



Epigrass Documentation

Release 2.0a1

Flavio Codeco Coelho

July 09, 2008

CONTENTS

1 Overview of Epigrass	3
1.1 Modeling Approach	3
1.2 Geographical Network Models	3
1.3 The Simulation	5
2 Building and Installing	7
2.1 Required Packages	7
3 Using Epigrass	11
3.1 Data	11
3.2 Specifying a model: the script	13
3.3 The Graphical User Interface(GUI)	17
4 Writing Custom Models	23
4.1 Getting Started	23
4.2 The Environment	25
5 Indices and tables	29
Index	31

Epigrass is a framework for the construction and simulation of complex network epidemiology models. At least this is its main vocation. However, Epigrass can be used to simulate any dynamical system on a set of nodes (connected into a network or not!). Epigrass is written in pure Python and is scriptable in Python, for your enjoyment!.

Contents:

Overview of Epigrass

Epigrass is a platform for network epidemiological simulation and analysis. It enables researchers to perform comprehensive spatio-temporal simulations incorporating epidemiological data and models for disease transmission and control in order to create sophisticated scenario analyses.

1.1 Modeling Approach

The geographical networks over which epidemiological processes take place can be very straightforwardly represented in a object-oriented framework. In such a framework, the nodes and edges of the geographical networks are objects with their attributes and methods.

Once the archetypal node and edge objects are defined with appropriate attributes and methods, then a code representation of the real system can be constructed, where cities (or other geographical localities) and transportation routes are instances of the node and edge objects, respectively. The whole network is also an object with a whole collection of attributes and methods.

This framework leads to a compact and hierarchical computational model consisting of a network object containing a variable number of node and edge objects. This framework also do not pose limitations to encapsulation, potentially allowing for networks within networks if desirable.

For the end user perspective, this framework is transparent since it mimics the natural structure of the real system. Even after the model is converted into a code object all of its component objects remain accessible to the user's models.

1.2 Geographical Network Models

Epigrass's geo-referenced models are built from two basic sources of data: a map (in shapefile format) which provide the cartographical base over which the models are represented and specific data about nodes and edges that are provided by the user for the network of interest.

1.2.1 Defining the Cartographic Background

If the user has a map for the georeferred data, this can be passed to Epigrass. In this case, the cartographic background is defined by defining the name of the shapefile file (with path relative to the working directory) in the model'.epg file. Along with the path to the shapefile, the variable in the shapefile, which contains the geocode of localities and their name must also be specified:

```
shapefile = ['riozonas_LatLong.shp', 'nome_zonas', 'zona_trafe']
```

If the user does not have a map in shapefile format, he can still use Epigrass. In this case, the georeferred data is read only from two .csv files (more on that ahead).

1.2.2 Defining Nodes

A graph has nodes and edges. Nodes can be cities or other localities depending on the model being constructed. The definition of nodes require the setting of many attributes listed below. The nodes will have many more attributes defined at run-time which will depend on other aspects of the model, these will be discussed later.

The data necessary at build time to create nodes come from a CSV (comma-separated-values) ASCII-text file, with the following attributes, (in this order):

Latitude, Longitude This attribute will be used to geo reference the node. The coordinate system must match those used in the cartographic base imported from GRASS. Coordinates can be coded in either decimal or sexagesimal format.

Name Used for identification purposes only. It can be a city name, for instance.

Population Since the simulation models will all be of a populational nature. Population size must be specified at build time.

Geocode A Unique Geocode (a number) is required. It will be used as a label/index to facilitate reference to specific nodes.

1.2.3 Defining Edges

In Epigrass' graphs, edges represent transportation routes. Similarly to nodes, edges are defined at build-time with a reduced set of attributes which will be extended at run-time. Epigrass also expects to get these attributes from a CSV file:

Source The name of the source node. The edges are bi-directional, but the nodes are labeled source and destination for reference purposes.

Destination The name of the destination node.

Forward migration Migration rate from source to destination, in number of persons per unit of time.

Backward migration Migration rate from destination to source, in number of persons per unit of time.

Length Distance in kilometers (or another unit) from source to destination via the particular route (not straight line distance).

Source's geocode This is the unique numerical identifier used in the sites file.

Destination's geocode This is the unique numerical identifier used in the sites file.

1.2.4 Defining models

The word model in Epigrass can mean two distinct objects: The network model and the node's epidemic model.

Node objects, in an Epigrass model, contain well-mixed population dynamic models within them. These models determine the dynamics of epidemics within the particular environments of each node. Epigrass comes with a few standard epidemiological models `index{Models!epidemiological models}` to choose from when setting up your network. Currently, The same model type is applied to every node although their parameterization is node-specific. Besides the built-in model types, users can define their own, as shown in the chapter *Using Epigrass*. Network models are specified in a ASCII-text script file (Called a `.epg` file). Epigrass comes with a few demo Network models for the user to play with until he/she is confident enough to build their own. Even then, it is advisable to use the demo scripts provided as templates to minimize syntax errors.

The script on the appendix specifies a network model with an stochastic SEIR (see chapter on epidemiological modeling) epidemic model in its nodes. The user should study this model and play with its parameters to understand the features of Epigrass. A step-by-step tutorial on how to edit the model script can be found in the chapter *Using Epigrass*.

1.3 The Simulation

A simulation run in Epigrass consists of a series of tasks performed at each time step ¹.

Calculate migration For all edges in the network, the number of persons traveling each way is determined for the current time-step.

Run epidemic models For each node in the network the epidemic demographics are updated based on the local number of infected and susceptible individuals which have been updated by the transportation system.

All aspects of the simulation such as number of passengers traveling on each edge, number of infected/susceptible on each node and etc., are recorded in a step-by-step basis. This complete record allows for the model to be analyzed after the simulation has been completed without having to recalculate it.

1.3.1 Output

The output of a simulation in Epigrass is three-fold: A graphical display which the animated outcome of the simulation, a written report, and a database table with numeric results.

Graphical display

During a simulation, selected epidemiological variables are animated in a 3-dimensional rendering over the map of the region containing the network.

Report Generation

The report contains a detailed analysis of the network model and the simulations ran with it. The report generates a LaTeX source file and compiles it to a PDF document for visualization.

