections>100</maxconnections>
    <power>10</power>
    <attenuationfactor>3.55</attenuationfactor>
    <type>omnidirectional</type>
    <Smin>-80</Smin>
    <Smid>-92.5</Smid>
    <SSteep>0.2</SSteep>
    <x>5000</x>
    <y>10000</y>
</antenna>
<antenna>
    <mno_name>MNO1</mno_name>
    <maxconnections>100</maxconnections>
    <power>10</power>
    <attenuationfactor>3.55</attenuationfactor>
    <type>omnidirectional</type>
    <Smin>-80</Smin>
    <Smid>-92.5</Smid>
    <SSteep>0.2</SSteep>
    <x>7000</x>
    <y>10000</y>
</antenna>
<antenna>
    <mno_name>MNO1</mno_name>
    <maxconnections>100</maxconnections>
    <power>10</power>
    <attenuationfactor>3.55</attenuationfactor>
    <type>omnidirectional</type>
    <Smin>-80</Smin>
    <Smid>-92.5</Smid>
    <SSteep>0.2</SSteep>
    <x>9000</x>
    <y>10000</y>
</antenna>
<antenna>
    <mno_name>MNO1</mno_name>
    <maxconnections>100</maxconnections>
```

```
        <power>10</power>
        <attenuationfactor>3.55</attenuationfactor>
        <type>omnidirectional</type>
        <Smin>-80</Smin>
        <Smid>-92.5</Smid>
        <SSteep>0.2</SSteep>
        <x>10000</x>
        <y>5000</y>
    </antenna>
    <antenna>
        <mno_name>MNO2</mno_name>
        <maxconnections>100</maxconnections>
        <power>10</power>
        <attenuationfactor>3.55</attenuationfactor>
