SILUMOD: A Simulation Language for User Mobility Models Definition in Multihop Networks

Yacine Benchaab
Institut Télécom / Télécom ParisTech / LTCI
CNRS UMR 5141
yacine.benchaib@telecom-paristech.fr

Claude Chaudet
Institut Télécom / Télécom ParisTech / LTCI
CNRS UMR 5141
claude.chaudet@telecom-paristech.fr

Abstract

In this paper, we present SILUMOD, an open source (LGPL) internal domain-specific language that we developed to facilitate the study and deployment of users mobility models for multihop wireless networks. Based on Scala\(^1\), SILUMOD uses the same syntax and the same libraries as this language, while adding a series of keywords and operators reflecting the common characteristics of the movements of mobile nodes. SILUMOD combines ease of use and efficiency by ensuring rapid and easy implementation of most mobility models.

Categories and Subject Descriptors I.6.2 [Simulation and Modeling]: Simulation Languages

General Terms Languages, Experimentation

Keywords Domain-specific language, Wireless multihop networks, Mobility models

1. Introduction

Research in the field of wireless multihop networks (ad-hoc, sensors, vehicular, ...) has extensively studied static networks and it is today shifting towards mobility. Among the set of tools available to the community, several experimental platforms (e.g. IBBT Wireless Lab\(^2\), SensLab\(^3\), FIT\(^4\), ...) now include mobile nodes, embarked for example on wheeled robots or on model trains. All packet-level simulators such as ns-2, ns-3, Omnet++ or Opnet handle nodes mobility and propose a set of built-in mobility models. A mobility model is the definition of how nodes move and interact with their environment (obstacles, other mobiles, etc.). Choosing the right mobility model to evaluate a protocol or an algorithm is not trivial. The mobility model should be representative and generic. It should not bias performance in a direction or the other. The coherence between modeling, simulation and experimental results should also be guaranteed and the same mobility model should be available to all the corresponding tools, which usually use different formalisms to define nodes movements. Simulators, on the one hand, often use a sequence of positions in a coordinates system, possibly mimicking a real trace. Robots, on the other hand, are often unable to localize themselves precisely and do not share the same clock. Their movements are often defined by specifying sequences of rotation angles and distances, or by controlling each wheel’s individual speed. As translation is difficult, simple random mobility models such as random waypoint, proposed in [5], or random direction proposed in [14] are often favored, even though they seldom reflect reality, as shown in [18].

In this paper, we introduce SILUMOD, a domain specific language based on Scala to describe nodes mobility in an easy and representative way. The objectives of SILUMOD are threefold:

- As it is a high-level language, researchers can express mobility without thinking about implementation. Specific tools and libraries shall perform the conversion to simulators traces or to particular robots model.
- Using similar translation techniques, mobility models can be automatically characterized and checked through formal verification tools.
- Defining a common language allows feeding a mobility models online repository. Models in this library can constitute a relevant reference for protocols comparison.

In this article, we first detail the goals behind SILUMOD in section 2. We then present related works in section 3. Section 4 extracts from the literature the characteristics of a realistic mobility model and lists a set of requirements.

\(^1\)http://www.scala-lang.org/
\(^2\)http://www.ibbt.be/en/develop-test/ilab-t/wireless-lab
\(^3\)http://www.senslab.info
\(^4\)http://fit-equipex.fr/english
2. From mobility specification to the implementation

Three categories of tools are generally used to evaluate protocols and algorithm for wireless multihop networks and their representation of nodes mobility differ considerably.

Mathematical modeling often uses statistical properties of the mobility models such as the dynamics of the connectivity graph, a graph in which edges between two vertices reflect the situation in which the two nodes are in communication range. Nodes mobility makes this graph evolve over time. Individual mobility may be characterized by the inter-contact time distribution or by the contact length distribution. Global properties are rendered by the evolution structural parameters such as the graph degree distribution or the number of connected components, for example. Such statistics, as well as other mobility models properties may be obtained on the basis of the model description by analysis. Lin et al. [11], for example, use renewal theory to characterize the properties of a few random mobility models. If such analysis is not possible or too complex, statistical properties may be obtained by extensive simulation.