Three types of report are currently available:

Report = 1 Returns a set of descriptors of the network, described in chapter

Report = 2 Returns a set of basic epidemiological measures and plots of the time series.

Report = 3 Report 1 + Report 2

Report Generation is an optional, though recommended, step done at the end of the simulation. For the report, descriptive statistics are generated for the network. These have to do with network topology and properties. Additional sections can be added to the report with basic statistical analyses of the output of pre-selected nodes ².

Database output

Time series of **L**, **S**, **E**, and **I**, from simulations, are stored in a MySQL database named *epigrass*. The results of each individual simulation is stored in a different table named after the model's script name, the date and time the simulation has been run. For instance, suppose you run a simulation of a model stored in a file named `script.epg`, then at the end of the simulation, a new table in the epigrass database will be created with the following name: `script_Wed_Jan_26_154411_2005`. Thus, the results of multiple runs from the same model get stored independently.

Epigrass also supports the SQLite database and CSV files as output for the time-series. The naming convention also applies to these other formats.

¹The number of time steps is defined in the model script

²Listed in the siteRep variable at the script

Building and Installing

This chapter will walk through all aspects of Epigrass installation. From obtaining, building and installing the prerequisites to the installation of Epigrass itself.

Most of the steps will be quite simple and similar since they will make use of standard tools for package installation on Debian GNU/Linux and derivatives. If you use a different distribution, you should check its documentation for package installation instructions.

If, on your distribution, a package is not available for the required version, you can try to obtain an updated version of the package at the web-sites provided. On the rare cases where pre-built packages are not available, instructions on how to build the software from source should also be available from its web-site.

2.1 Required Packages

2.1.1 Python

Web-site: <http://www.python.org>

Version required: ≥ 2.5

Python is a simple yet powerful object-oriented language. Its simplicity makes it easy to learn, but its power means that large and complex applications can also be created easily. Its interpreted nature means that Python programmers are very productive because there is no edit/compile/link/run development cycle.

Python is probably installed automatically by your GNU/Linux distribution (it is on Debian). If not, it is best to use your distribution's standard tools for package installation. On Debian for example:

```
$ sudo apt-get install python python-dev
```

2.1.2 Numeric Python

Web-site: <http://www.scipy.org/numpy>

Version required: ≥ 1.0

Numeric Python is a module for fast numeric computations in Python.

Example installation:

```
$ sudo apt-get install python-numpy
```

2.1.3 Matplotlib

Web-site: <http://matplotlib.sourceforge.net>

Version required: >0.9

Matplotlib is a Module that provides plotting capabilities to Python.

```
$ sudo apt-get install python-matplotlib python-matplotlib-data python-matplotlib-doc
```

2.1.4 PyQt

Web-site: <http://www.riverbankcomputing.co.uk/pyqt>

Version required: >4.0

PyQt is a set of Python bindings for the Qt toolkit. PyQt combines all the advantages of Qt and Python. A programmer has all the power of Qt, but is able to exploit it with the simplicity of Python.

PyQt depends on the Qt libraries to run. This dependency will be taken care by the package installation tools of most distributions, which will automatically install the required version of Qt.

Example installation:

```
$ sudo apt-get install python-qt4
```

2.1.5 PyQwt

Web-site: <http://pyqwt.sourceforge.net/>

Version required: >5.0

PyQwt is a Python binding for the Qwt5 technical widget library. It is necessary for some of Epigrass' visualization capabilities.

Example installation:

```
$ sudo apt-get install python-qwt5-qt4
```

2.1.6 MySQL

Web-site: <http://www.mysql.com>

Version required: >5.0

MySQL is a fast, multi-threaded, multi-user SQL database server. If you have a MySQL server available in your LAN, you may skip this step after making sure you have permission to access and use it to store your data.

Example installation:

```
$ sudo apt-get install mysql-server mysql-client python-mysqldb
```

Post-install configuration

After installing MySQL, you will need to create a new database in the server, called *epigrass* and a user with all privileges to access and modify it. This is the user Epigrass will use to interact with MySQL.

2.1.7 R

Web-site: <http://www.r-project.org>

Version required: The most recent.

R is a statistical computing platform whis is very useful to analyze the output of Epirass's models. The Epigrass source distribution includes some example **R** scripts to interact with Epigrass-generated data.

Example installations:

```
$ apt-get install r-base
```

After installing R, start an interactive session, and install a few extra packages.

```
$ R
> update.packages()
> install.packages('RMySQL')
> install.packages('DBI')
> install.packages('lattice')
```

section{Installing Epigrass} If you got through all the steps above, it will be an easy task to install Epigrass. There is a *.deb* package for installing Epirass on Debian and Ubuntu. However, since it is not maitained by the developers of Epigrass, It may very well be outdated. So please download Epigrass's source tarball (something named `Epigrass_someversion.tar.gz`) from Sourceforge and, after unpacking it to some temporary directory, install it by typing:

```
$ sudo python setup.py install
```

If the installation proceeds without errors, you will have three new executables available on your system:

epigrass This starts Epigrass graphical user interface (GUI).

epirunner This is a command line version of Epigrass. With it you can run models without invoking the GUI. It's great for batch simulations and for remote use. for a quick help, try "epirunner -h".

epgeditor A graphical editor of .epg models. A easy way to edit already existing models. Contains detailed explanations of every section of the EPG file. ***:math:'x_3+y'***

Unknown interpreted text role "math".

Using Epigrass

To simulate an epidemic process in Epigrass, the user needs to have in hand at least three files: Two files containing the site and edge data and a third file which is a script that defines what it is to be done. Here we go through each one of them in detail. The last part of this chapter is a step-by step guide the Graphical User Interface (GUI).

3.1 Data

3.1.1 Site data file

See below an example of the content of a site file for a network of 10 cities. Each line corresponds to a site (except the first line which is the title). For each site, it is declared, in this order: its *spatial location* in the form of a pair of coordinates ($[X,Y]$); a site $\$name\$$ to be used in the output; the site's population; the site geocode (an arbitrary unique number which is used internally by Epigrass).