        <type>directional</type>
        <Smin>-75</Smin>
        <tilt>5</tilt>
        <azim_dB_back>-30</azim_dB_back>
        <elev_dB_back>-30</elev_dB_back>
        <beam_h>65</beam_h>
        <beam_v>9</beam_v>
        <direction>135</direction>
        <Smid>-92.5</Smid>
        <SSteep>0.2</SSteep>
        <x>5000</x>
        <y>5000</y>
        <z>10</z>
    </antenna>
</antennas>
```

Appendix B

The detailed class diagrams

We present in this annex the detailed inheritance and collaboration diagrams of the component classes.

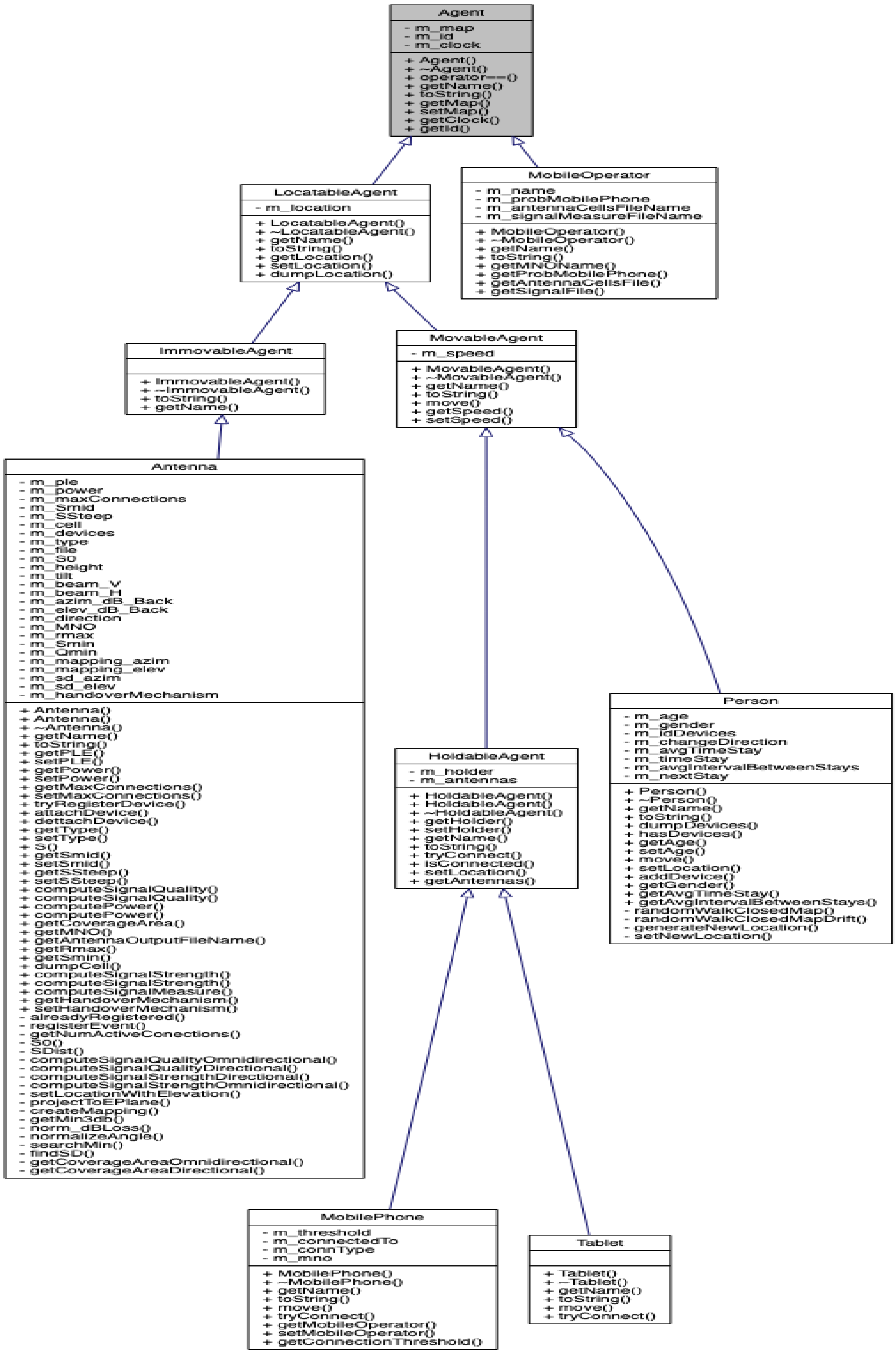


Figure B.1: Agent class inheritance diagram

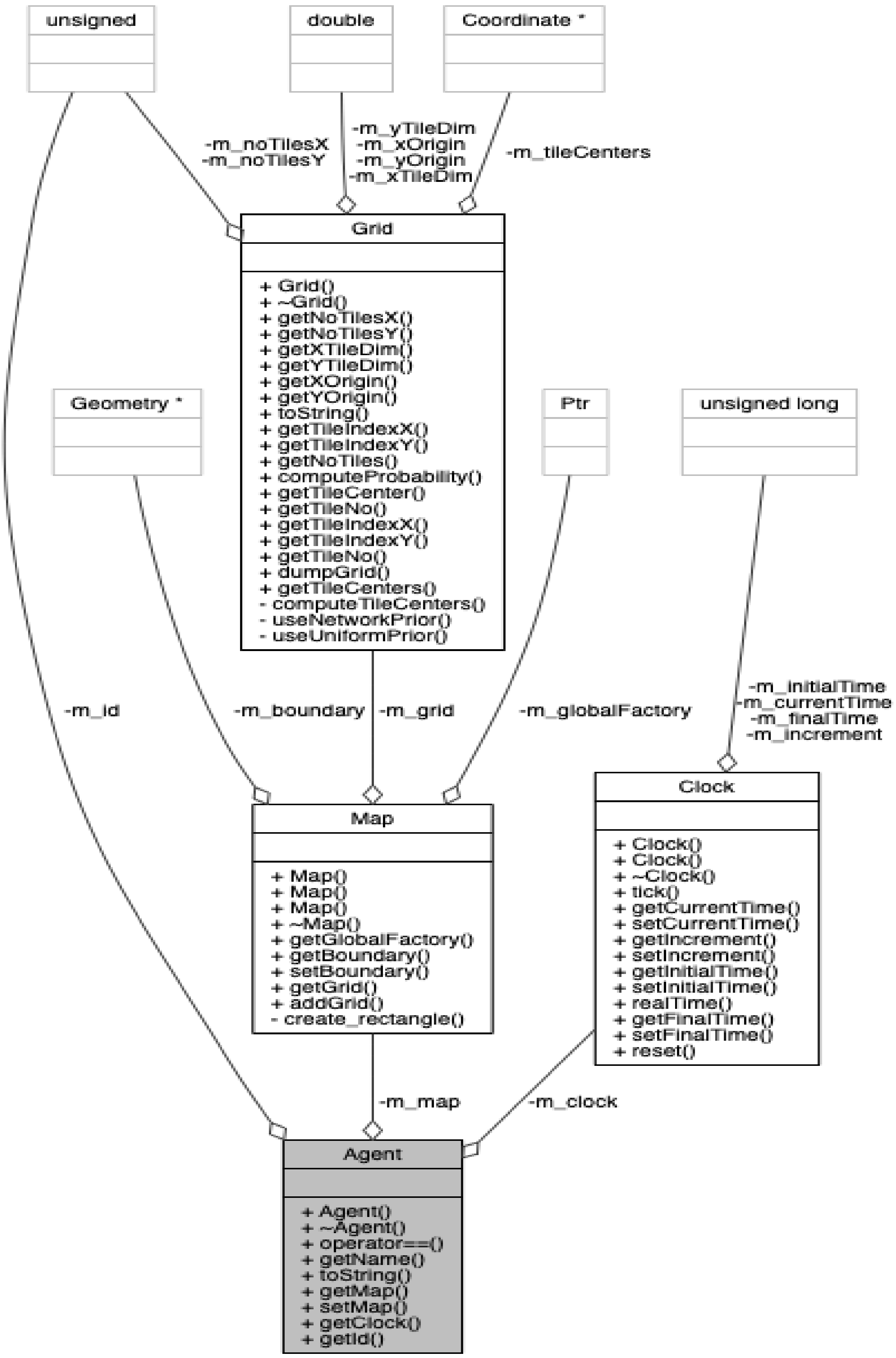


Figure B.2: Agent class collaboration diagram

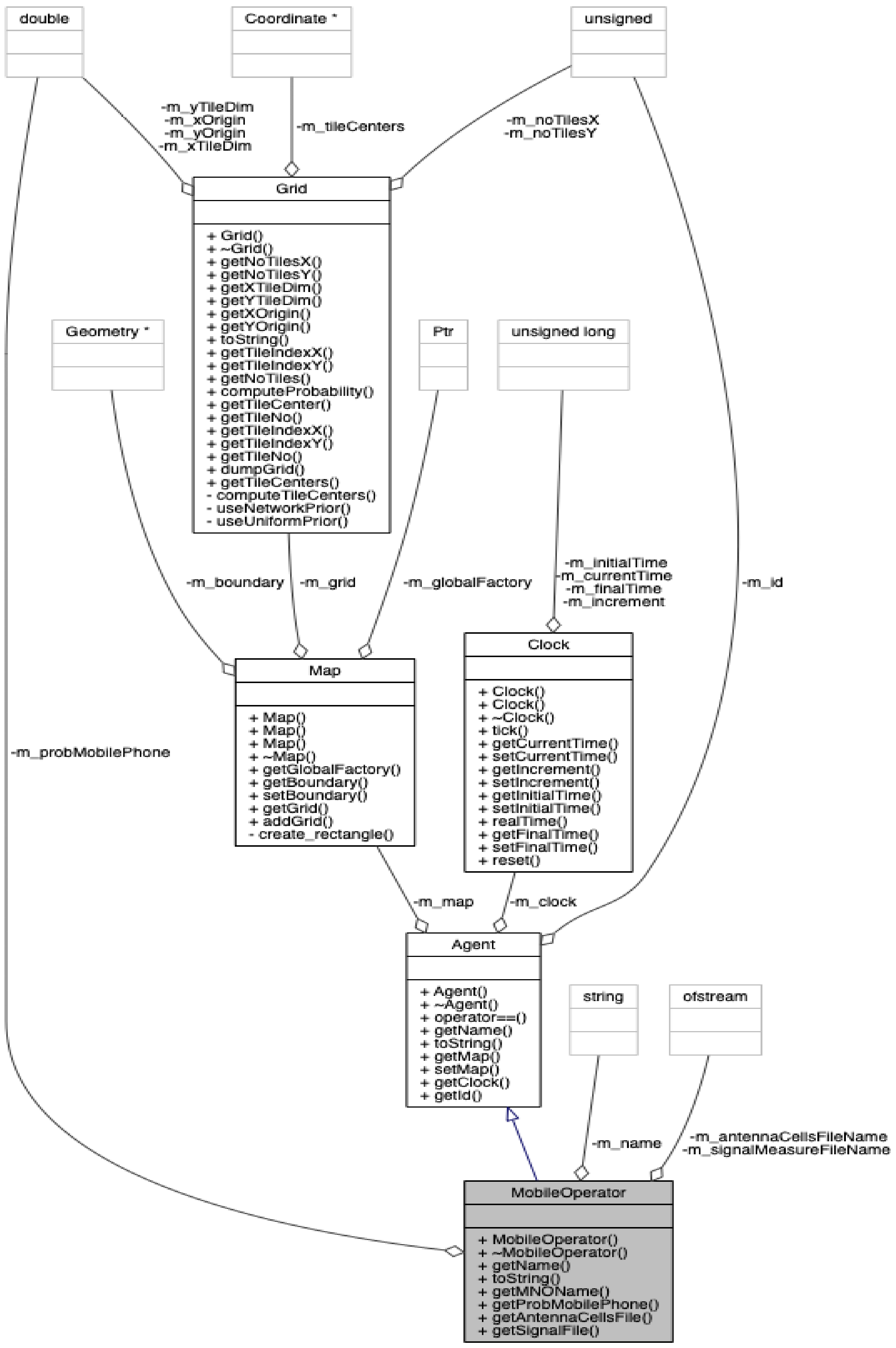


Figure B.3: MobileOperator class collaboration diagram

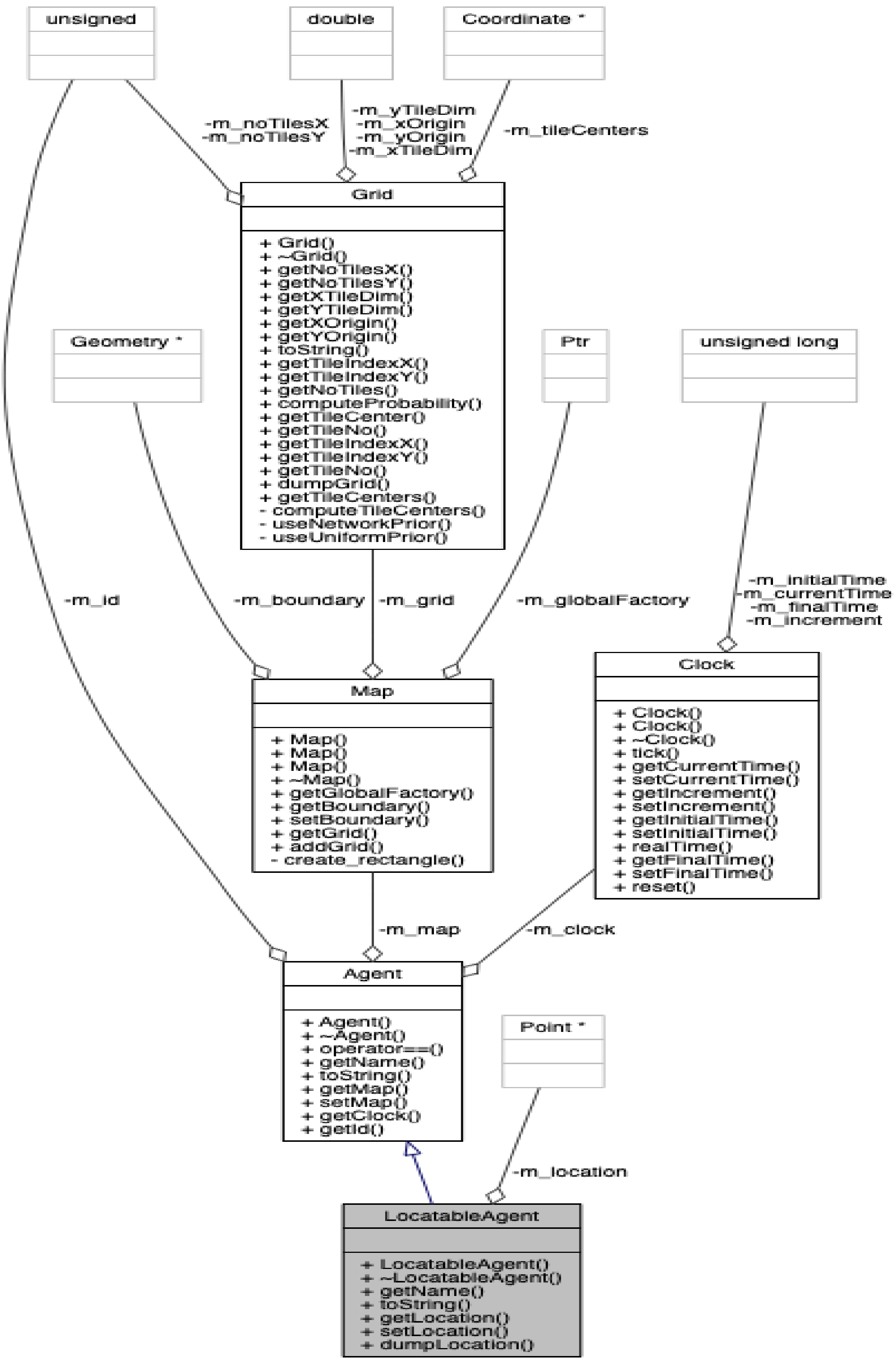


Figure B.4: LocatableAgent class collaboration diagram