Simulation tools such as Opnet Modeler\(^5\), Omnet++\(^6\), NS-2\(^7\) or GlomoSim / QualNet\(^8\) expect nodes movements to be represented as a series of (time, position) couples. Listing 1 represents an example of definition of a node mobility in the NS-2 simulator. The node with ID 3 starts at date \(t = 10\)s to move linearly towards the position \((x = 50.0; y = 50.0)\) at a speed of \(10\) m/s, then turns at date \(t = 20\)s towards a second position. These waypoints can be defined randomly by internal code or by an external tool, or can be extracted from a real trace. Between two waypoints, which can be arbitrarily close, the node is usually supposed to move linearly at a given speed. Most simulators implement natively the random waypoint and random direction models. However, several contributions such as [1] point that random models do not correspond to reality and bias performance evaluation results. The generation of a list of timed positions is immediate from a high-level specification of a mobility model.

Experimental platforms that use mobile robots usually rely on languages like which allow specifying a series of movements in terms of positions, direction changes, speed etc. depending on the physical robot capacities. A robot capable of self-localization will be able to move between way-

\[\text{Listing 1: Trajectory definition example in NS-2 simulator}\]

```
$ns_ at 10.0 "\$node_(3) setdest 50.0 50.0 10.0"
$ns_ at 20.0 "\$node_(3) setdest 70.0 150.0 3.0"
```

4http://www.ometpp.org/
8http://www.isi.edu/nsnam/ns/

3http://pcl.cs.ucla.edu/projects/glomosim/

points, but robots that are not equipped with localization devices move by defining linear and angular speed, or by controlling the rotation speed of each individual wheel. Translating a high-level mobility specification into such an atomic language may require more or less complex compilation, but this translation can be provided for each robot type once and for all.

The objective of our work is therefore to provide a high-level description language for mobility that is easy enough to hide implementation capabilities and expressive enough to allow complex patterns specification. This language should allow, through a set of tools, to characterize and verify formally properties of the models and to instantiate a run of the model as a list of positions for a simulator or a robot of self-localization, or as a list of directives that define the behavior of a robot.

3. Related Work

The literature is filled with numerous mobility models that correspond to a scenario or another. However, few articles aimed at providing a generic way to specify nodes mobility.

Stepanov et al. [16] propose a generic model that decomposes users movement definition in three components. First, the environment, composed of obstacles, points of interest, etc., defines the scene in which mobiles evolve and imposes constraints on the movements. Environment can be extracted from a geographic database. Second, the users behavior is defined by their activities. Users move when they are on a trip, going from one place to another. Third, the movement dynamics (speed, acceleration, ...) define how the users move. The authors propose an XML format to allow users to specify these three components. A java-based tool (Canu-MobiSim) then translates this XML specification to the NS-2, Glomosim and QualNet formats. Authors provide a few random, vehicular and physics-based models as examples.

Robsys ([17]) focuses on mobile robots programming and proposes a hierarchical language to specify the movement of a robots fleet. The high-level, object-oriented language is dedicated to the specification of multiple robots (agents) behavior, providing ways to express synchronization, communication, etc. while the lower-level language is closer to assembly language and dedicated at controlling one single robot. Mobility is expressed as a set of instructions for a robot (moving to a position, advancing of a few meters, turning, etc.).

Finally, the Condition Probability Event (CPE) model [12] proposes to represent pedestrians mobility as a set of rules that mix probabilities and interactions with the environment.
A movement is described as a series of rules composed of a condition (trigger) that can include the node’s environment (time, obstacles, ...), a probability, an action and a set of consequences on a node’s internal state variables (position, direction, speed, etc.). A tool converts this set of rules to Qualnet and Opnet models.

These three contributions are either linked to the mobile ad-hoc networks field, or to the robotics field and none of them truly provides an interface between these two worlds, even though such a connection is evoked. MANET/VANET tools usually target simulation tools and suppose that mobility can be specified by a series of positions, which supposes that nodes are able to localize themselves accurately. On the other hand, Robsys identifies the need for agents collaboration but the high-level language is limited regarding interaction with the nodes environment. The language we introduce in this paper aims at filling the gap between these communities through the definition of a high level language that can be compiled or translated for each target while keeping the same properties.