X	Y	City	Pop	Geocode
1	4	"N1"	1000000	1
2	4	"N2"	100000	2
3	4	"N3"	1000	3
4	4	"N4"	1000	4
5	4	"N5"	1000	5
1	3	"N6"	100000	6
2	3	"N7"	1000	7
3	3	"N8"	100000	8
4	3	"N9"	100000	9
5	3	"N10"	1000	10
1	2	"N11"	1000	11

In this example, the first site is located at $[X,Y]=[1,4]$, it is named N1, its population is 1000000 and its geocode is 1. This is minimum configuration of a site data file and it must contains this information in exactly this order.

In some situations, the user may want to add other attributes to the sites (different transmission parameters, or vaccine coverage or initial conditions for simulations). This information is provided by adding new columns to the minimum file. For example, if one wishes to add information on the vaccine coverage in cities N1 to N10 ($\$vac\$$) as well as information about average temperature (which hypothetically affects the transmission of the disease), the file becomes:

X	Y	City	Pop	Geocode	Vac	Temp
1	4	“N1”	1000000	1	0.9	32
2	4	“N2”	100000	2	0.88	29
3	4	“N3”	1000	3	0.7	25
4	4	“N4”	1000	4	0.2	34
5	4	“N5”	1000	5	0	26
1	3	“N6”	100000	6	0	27
2	3	“N7”	1000	7	0	31
3	3	“N8”	100000	8	0	30
4	3	“N9”	100000	9	0	24
5	3	“N10”	1000	10	0	31

During the simulation, each site object receives these informations and store them in appropriate variables that can be used later during model specification. Population is stored in the variable N ; while the extra columns (those beyond the geocode) are stored in a tuple named $\{values\}$. For example, for the city $N1$, we have $N = 1000000$ and $values=[0.9,32]$. During model specification, we may use N to indicate the population size and/or we can use $values[0]$ to indicate the level of vaccination of that city and $values[1]$ to indicate the temperature.

It is up to the user, to know what means the elements of the tuple $values$. Note that the first element of the tuple has index 0, the second one has index 1 and so on.

When using real data, one may wish to use actual geocodes and coordinates. For example, for a network of Brazilian cities, one may build the following file:

latitude	longitude	local	pop	geocode
-16:19:41	-48:57:10	ANAPOLIS	280164	520110805
-10:54:32	-37:04:03	ARACAJU	461534	280030805
-21:12:27	-50:26:24	ARACATUBA	164449	350280405
-18:38:44	-48:11:36	ARAGUARI	92748	310350405
-21:13:17	-43:46:12	BARBACENA	103669	310560805
-22:32:53	-44:10:30	BARRA_MANSA	165134	330040705
-20:33:11	-48:34:11	BARRETOS	98860	350550005
-26:54:55	-49:04:15	BLUMENAU	241943	420240405
-22:57:09	-46:32:30	B.PAULISTA	111091	350760505

In this file, the coordinates are the actual geographical latitude and longitude coordinates. This information is important when using Epigrass with a map in shapefile format. The geocode is also the official geocode of these localities (the same one used in the shapefile). Despite the cumbersome size of the number, it may be worth using it because demographic official databases are often linked by this number.

3.1.2 Edge data file

The edge data file contains all the direct links between sites. Each line in the file (except for the first, which is the header) corresponds to an edge. For each edge (or link) one must specify (in this order): the *names of the sites* connected by that edge; the *number of individuals traveling from source to destination*; the *number of individuals travelling from destination to source* per time step; the *distance or length* of the edge. At last, the file must contain, in the fifth and sixth columns, the *geocodes* of the source and destination sites*. This is very important as the graph is built internally connecting sites through edges and this is done based on geocode info.

Warning: It is required that the order of columns is kept the same.

See below the list of the 8 edges connecting the sites $N1$ to $N10$. Let’s look the first one, as an example. It links $N1$ to $N2$. Through this link passes 11 individuals backwards and forwards per time step (a day, for example). This edge has length 1 (remember that $N1$ is at $[X,Y]=[1,1]$ and $N2$ is at $[X,Y]=[1,2]$, so the distance between them is 1). The last two columns show the geocode of $N1$ (geocode 1) and the geocode of $N2$ (geocode 2).

Source	Dest	flowSD	flowDS	Distance	geoSource	geoDest
N1	N2	11	11	1	1	2
N2	N4	0.02	0.02	1	3	4
N3	N8	1.01	1.01	1	3	8
N4	N9	1.01	1.01	1	4	9
N5	N10	0.02	0.02	1	5	10
N6	N5	1.01	1.01	1	7	8
N7	N10	1.01	1.01	1	7	8
N9	N10	1.01	1.01	1	9	10

Note that it doesn't matter which site is considered a Source and which one is considered a Destination. I.e., if there is a link between *A* and *B*, one may either named *A* as source and *B* as destination, or the other way around.

If the edge represents a road or a river, one may use the actual metric distance as length. If the edge links arbitrary localities, one may opt to use euclidean distance, calculated from the *x* and *y* coordinates (using Pythagoras theorem).

3.2 Specifying a model: the script

Once the user has specified the two data files, the next step is to define the model to be executed. This is done in the `.epg` script file. The `.epg` script is a text file and can be edited with any editor (not word processor). This script must be prepared with care.

The best way to write down your own `.epg` is to edit an already existing `.epg` file. So, open Epigrass, choose an `.epg` file and click on the **Edit** button. Your favorite editor will open and you can start editing. Don't forget to save it as a new file in your working directory. Of course, there is an infinite number of possibilities regarding the elaboration of the script. It all depends on the goals of the user.

Note: Another way to edit an `.epg` file is to open it with the graphical editor provided with Epigrass. Just type `'epgeditor yourmodel.epg'`.

For the beginner, we suggest him/her to take a look at the `.epg` files in the demo directory. They are all commented and may help the user in getting used with Epigrass language and capabilities.

Some hints to be successful when editing your `.epg`:

- All comments in the script are preceded by the symbol `#`. These comments may be edited by the user as he/she wishes and new lines may be added at will. Don't forget, however, to place the symbol `#` in every line corresponding to a comment.
- The script is divided into a few parts. These parts have capital letter titles within brackets. Don't touch them!
- Don't remove any line that is *not* a comment. See below how to appropriately edit these command lines.