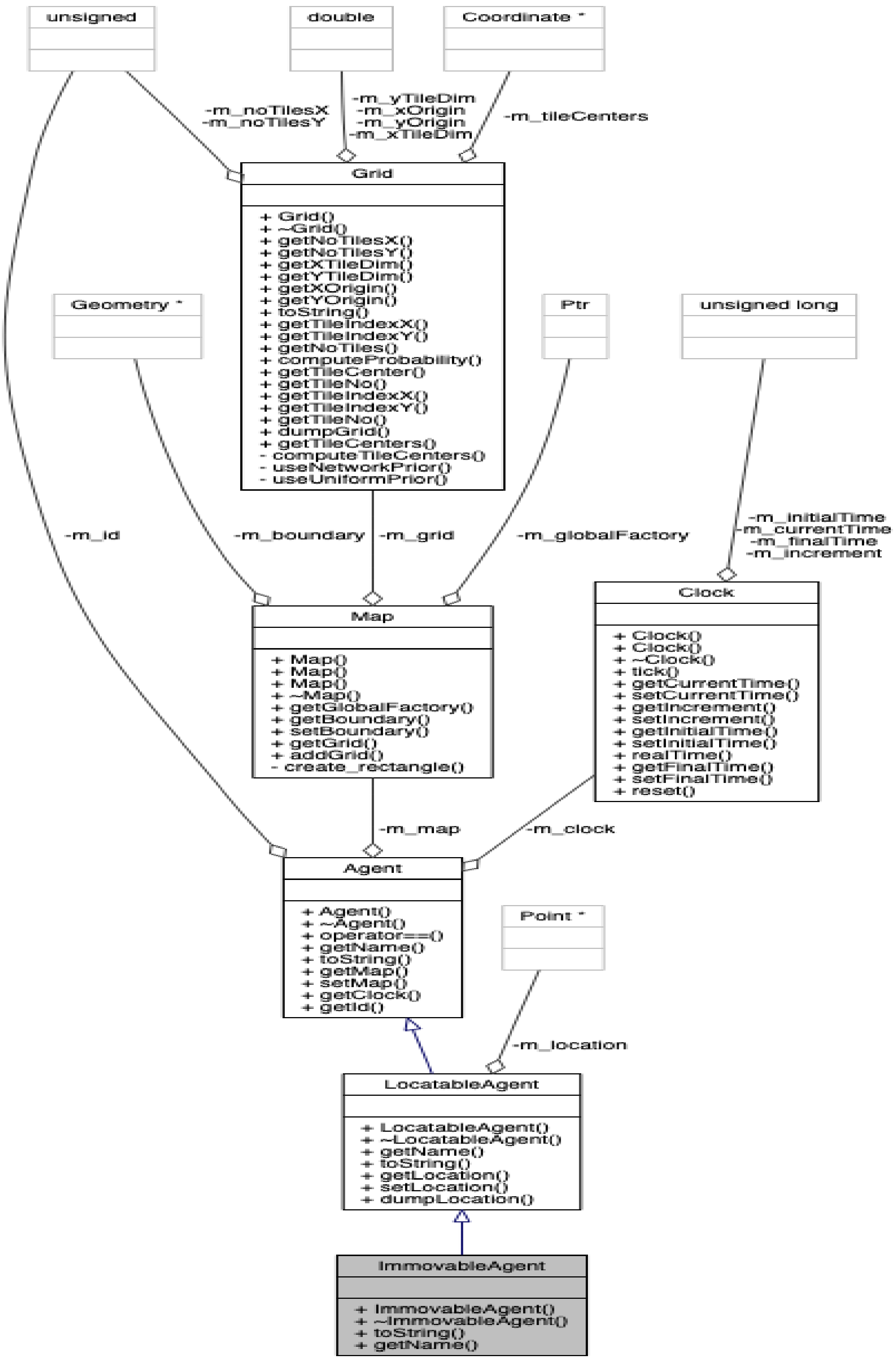


Figure B.5: ImmovableAgent class collaboration diagram

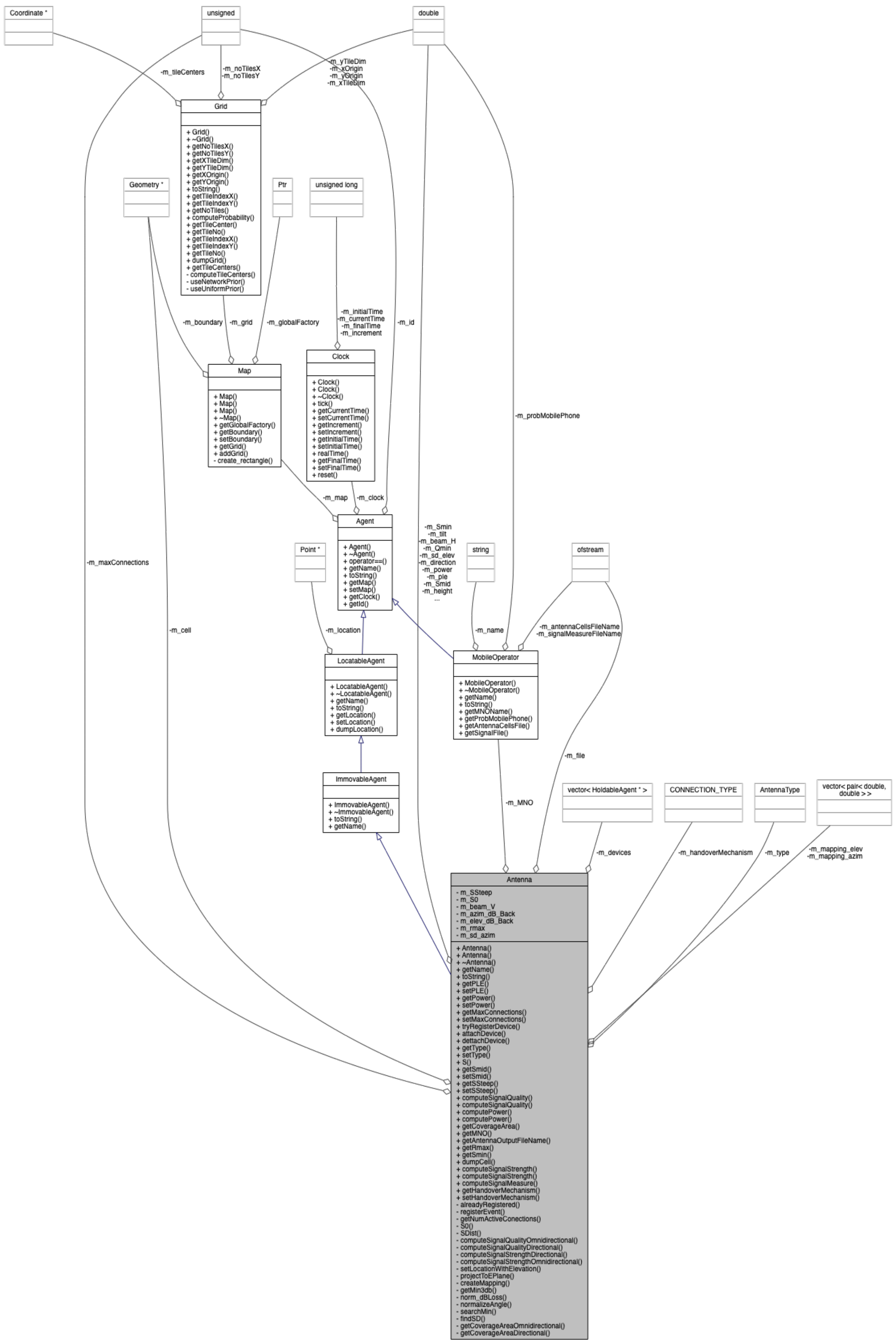


Figure B.6: Antenna class collaboration diagram

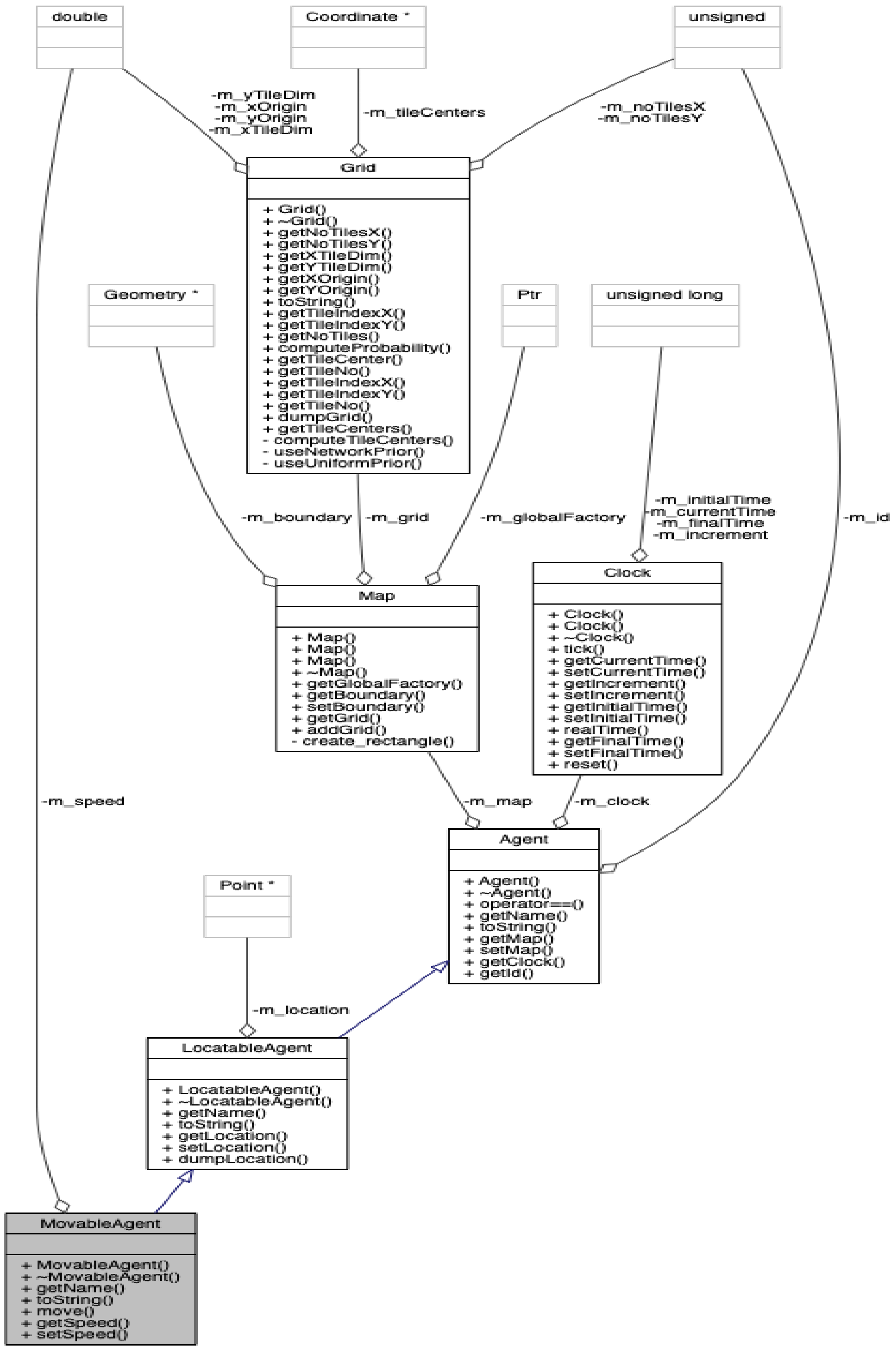


Figure B.7: MovableAgent class collaboration diagram

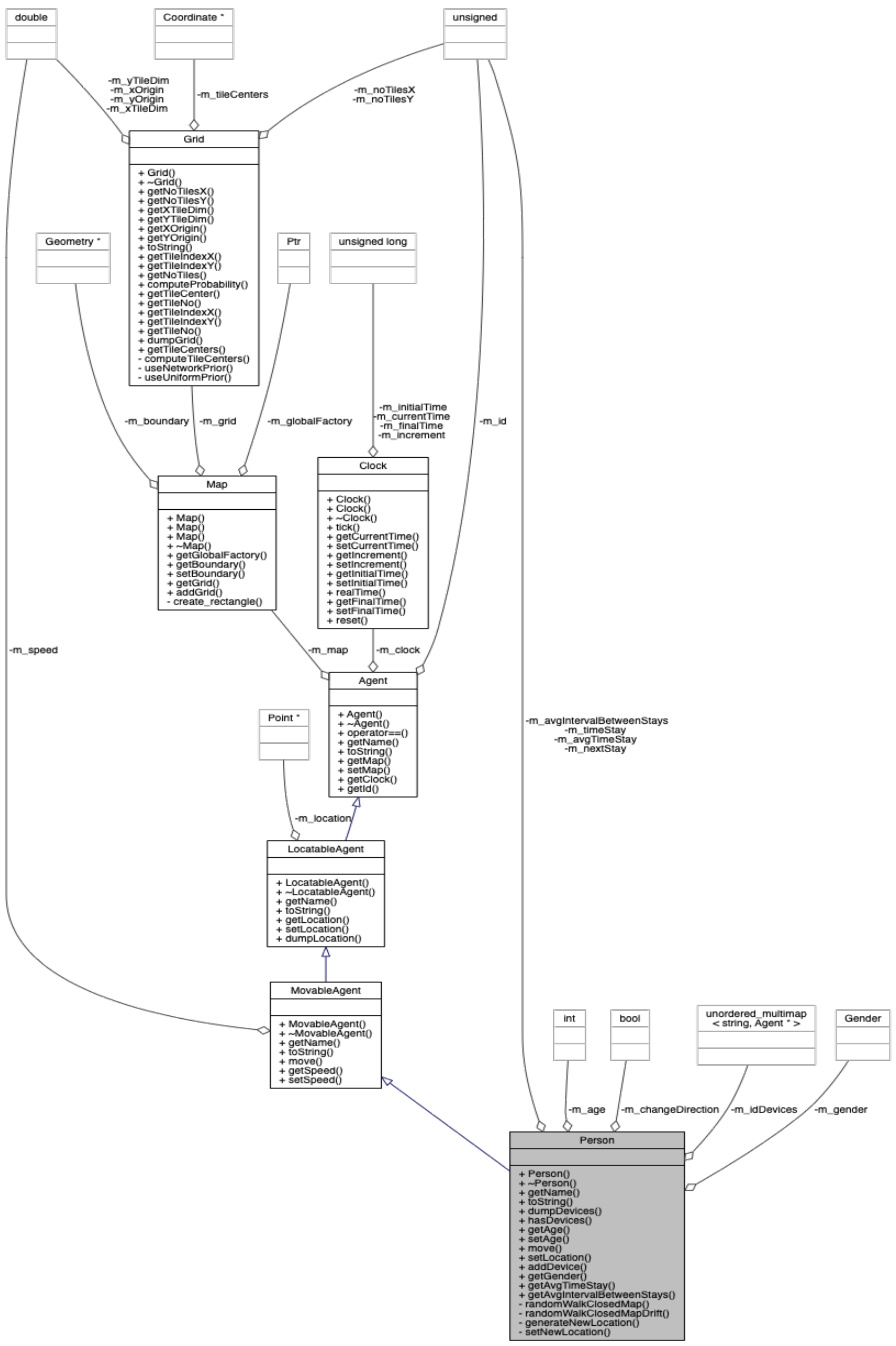


Figure B.8: Person class collaboration diagram

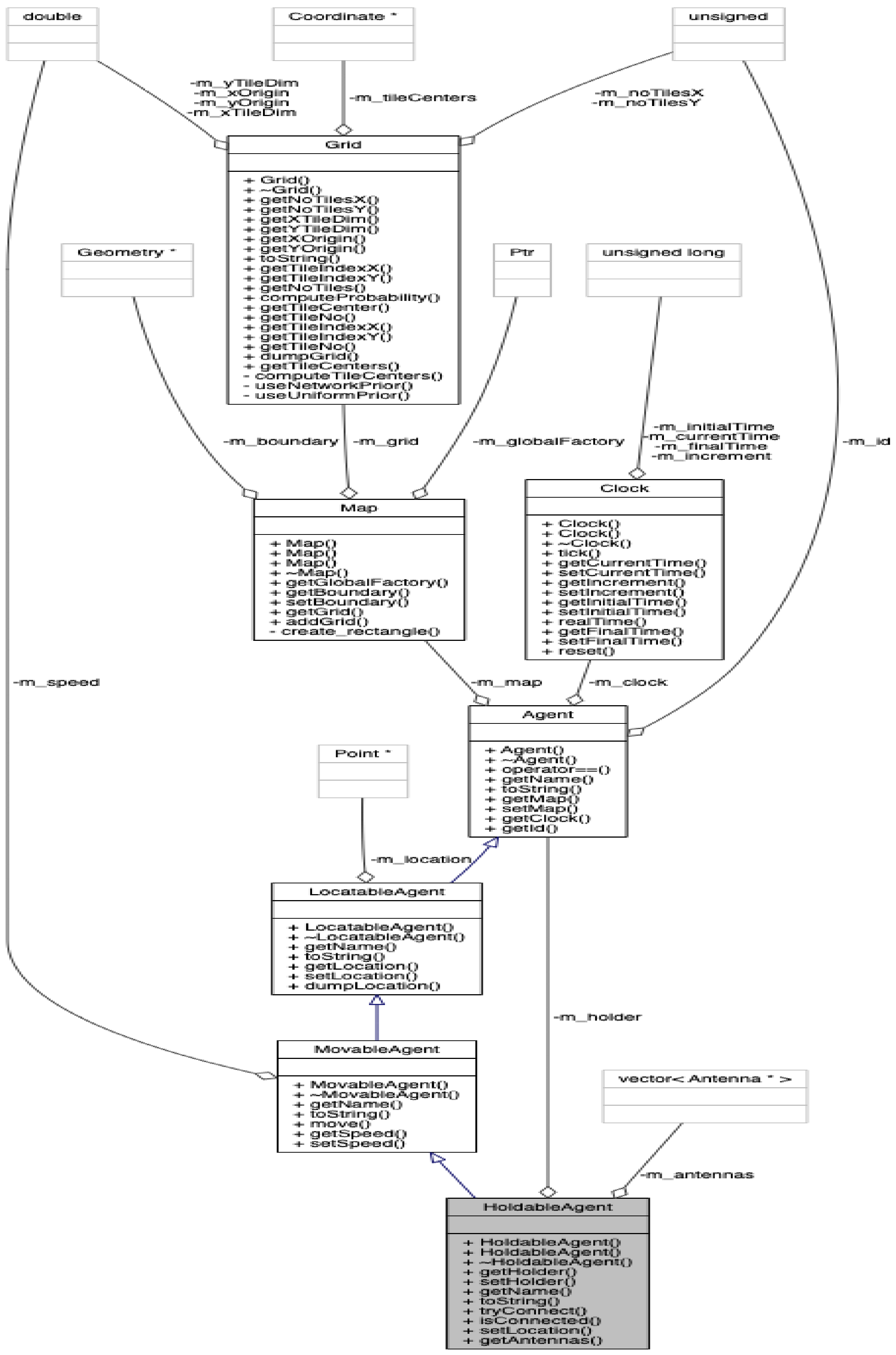


Figure B.9: HoldableAgent class collaboration diagram