4. Components of a realistic mobility model

In the fields of mobile ad-hoc networks (MANET) and vehicular ad-hoc networks (VANET), many contributions aimed to create mobility models that were more representative of real situations than the classical random models. However, these articles seldom try to define a general formalism to express mobility and rather focus on one scenario or one category of users.

4.1 Modeling the scene: boundaries and objects

RMobiGen ([2]) defines mobility of nodes as a set of state variables (speed, direction, distance to travel, duration, destination) and proposes few alternatives for the scenario when a node reaches the scene boundaries. Nodes may bounce on this boundary, wrap around or disappear. Then, the traces of movement can be exported as a data format usable on NS-2.

Similarly, in the fields of robotics, Arkin et al. [3] define mobility either by a series of waypoints coordinates, or by letting robots follow a path in a corridor. Mobiles may detect movement can be exported as a data format usable on NS-2.

The presence of obstacles is also one of the key characteristics of realistic mobility models. Jardosh et al. [10], for instance, organize nodes movement around a set of obstacles using Voronoi diagrams built with the help of the objects corners. The mobility model it then used in GlomoSim to evaluate the performance of an ad-hoc routing protocol.

In vehicular networks (VANETs), nodes movements are constrained on a city map or a map extracted from a Geographic Information Service (GIS), which is not more than an extended area mostly filled with obstacles. Saha et al. [15], for example, use the TIGER (Topologically Integrated Geographic Encoding and Referencing) streets database to map users mobility on the Houston city map. The authors then use NS-2 to evaluate the performance of another ad-hoc routing protocol. Härri et al. [7] propose to enhance VANET simulation realism by taking into account the streets map and by using an intelligent driver model. Their simulator VanetMobiSim, extends CanuMobiSim and thus uses the same XML data format. MobiREAL ([12]) extends the CPE model to take into account the current status (position, direction, speed, destination) and the environment (time, global information, etc.) of a node over a street map.

4.2 Nodes behavior: group mobility and attractors

Hong et al. [9] wrote one of the first contributions to stress out the importance of group mobility. The movement patterns of individual users and devices often depend on other nodes behaviors. Musolesi et al.[13] define a group mobility model based on social interactions, using the strength of the connections between users to form an evolving topological space.

GEMM ([6]) proposes to modify the random waypoint mobility model by defining particular locations, which may act as attraction points. Users are then grouped into classes who are attracted. The mobility pattern defined this way is stored in the BonnMotion format, which can be used in NS-2 or GlomoSim. Hollick et al.[8] follow the same logic of activities and attraction to model metropolitan area networks. Their mobility model is used on a modified NS-2. Aravind et al. [1] use the attraction and repulsion points concept to define the behavior of a vehicular user that travels on a city map when reaching an intersection.

5. SILUMOD

In the previous section, we have compiled a set of essential characteristics to the definition of a mobility model for multihop networks. A mobility model should allow defining precise trajectories, through a set of successive positions (waypoints) or it should be able to reflect how a node’s behavior is influenced by its environment. The environment is composed of objects, has (artificial) boundaries, and may include fixed or mobile attraction and repulsion points. Group mobility can be defined through the definition and correct weighting of such attraction points. In this section, we detail the first release of SILUMOD that includes environment boundaries handling and objects. Attractors will be implemented in a second release.

SILUMOD is an internal domain specific language that uses Scala as a host language. We chose to rely on Scala as it provides an easy way to define new operators, and not only to supersede existing operators, which results in an infix intuitive and expressive syntax. Scala is a language that supports imperative programing and is therefore more intuitive for classical users than OCaml or Haskell.

SILUMOD is therefore based on the same grammar as Scala. It is implemented through an external package that is
imported simply by including SILUMOD in the list of locations where libraries are found (classpath). All programs written in SILUMOD are therefore executed through the regular Scala interpreter with minor configuration. SILUMOD benefits from the structures and keywords defined by Scala without re-writing a new compiler or interpreter. SILUMOD can therefore be extended easily, using all the expressiveness of Scala. It also allows us to focus on the implementation of the domain-specific constructions while ensuring that the user will have a good level of flexibility when defining new mobility models.

5.1 Defining the environment
Mobile node moves on a limited surface on which obstacles may be defined and positioned. The coordinates range is dynamic and user can thus represent at least a limited area which could be similar to a room in an apartment or more complex, a neighborhood in a city with these streets, these buildings, etc. Figure 1 represents an example of a rectangular environment in which 6 wall objects are positioned. The corresponding definition in SILUMOD is represented by the listing 2.