Let's take a look now at each part of a script (this is the script `.epg` demo file):

Warning: All variables defined in a `.epg` are **case-insensitive**. Consider this fact when naming your model's variables.

3.2.1 Part 1: THE WORLD

The first section of the script is titled: THE WORLD. An example of its content is shown:

```
shapefile = ['riozonas_LatLong.shp', 'nome_zonas', 'zona_trafe']
edges = edges.csv
sites = sites.csv
encoding =
```

where,

shapefile Is a list with 3 elements: the first is the path, relative to the working directory, of the shapefile file; the second is the variable, in the shapefile, which contains the names of the localities (polygons of the map); the third and last is the variable, in the shapefile, which contains the geocode of the localities. If you don't have a map for you simulation, leave the list empty: `location = []`).

edges This is the name of the .CSV (comma-separated-values) file containing the list of edges and their attributes.

sites This is the name of the .CSV file containing the list of sites and their attributes.

encoding This is the encoding used in your sites and edges files. This is very important when you use location names which include non-ascii characters. The default encoding is *latin-1*. If you use any other encoding, please specify it here. Example: *utf-8*.

Note: All paths in the .epg file are relative to the working directory.

3.2.2 Part 2: EPIDEMIOLOGICAL MODEL

This is the main part of the script. It defines the epidemiological model to be run. The script reads:

```
modtype = SIR
```

Here, the type of epidemiological model is defined, in this case is a deterministic *SIR* model. Epigrass has some built-in models:

Name	Determ.	Stochastic
Susceptible-Infected-Recovered	<i>SIR</i>	<i>SIR_s</i>
Susceptible-Exposed-Infected-Recovered	<i>SEIR</i>	<i>SEIR_s</i>
Susceptible-Infected-Susceptible	<i>SIS</i>	<i>SIS_s</i>
Susceptible-Exposed-Infected-Susceptible	<i>SEIS</i>	<i>SEIS_s</i>
SIR with fraction with full immunity	<i>SIpRpS</i>	<i>SIpRpS_s</i>
SEIR with fraction with full immunity	<i>SEIpRpS</i>	<i>SEIpRpS_s</i>
SIR with partial immunity for all	<i>SIpR</i>	<i>SIpR_s</i>
SEIR with partial immunity for all	<i>SEIpR</i>	<i>SEIpR_s</i>
SIR with immunity wane	<i>SIRS</i>	<i>SIRS_s</i>

A description of these models can be found in the chapter *Epidemiological modeling*. The stochastic models use *Poisson* distribution as default for the number of new cases ($L(t+1)$). Besides these, the user may define his/her own model and access by the protect word Custom.

3.2.3 Part 3: MODEL PARAMETERS

The epidemic model is defined by variables and parameter which require initialization:

```
#####
# They can be specified as constants or as functions of global or
# site-specific variables. Site-specific variables are provided
# in the sites .csv file. In this file, all columns after the 4th
# are collected into a values tuple, which can be referenced here
# as values[0], values[1], values[2], etc.
# Examples:
# beta = 0.001
# beta=values[0] #assigns the first element of values to beta
# beta=0.001*values[1]
# beta=0.001*N # N is a global name for total site population
# Currently, Epigrass requires that parameters beta, alpha, e, r, delta, B, w, p
# be present in the .epg even if they will not be used. Do not erase these lines.
# Just disregard them if they are not useful to you.
```

```

beta = 0.4    #transmission coefficient (contact rate * transmissibility)
alpha = 1    # clumping parameter
e = 1       # inverse of incubation period
r = 0.1     # inverse of infectious period
delta = 1   # probability of acquiring full immunity [0,1]
B = 0       # Birth rate
w = 0       # probability of immunity waning [0,1]
p = 0       #

```

These are the model parameters. Not all parameters are necessary for all models. For example, *e* is only required for SEIR-like models. Don't remove the line, however because that will cause an error. We recommend that, if the parameter is not necessary, just add a comment after it as a reminder that it is not being used by the model.

In some cases, one may wish to assign site-specific parameters. For example, transmission rate may be different between localities that are very distant and are exposed to different climate. In this case site specific variables can be added as new columns to the site file. All columns after the geocode are packed into a tuple named *values* and can be referenced in the order they appear. I.e., the first element of the tuple is *values[0]*, the second element is *values[1]*, the third element is *values[2]* and so on.

Part 4: INITIAL CONDITIONS

In this part of the script, the initial conditions are defined. Here, the number of individuals in each epidemiological state, at the start of the simulation, is specified. It reads:

```

#####
# Here, the number of individuals in each epidemiological
# state (SEI) is specified. They can be specified in absolute
# or relative numbers.
# N is the population size of each site.
# The rule defined here will be applied equally to all sites.
# For site-specific definitions, use EVENTS (below)
# Examples:
#   S,E,I = 0.8*N, 10, 0.5*N
#   S,E,I = 0.5*N, 0.01*N, 0.05*N
#   S,E,I = N-1, 1, 0
S = N
E = 0
I = 0

```

Here, *N* is the total population in a site (as in the datafile for sites). In this example, we set all localities to the same initial conditions (all individuals susceptible) and use an event (see below) to introduce an infectious individual in a locality. The number of recovered individuals is implicit, as $R = N - (S + E + I)$

Another possibility is to define initial conditions that are different for each site. For this, the data must be available as extra columns in the site data file and these columns are referenced to using the *values* tuple explained above.

3.2.4 Part 5: EPIDEMIC EVENTS

The next step is to define events that will occur during the simulation. These events may be epidemiological (arrival of an infected, for example) or a public health action (vaccination campaign, for example):

```

#####
# Specify isolated events.
# Localities where the events are to take place should be Identified by the geocode, which
# comes after population size on the sites data file.
# All coverages must be a number between 0 and 1.
# Seed : [('locality1's geocode', epid state, n), ('locality2's geocode', epid state, n), ...]
# N infected cases will be added to locality at time 0.
# Vaccinate: [('locality1's geocode', [t1,t2,...], [cov1,cov2,...]), ('locality2's geocode', [t1,
# Quarantine: [(locality1's geocode,time,coverage), (locality2's geocode,time,coverage)]
#seed = [(4550601,'ip20',10)] #santo cristo #

```

```
seed = [(4552110,'ip20',10)] #pechincha #
#seed = []
Vaccinate = [] #
Quarantine = []
```

The events currently implemented are:

seed One or more infected individual(s) are introduced into a site, at the beginning of the simulation. The notation for a single event is: *Seed* = [(‘locality1’s geocode’, epid state, n), (‘locality2’s geocode’, epid state, n), ...]. For example, *seed* = [(2, I, 10)] programs the arrival of 10 infected individuals at site geocode 2, at time 1.