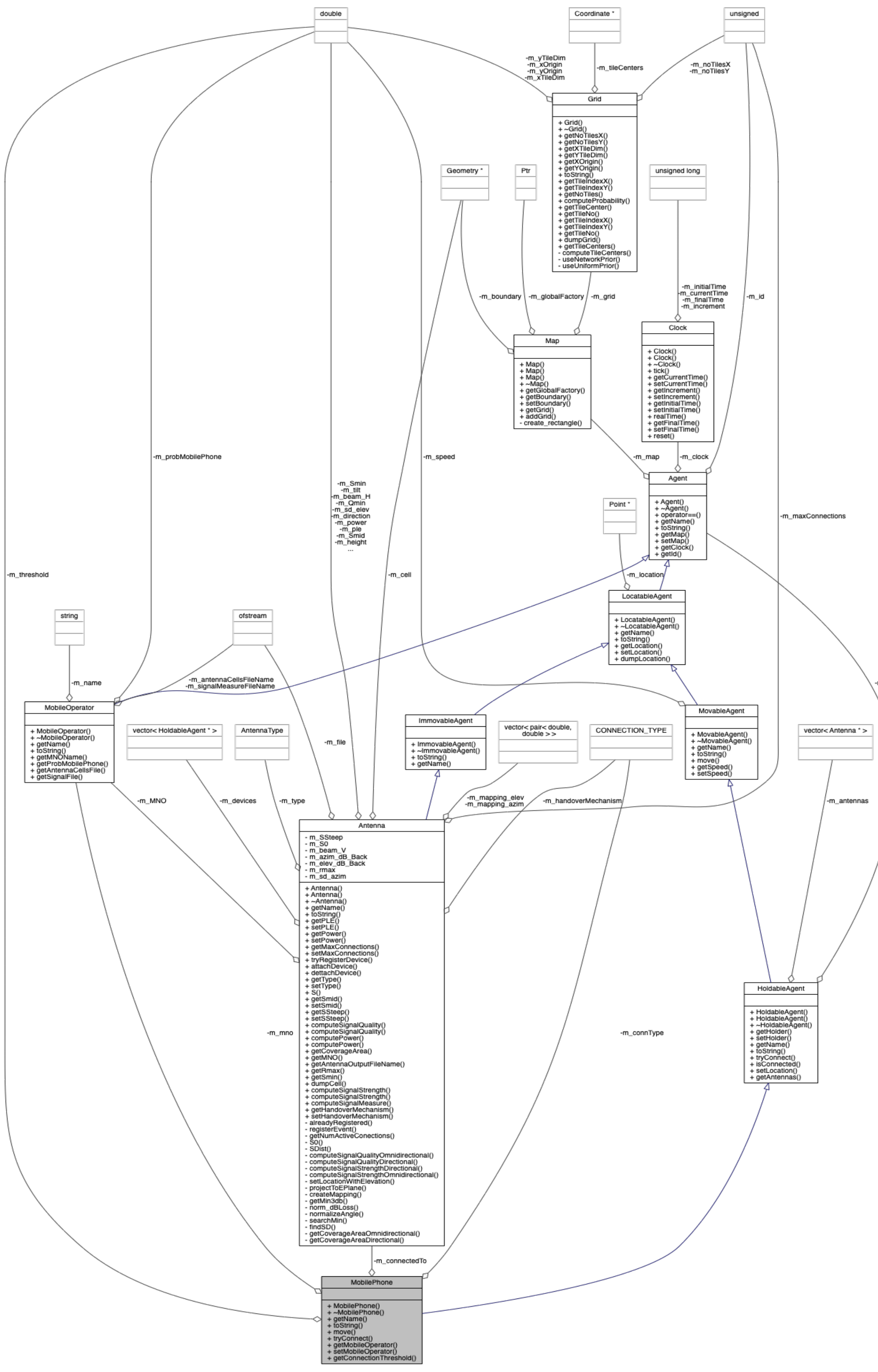


Figure B.10: MobilePhone class collaboration diagram

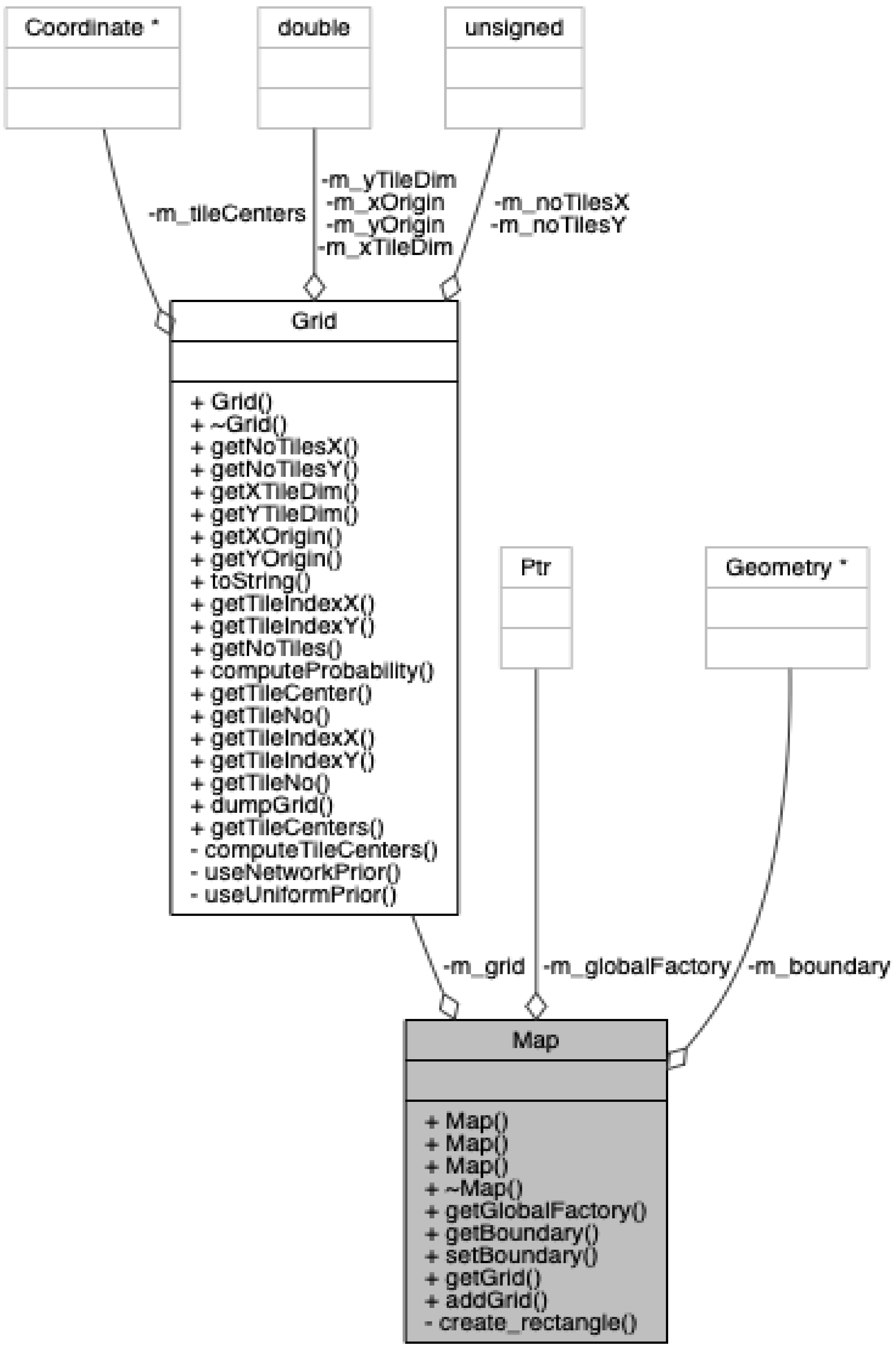


Figure B.11: Map class collaboration diagram

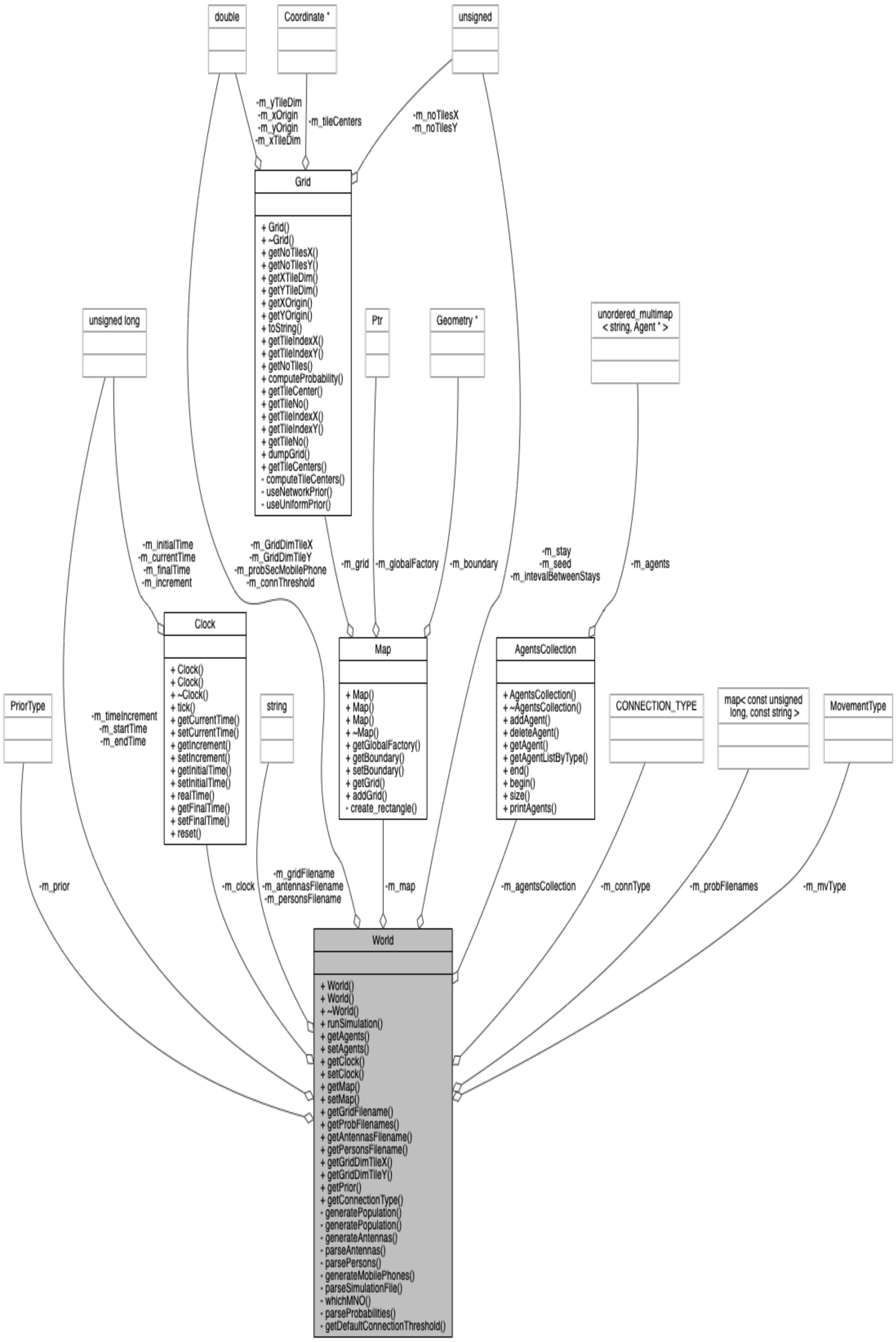


Figure B.12: World class collaboration diagram

Bibliography

UN Global Working Group on Big Data for Official Statistics (2017). *Handbook on the use of Mobile Phone data for Official Statistics*.