![Figure 1. A rectangular playground with 6 inner walls](image)

<table>
<thead>
<tr>
<th>Listing 2: Example of definition of a scene boundaries and objects</th>
</tr>
</thead>
<tbody>
<tr>
<td>playground define(0,0,600,400,“PG”,“black”,“no”) block define(100,70,500,80,“BK1”,“red”,“yes”) block define(400,70,410,250,“BK2”,“red”,“yes”) block define(320,320,500,330,“BK3”,“yellow”,“yes”) block define(220,250,230,400,“BK4”,“pink”,“yes”) block define(100,160,310,170,“BK5”,“purple”,“yes”) block define(100,160,110,340,“BK6”,“purple”,“yes”)</td>
</tr>
</tbody>
</table>

The `playground` keyword is used to define the limits of the experiment. The `block` keyword positions, on the scene, a rectangle object. The parameters of this function are the coordinates of the top-left corner and of the bottom-down one. It therefore allows placing horizontal or vertical rectangles. The two last parameters are for display only purposes and are color and filling information.

These two constructs are enough to represent indoor scenario and a few urban scenarios. For example, it is relatively easy to convert OpenStreetMap topological data structure, which represents a city as a set of polygons, in such formalism. Such a conversion tool would allow accessing a large database of VANET scenarios. As of today, SILUMOD only allows defining rectangles, which can be enough approximate complex environments, but future releases could include multi-edges polygons, should the need for high-resolution maps appear.

5.2 Defining interactions with the environment
Moving nodes regularly encounter an object in their environment. They then modify their trajectory according to their objective, or to behavioral rules. These behaviors can be arbitrarily complex, thanks to the Scala basis of SILUMOD.

In SILUMOD, we distinguish two types of obstacles: the playground limits and the objects inside the playground, as the behavior of a node may be slightly different in both situations. When reaching the zone boundaries, a node may either bounce back, as if it were in a building, or disappear, or even reappear on the opposite side of the playground, which then becomes a sphere. Inner objects are obstacles (`block`) and the usual behavior of the mobiles, when reaching such a block, is either to bounce back, or to try to pass around the obstacle.

This defines three types of interacting actor that SILUMOD represents by the objects `MOBILE`, `WALL` and `LIMITS`. These keywords are part of the language and do not need to be defined by the user.

A dedicated operator is defined in SILUMOD to express the interaction between actors, `MOBILE` with `WALL` and `MOBILE` with `LIMITS`. The `->|` operator returns `true` when the mobile reach a wall or the limits and returns `false` otherwise. Examples of use are presented in section 6.

5.3 Specifying nodes movements
SILUMOD allows defining how a node moves in two ways that will be illustrated in section 6.

The first way is to use the characteristics of the movement of a mobile. With reference to the dependency rules such as \( \text{speed} = \frac{\text{distance}}{\text{duration}} \) for example, the user specifies a number of parameters – for example distance and date, or speed and date – and SILUMOD determines the points constituting the path until it reaches the distance or time required.

The keywords that can be used to define these parameters are `angle`, `speed`, `distance` and `duration`. A new value can be defined for each parameter by associating these keywords with the keyword `value`. The parameter associated
with value can be fixed or drawn randomly. For random number generation, SILUMOD provides the <-> operator that returns a value drawn uniformly between two bounds (e.g. 10 <-> 100).

The second way is to act directly on the position of the mobile. A node’s position can be defined at a given time with the PositionX and PositionY keywords. In this case, the node will "jump" instantaneously to the specified position. Alternatively, the user can specify a position to reach with toPositionX and toPositionY. In the second case, SILUMOD then calculates the various components of a realistic movement (angle, distance, ...) and user-defined parameters.

### 5.4 Output-specific constructions

The purpose of SILUMOD is to be instantiated at various levels. As of today, the mobility models expressed in this language can be visualized with a graphical user interface and used by a network emulator, defining the connections capabilities over time in virtualized network environment. Both environments have their specificities and SILUMOD allows expressing these specificities through a set of keywords that are ignored when they are not relevant for a given output. This approach can be generalized for any output.