Vaccinate Implements a campaign that vaccinates a fraction of the population in a site, at a pre-defined time-step. For multiple events, the notation is: [(‘locality1’s geocode’, [t1,t2,...], [cov1,cov2,...]), (‘locality2’s geocode’, [t1,t2,...], [cov1,cov2,...])], where the first element of every triplet is the geocode of the city, the second element is a list of the time(s) when the campaign is carried on, and the third element is the coverage(s). For example, the event [(2,[10],[0.7])] means that city 2, at time 10, has 70% of its population vaccinated. Mathematically, it means (in the model), the removal of individuals from the susceptible to the recovered stage.

Quarantine Prevents any individual from leaving a site, starting at *t*. Currently disabled.

3.2.5 Part 6: TRANSPORTATION MODEL

Here, there are two options regarding the movement of infected individuals from site to site (through the edges). If *stochastic* = 0, the process is simulated deterministically. The number of infected passengers commuting through an edge is a fraction *p* of the infected population that is traveling. *p* is calculated as *total passengers/total population*.

If *stochastic* = 1, the number of passengers is sampled from a Poisson distribution with parameter given by the expected number of travelling infectives (calculated as above):

```
#=====#
# If doTransp = 1 the transportation dynamics will be
# included. Use 0 here only for debugging purposes.
doTransp = 1

# Mechanism can be stochastic (1) or deterministic(0).
stochastic = 1
#Average speed of transportation system in km per time step. Enter 0 for instantaneous travel.
#Distance unit must be the same specified in edges files
speed =0 #1440 km/day -- equivalent to 60 km/h
```

That ends the definition of the model.

3.2.6 Part 7: SIMULATION AND OUTPUT

Now it is time to define some final operational variables for the simulation:

```
#=====#
# Number of steps
steps = 50

# Output dir. Must be a full path. If empty the output will be generated on the
# same path as the model script.
outdir =

# Output file
outfile = simul.dat
```

```

# Database Output
# MySQLout can be 0 (no database output) or 1
MySQLout = 1

# Report Generation
# The variable report can take the following values:
# 0 - No report is generated.
# 1 - A network analysis report is generated in PDF Format.
# 2 - An epidemiological report is generated in PDF Format.
# 3 - A full report is generated in PDF Format.
# siteRep is a list with site geocodes. For each site in this list, a detailed report is apended to
report = 0
siteRep = []

# Replicate runs
# If replicas is set to an integer(n) larger than zero, the model will be run
# n times and the results will be consolidated before storing.
# RandSeed = 1: the seed will be randomized on each replicate
# RandSeed = 2: seeds are taken sequentially from the site's file
# Note: Replicate mode automatically turn off report and batch options.
Replicas = 10
RandSeed = 2
#Batch Run
# list other scripts to be run in after this one. don't forget the extension .epg
# model scripts must be in the same directory as this file or provide full path.
# Example: Batch = ['model2.epg', 'model3.epg', '/home/jose/model4.epg']
Batch = []#['sarsDF.epg', 'sarsPA.epg', 'sarsRS.epg']

```

where,

step Number of time steps for the simulation.

outdir Directory for data output (currently not in use)

outfile .csv filename that can be imported into R as a dataframe. This .csv file contains the simulated timeseries for all nodes.

MySQLout Use `MySQLout = 1` if simulated time series are to be stored in MySQL database. Time series of *L*, *S*, *E*, and *I*, from simulations, are stored in a MySQL database named *epigrass*. The results of each individual simulation is stored in a different table named after the model's script name, the date and time the simulation has been run. For instance, suppose you run a simulation of a model stored in a file named `script.epg`, then at the end of the simulation, a new table in the *epigrass* database will be created with the following name: `script_Wed_Jan_26_154411_2005`. Thus, the results of multiple runs from the same model get stored independently.

Batch=[] Script files included in this list are executed after the currently file is finished.

3.3 The Graphical User Interface(GUI)

Epigrass comes with a simple but effective GUI, that allows the user to control some aspects of the run-time behavior of the system. The Gui can be invoked by typing `texttt{epigrass}` in prompt of a console. We suggest the user to start Epigrass from the same directory where his/her model definition is located (`.csv` and `.epg` files).

All the information that is entered via the GUI gets stored in a hidden file called `texttt{.epigrassrc}` stored in the home folder of the user. Every time the GUI is invoked, the data stored in the `texttt{.epigrassrc}` file is used to fill the forms in the GUI. The gui is designed as a tabbed notebook with four tabs (Run Options, Settings, Utilities, and Visualization).

At the bottom of the Gui there are three buttons **Help**, **Start** and **Exit**. Their functions will be explained below. Immediately above the **Run** and **Exit** buttons, there is a small numeric display that will display the simulation progress after it has been started.