Bordin, M. V. (2017). *A Call Detail Record (CDR) generator*. `https://github.com/mayconbordin/cdr-gen`.

Real Impact Analysis (2014). *CDR generator*. `https://github.com/RealImpactAnalytics/cdr-generator`.

Tetcos (2019). *NetSim User Manual*. `https://www.tetcos.com/downloads/v12/NetSim_User_Manual.pdf`.

Zhen, L., Hongji, Y. (2012) *Unlocking the Power of OPNET Modeler.* Cambridge University Press, New York.

Krajzewicz, D., J. Erdmann, M. Behrisch, and L. Bieker (2012). *Recent Development and Applications of SUMO - Simulation of Urban MObility*. *Journal On Advances in Systems and Measurements 5* (3&4), 128–138.

Horni, A., K. Nagel, and K. W. Axhausen (2016). *The Multi-Agent Transport Simulation MATSim*. Ubiquity Press, London.

Schelling, T. (1971). *Dynamic Models of Segregation*. *Journal of Mathematical Sociology 1* (2), 143–186.

Axelrod, R. (1997). *The Complexity of Cooperation: Agent-Based Models of Competition and Collaboration*. Princeton University Press, Princeton.

WP5 of ESSnet on Big Data (2017). Guidelines for the access to mobile phone data within the ESS. Deliverable 5.2. `https://webgate.ec.europa.eu/fpfis/mwikis/essnetbigdata/images/6/65/WP5.Deliverable1.2.pdf`.

WP5 of ESSnet on Big Data (2018). Proposed Elements for a Methodological Framework for the Production of Official Statistics with Mobile Phone Data. Deliverable 5.3. `https://webgate.ec.europa.eu/fpfis/mwikis/essnetbigdata/images/4/4d/WP5_Deliverable_5.3_Final.pdf`.

WP5 of ESSnet on Big Data (2018). Some IT elements for the use of mobile phone data in the production of official statistics. Deliverable 5.4. `https://webgate.ec.europa.eu/fpfis/mwikis/essnetbigdata/images/c/ce/WP5_Deliverable_5.4_Final.pdf`.

Stroustrup, B. (2013). *The C++ Programming Language*. Addison-Wesley Professional, Upper Saddle River, New Jersey.

Stroustrup, B. (2018). *A Tour of C++ (Second edition)*. Addison-Wesley Professional, Upper Saddle River, New Jersey.

Bass, L., P. Clements, and R. Kazman. (2012). *Software Architecture in Practice*. Addison-Wesley Professional, Upper Saddle River, New Jersey.

ISO/IEC. (2017). *ISO/IEC 14882:2017 Programming languages — C++*. ISO, Geneva, Switzerland.

ISO/IEC. (2016). *ISO/IEC 13249-3:2016(en) Information technology — Database languages — SQL multimedia and application packages — Part 3: Spatial*. ISO, Geneva, Switzerland.

Open Geospatial Consortium. (2011). *OpenGIS® Implementation Standard for Geographic information - Simple feature access - Part 1: Common architecture*. OGC Wayland, USA

Tennekes, M. (2018). *mobloc - R package vignette*. `https://github.com/MobilePhoneESSnetBigData/mobloc_v0.1`

Tennekes, M., Y. A. P. M. Gootzen, and S. Shan (2019). *Deriving geographic location of mobile devices from network data*. Private communication.

Y. Shafranovich. (2005). *RFC 4180 - Common Format and MIME Type for Comma-Separated Values (CSV) Files*. IThe Internet Society. `https://tools.ietf.org/pdf/rfc4180.pdf`.

Tim Bray, Jean Paoli, C.M. Sperberg-McQueen, Eve Maler, Francois Yergeau eds. (2008). *Extensible Markup Language (XML) 1.0 (Fifth Edition) W3C Recommendation*. `https://www.w3c.org`.

Panwar, N., S. Sharma, and A. K. Singh (2016). *A survey on 5G: The next generation of mobile communication. Physical Communication. Special Issue on Radio Access Network Architectures and Resource Management for 5G. 18*, 64–84.

GDAL/OGR contributors (2019). *GDAL/OGR Geospatial Data Abstraction software Library. Open Source Geospatial Foundation.*. `https://gdal.org`.

GEOS contributors (2019). *GEOS (Geometry Engine – Open Source) . The Open Source Geospatial Foundation.*. `https://www.osgeo.org/projects/geos/`.

GRASS Development Team (2019). *Geographic Resources Analysis Support System (GRASS GIS) Software, Version 7.2. Open Source Geospatial Foundation.*. `http://grass.osgeo.org`.

EPSG (2018). epsg.io – Coordinate Systems Worldwide. `https://epsg.io/28992`.

Open Street Map Foundation. `https://www.openstreetmap.org`.

ESS (2017). ESSnet on Big Data. `https://webgate.ec.europa.eu/fpfis/mwikis/essnetbigdata/index.php`.

UNECE (2013). Generic Statistical Business Process Model v5.0. `https://statswiki.unece.org/display/GSBPM/Generic+Statistical+Business+Process+Model`.

WP5 of ESSnet on Big Data (2016). Deliverable 5.1. `https://webgate.ec.europa.eu/fpfis/mwikis/essnetbigdata/images/6/65/WP5_Deliverable_1.1.pdf`.

WP5 of ESSnet on Big Data (2017). Guidelines for the access to mobile phone data within the ESS. Deliverable 5.2. `https://webgate.ec.europa.eu/fpfis/mwikis/essnetbigdata/images/6/65/WP5.Deliverable1.2.pdf`.

WP5 of ESSnet on Big Data (2018). Some IT elements for the use of mobile phone data in the production of official statistics. Deliverable 5.4. `https://webgate.ec.europa.eu/fpfis/mwikis/essnetbigdata/images/c/ce/WP5_Deliverable_5.4_Final.pdf`.

Jerry Banks, John S. Carson II, Barry L. Nelson, David M. Nicol (2010). *Discrete-Event System Simulation (Fifth Edition)*. Pearson Education Inc., Upper Saddle River, New Jersey.

James J. Nutaro (2011). *Building Software for Simulation - Theory and Algorithms with Applications in C++*. John Wiley and Sons, Inc. Hoboken, New Jersey.