#### 5.4.1 Network emulation

SILUMOD was developed in conjunction with VIRMANEL [4], a tool that combines virtual machines with firewall rules to emulate multihop wireless networks. Firewall rules are activated and de-activated dynamically to reflect connectivity changes between the nodes, which allows emulating a whole mobile network. Users have to specify a node’s regular IP address, which allows it to communicate with other nodes, and a management IP address, used by SILUMOD to access the node at all times. Listing 3 represents such parameters definition for a given node. The two IP addresses are defined for the node named "Mobile1", which will result in the creation of two distinct network interfaces in the virtual machine.

The next parameter defines the communication range of the node that is passed to VIRMANEL, which in turn decides to activate and de-activate firewall rules. This communication range can be dynamically updated by the user to reflect, for example, a decision to reduce transmission power when batteries become scarce.

```plaintext
name value "Mobile1"
mgmtIp value "172.16.0.1"
nodeIp value "192.168.1.1"
coverageRadius value 200
```

Listing 3: Definition of parameters specific to network emulation

Once the mobility model is passed to VIRMANEL, the user gains full control of the virtual machines that correspond to the mobile nodes. He can install software on these machines that run, in this version, a light distribution of Linux. For example, the user interested in evaluating the behavior of MANET routing protocols under various mobility patterns can install the real OLSR code included in the Debian Linux distribution, run a real benchmark application such as netperf or simply ping, and compare different scenarios, varying movement speeds, network connectivity, etc. The virtualization tools cannot be used to evaluate the true delay, as the physical layer is emulated and hence not realistic, but the algorithmic behavior of the protocol and its reaction to mobility can be evaluated precisely. Implementations can also be compared in terms of memory or CPU usage, for example, which is not possible with classical network simulators.

#### 5.4.2 Graphical User Interface interaction

The trajectory and the connection capabilities of a mobile node can be represented on the Graphical User Interface of VIRMANEL. This allows understanding and easily adjusting the algorithm of mobility used, provided that the user is able to monitor particular nodes or links. In order to distinguish certain nodes, link or categories, SILUMOD allows the user to specify color information. A color change of the trajectory can also be used to highlight the different phases of a movement like acceleration, particular speed, modification of particular values, etc. Listing 4 represents the definition of such parameters. The first two lines define that trajectories shall be displayed with red lines, and the next ones that radio connections should be displayed and colored in blue.

```plaintext
trace show "yes"
trace color "red"
connection show "yes"
connection color "blue"
```

Listing 4: Definition of parameters specific to graphical visualization

### 6. Examples

For emulation, we use VIRMANEL, a network emulator that we developed through our research on multihop mobile networks. In addition to managing nodes pairs connectivity, VIRMANEL receives and displays the cartesian coordinates of every node on its Graphic User Interface with a dedicated connection. In this section, we present the expression of two mobility models in SILUMOD, and the resulting display in VIRMANEL GUI.

#### 6.1 Random Walk Mobility Model

##### 6.1.1 Definition

Random walk is a very simple mobility model in which all mobiles have the same behavior: each node chooses a ran-
dom speed and a random movement direction in coherence with the scene limits. It then moves until it has traveled a predefined distance and then chooses a new speed and angle. When the mobile node encounters an obstacle, it bounces back, changing its angle to perform a reflection on the surface.

6.1.2 Implementation in SILUMOD

The expression of this mobility model in SILUMOD is represented on listing 5. After defining the (single) node’s initial position and the step distance, the script generates 100 movements.

```java
import silumod.language._

playground define(0,0,600,400,"PG","blue","no")

var x = 0
positionX value 100
positionY value 200
distance value 150

while ( x < 100 ) {
    speed value ( 10 <-> 40 )
    angle value ( 0 <-> 360 );

    while ( MOBILE calculateNewPosition ) {
        if ((MOBILE ->| LIMITS) || (MOBILE ->| WALL))
            angle reflexive
        else { MOBILE Move }
        
        x = x + 1
    }
}
```

Listing 5: Specification of the Random Walk Mobility Model in SILUMOD

Figure 2. A trace of the Random Walk Mobility Model

Each movement starts by drawing randomly an angle and a speed and the mobile then starts moving, small step by small step, until the next iteration or until it reaches a wall or the scene limits. The trajectory of figure 2 results from a run of this model in VIRMANEL. The produced trajectory is composed of 100 segments of similar length, separated by sudden turns.