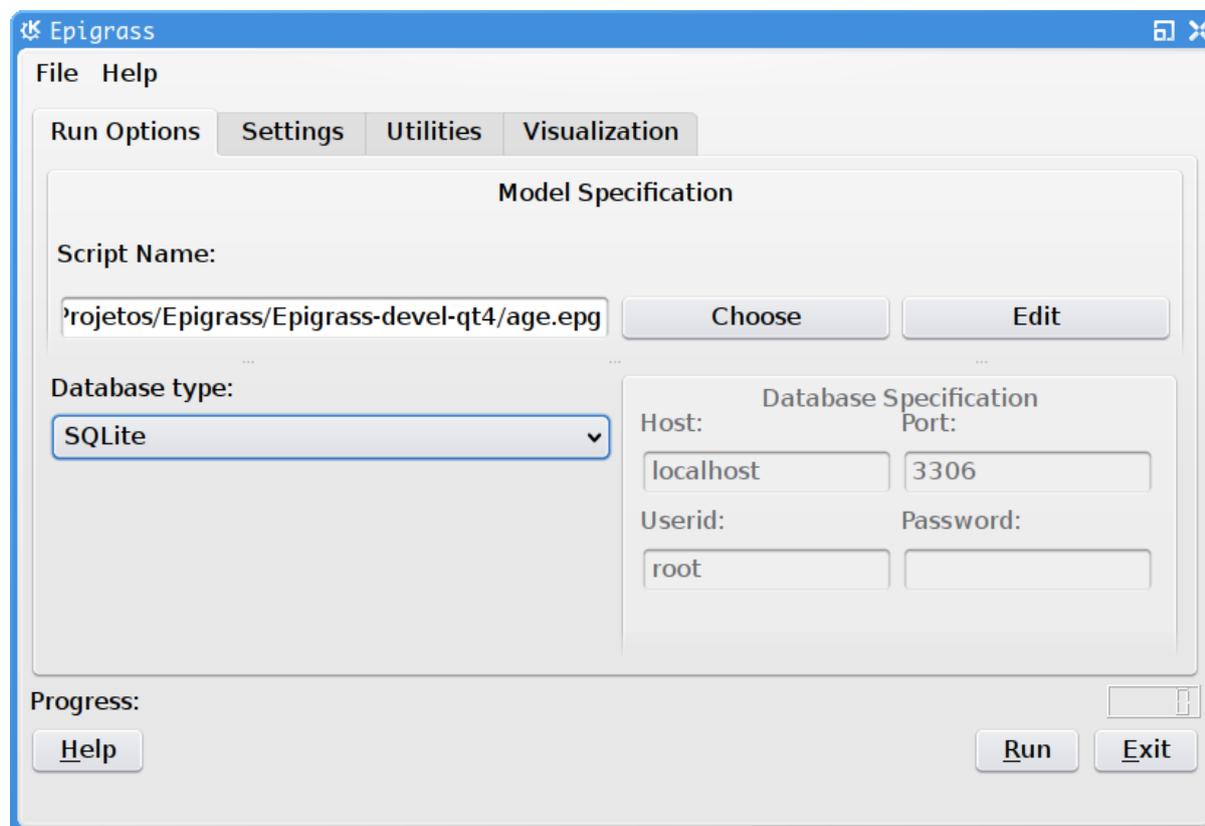


Figure 3.1: First tab of Epigrass GUI. This is where you setup your database output settings and the model to be run.

3.3.1 Run Options

The first tab of the GUI, contains a number of variables that, with the exception of the model script filename, should remain the same for most simulations you are going to run.

On the top of the first tab is a text box to enter the file name of the model script (`something.epg`). By clicking on the **Choose** button at the right of this box, you get a file selection dialog to select your script file. If you need you can click on the **Edit** button below to edit the script file with your favorite text editor.

Below, you can enter details about the MySQL database that will store the output of your simulations. Here you can enter the server IP, port, user and password. On the first time you run the GUI these input boxes will be filled with the default values for these variables (server on localhost, port 3306, user epigrass and password epigrass)

3.3.2 Settings

On the settings page, you can enter personal details such as user name (To be used in the simulation report), preferred text editor and preferred PDF reader. The preferred text editor will be used to open your script from the GUI, when you click on the edit button in the first tab. The PDF reader specified, will be used to open the report file, when requested (Utilities tab) and the user manual, when the user clicks on the help button on the bottom-left corner of the GUI.

On this tab, the language of the GUI can also be selected from a list of available translations. The effects of language changes will only take place when the next time the GUI is started.

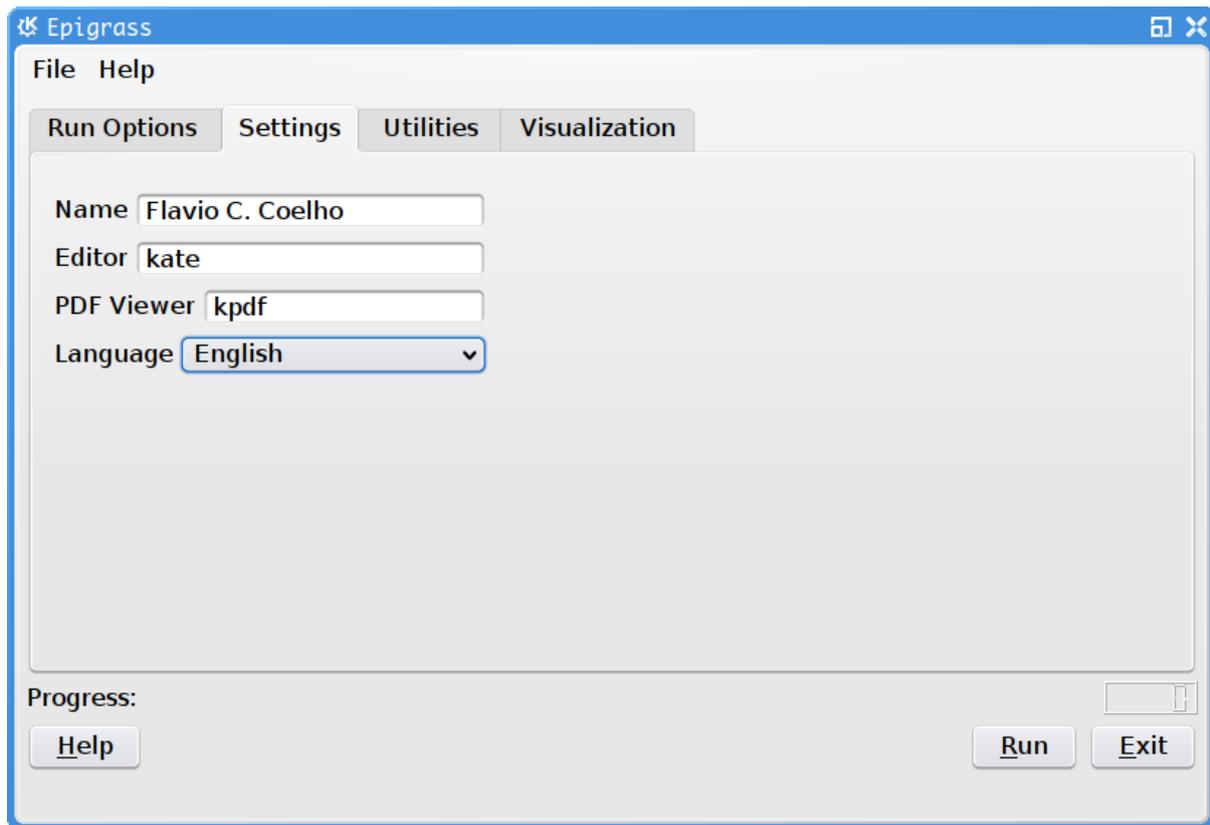


Figure 3.2: Settings tab of the Epigrass GUI. This is where you configure the behavior of the GUI. Values set here will be remembered on future runs.

3.3.3 Utilities

In the Utilities tab, you can get feedback from the simulator. Especially during long simulation runs, it is good to know how it is progressing. During the simulation, text messages regarding the status of the simulation are written to the text box on the left.

On the right, there is a button for backing up the data base and another for opening the report generated by the last simulation. Since report PDFs are stored in folders directly below the ones on which the simulation is started, older reports should still be accessible and can be opened directly by selecting the desired report using the operating system's file manager.

3.3.4 Visualization

The fourth tab of the GUI is the visualization Tab. This tab was designed for playing animations of any simulation data that is stored in the database. Pressing the **Scan DB** button, causes the available tables in the epigrass database to be listed in the *Simulations stored* combo box. The user can then select one of these simulations to visualize. Once the simulation is selected the *Variable to display* combo-box will fill-up with the variables in the table devoted to the simulation. Select a variable.

Once the **Start animation** button is pressed, a graphical display window pops up, and the simulation results will be displayed as a map or a graph (if no map was specified at the `.epg`). The animation can be replayed or moved to any timestep by dragging on the slider under the display widget. When the user moves the mouse over a polygon in the map (or node in the graph) its name and geocode appears as a tooltip. Polygons (nodes) can be selected with the mouse to display their full time series in the plot below the top display widget.

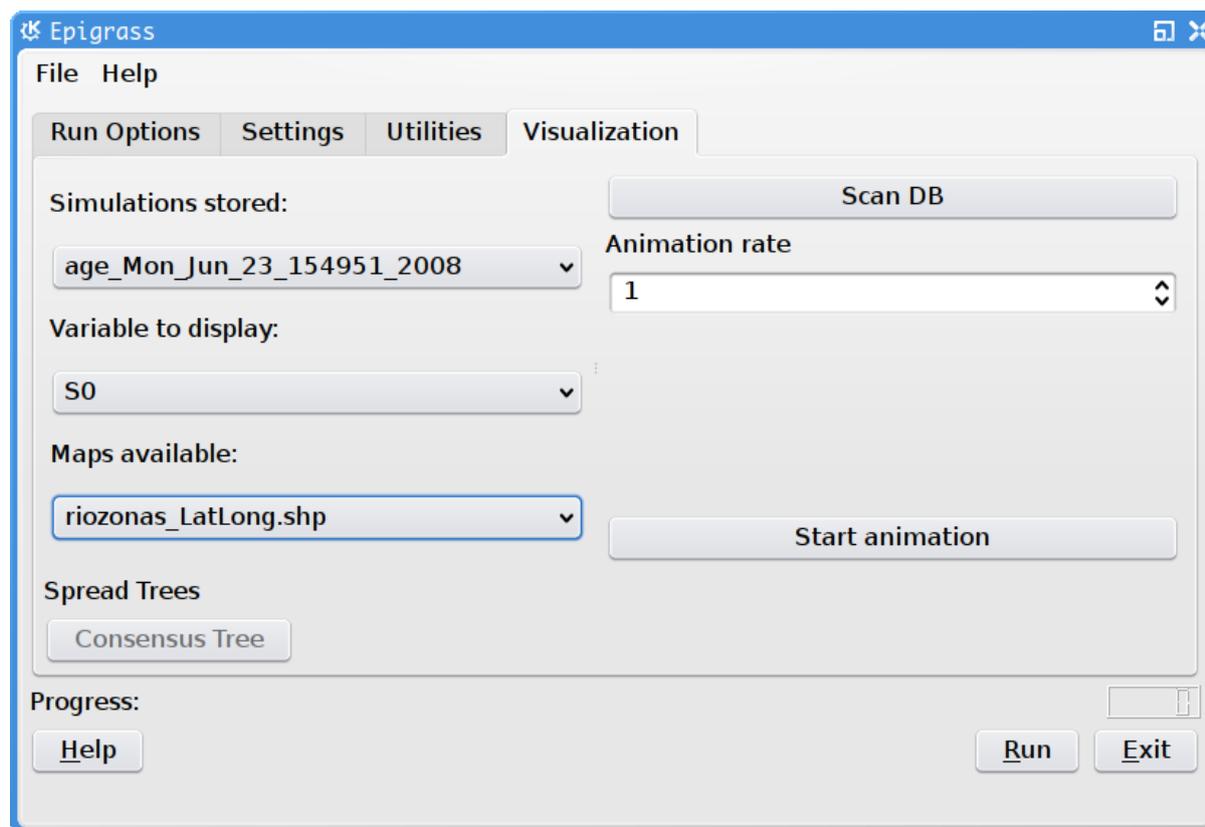


Figure 3.3: Visualization tab of the Epigrass GUI. The simulations and variables to inspect are chosen here.

3.3.5 Operation

For running simulations, after all the information has been entered and checked on the first tab of the GUI, you can press the **Run** button to start the simulation or the `:guilabel'Exit'` button. When the **Run** button is pressed, the `~/epigrassrc` file is updated with all the information entered in the GUI. If the **Exit** button is pressed, all information entered since the last time the **Run** button was pressed is lost.