6.2 Restricted Random Waypoint Mobility Model

6.2.1 Definition

This mobility model aims at recreating the movements of a mobile node between several points of interest represented by cities in the example. The mobile node moves within a city by choosing successive random destination points (Random Waypoint Mobility Model). After twenty movements, it moves to the next city. Rules when encountering an obstacle are the same as for the Random Walk Mobility Model except when the mobile node move from one city to another. In this case, the mobile node does not bounce when it encounters the limits of the current and the next city but it is allowed to cross these limits.

6.2.2 Implementation in SILUMOD

Listing 6 represents the implementation of this model in SILUMOD. After defining graphical objects (the playground, blocks representing the walls and cities boundaries), the (single) mobile starts at a predefined position and makes 100 movements, changing cities every 20 steps.

For each city a destination (x,y) coordinates are drawn randomly in the pre-defined interval and the mobile moves at the pre-defined speed towards this destination before passing to the next step.

The resulting movement trace is represented on Figure 3. We can notice that the mobiles tend to remain in the center of each city, which is classical with random waypoint mobility, and that SILUMOD only takes into account cities boundaries when a node is not supposed to move between cities.

Figure 3. A trace of the Restricted Random Waypoint
import silumod.language._

playground define(0,0,600,400,"PG","blue","no")
block define(30,30,200,200,"City1","black","no")
block define(300,30,570,200,"City2","red","no")
block define(100,250,500,350,"City3","blue","no")

var x = 0
var numberWayPointChoice = 0
var city = 1
var MovetoAnotherCity:Boolean = false
var exitPreviousCity:Boolean = false
positionX value 55
positionY value 55
speed value 50

while ( x < 100 ) {
    numberWayPointChoice = numberWayPointChoice + 1
    MovetoAnotherCity = false
    if ( numberWayPointChoice % 20 == 0 ) {
        city = ( city % 3 ) + 1
        MovetoAnotherCity = true
    }
    if ( city == 1 ) {
        toPositionX value ( 31 <-> 199 )
        toPositionY value ( 31 <-> 199 )
    }
    if ( city == 2 ) {
        toPositionX value ( 301 <-> 569 )
        toPositionY value ( 31 <-> 199 )
    }
    if ( city == 3 ) {
        toPositionX value ( 101 <-> 499 )
        toPositionY value ( 251 <-> 349 )
    }
    while ( MOBILE calculateNewPosition ) {
        if ( ((MOBILE ->| LIMITS) || (MOBILE ->| WALL))
            && ( MovetoAnotherCity == false ) )
            { angle reflexive }
        else { MOBILE Move }
    }
    x = x + 1
}

Listing 6: Specification of the Restricted Random Waypoint Mobility Model in SILUMOD

7. Discussion

It is possible to define as many mobility models for multihop networks as the human brain is able to imagine. Determining the basic common features valid for all models of mobility is not an easy task. However, the exercise is really beneficial.

Rather than change each time a source code to fit the new model under study, even a novice user can easily program his own mobility model in few minutes. A SILUMOD program is written in the same way as a real description of the mobility model. The limited set of added keywords and the natural structure of Scala make SILUMOD easy to learn and to use.

Since the beginning of its conception, SILUMOD was designed to evolve. Other keywords and operators shall be added in the same way as the current ones, without going through a phase change in the compiler. We will first try to implement as many mobility models as possible and include progressively concepts such as attractors. One of the first operators that we intend to add is the @^@ operator that allows distinguishing between a node’s perception (vision) and its real state.

8. Conclusion

In this paper, we introduced SILUMOD, a domain specific language dedicated to the definition of mobility models for multihops network. SILUMOD provides a quick and easy way to specify mobility models. Its compatibility with the Scala language provides the ability to use many libraries, paradigms and keywords specific to Scala. A specification written in SILUMOD can then be translated into several formalisms, including simulation languages and robots control.

Acknowledgments

This work is supported in part by the FIT project (http://fit-equipex.fr/) granted by the French National Research Agency (ANR) and by the FITTING project funded by EIT ICT Labs.

References