Epigrass also allows running simulation straight from the command line, with the **epirunner** executable. all you have to do is:

```
$ epirunner mymodel.epg
```

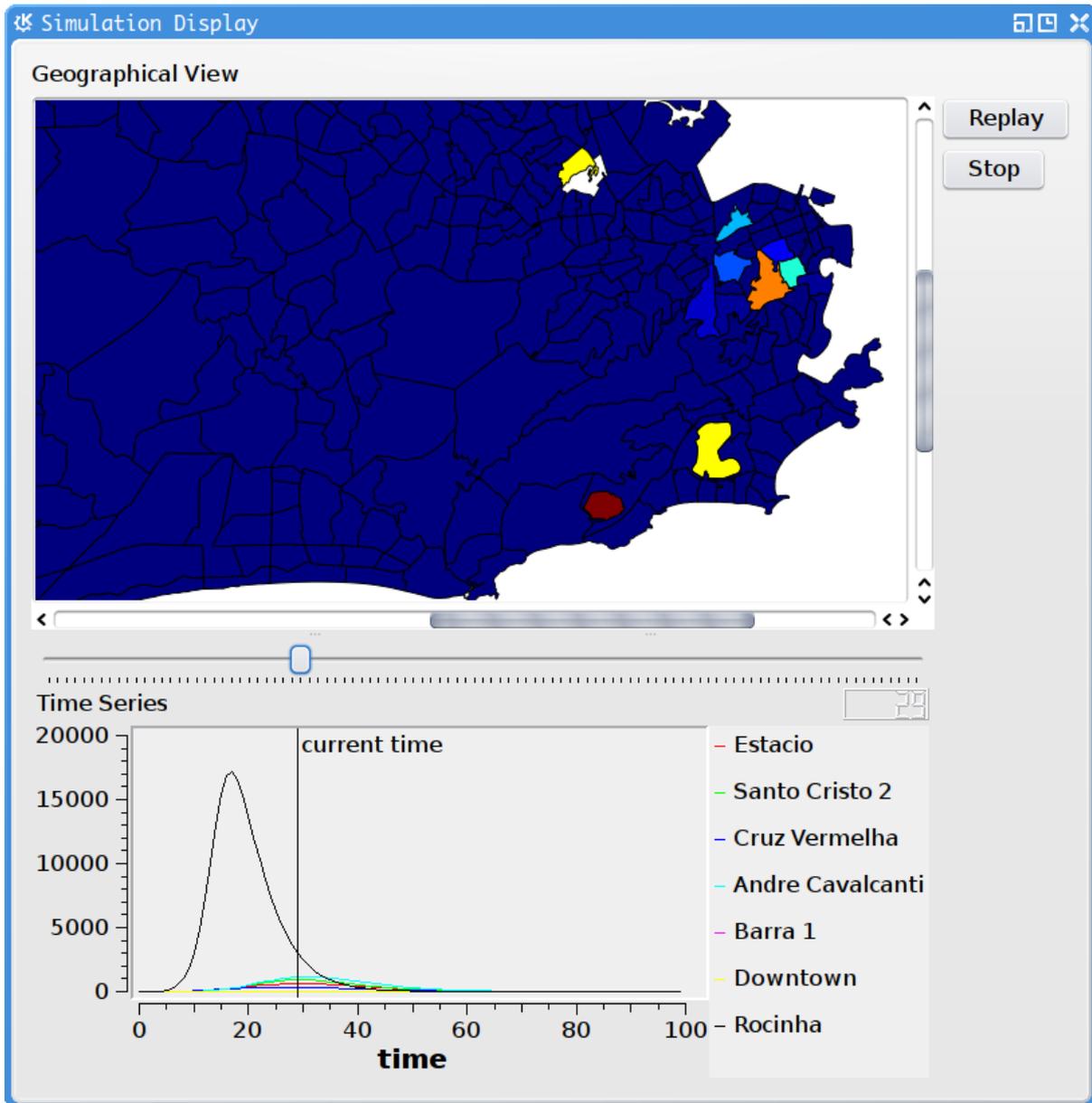
and the model will be executed with the settings specified in the `~/epigrassrc` file. for help with *epirunner*, type:

```
$ epirunner -h
```

```
Usage: epirunner [options] your_model.epg
```

Options:

```
--version          show program's version number and exit
-h, --help         show this help message and exit
-b <mysql|sqlite|csv>, --backend=<mysql|sqlite|csv>
                                                           Define which datastorage backend to use
-u DBUSER, --dbusername=DBUSER
                                                           MySQL user name
-p DBPASS, --password=DBPASS
                                                           MySQL password for user
-H DBHOST, --dbhost=DBHOST
                                                           MySQL hostname or IP address
```



Writing Custom Models

The most powerful feature of Epigrass is the ability to use custom models. It allows the user to specify intra-node dynamics and, in doing so, break away from the limitations imposed by the built-in models.

By learning to write his/her own models, the user begins to realize the full potential of Epigrass, which goes beyond being a platform to simulate networked epidemics. In reality Epigrass can be used to model any distributed dynamical system taking place on a set of nodes (connected or not).

4.1 Getting Started

The best way to get started in writing custom models is to look at the example distributed with Epigrass. It can be found in `demos/CustomModel_example.py`:

```
# This is a custom model to used in place of Epigrass' built-in models. Custom
# models must always be on a file named CustomModel.py and contain at least
# a function named Model. Both the File name and the function Names are case-sensitive,
# so be careful. Please refer to the manual for intructions on how to write your
# own custom models.

def Model(self,vars,par,theta=0, npass=0):
    """
    Calculates the model SIR, and return its values.
    * vars The state variables of the models
    * par The parameters (Beta, alpha, E,r,delta,B, w, p) see docs.
    * theta = infectious individuals from neighbor sites
    * npass = Total number of people arriving at this node
    """
    # Get state variables' current values

    if self.parentSite.parentGraph.simstep == 1: # if first step
        # Define variable names to appear in the output
        self.parentSite.vnames = ('Exposed','Infectious','Susceptible')
        # And get state variables's initial values (stored in dict self.bi)

        E,I,S = (self.bi['e'],self.bi['i'],self.bi['s'])
    else: # if nor first step
        E,I,S = vars

    # Get parameter values
    N = self.parentSite.totpop
    beta,alpha,e,r,delta,B,w,p = (self.bp['beta'],self.bp['alpha'],
    self.bp['e'],self.bp['r'],self.bp['delta'],self.bp['b'],
    self.bp['w'],self.bp['p'])

    #Vaccination event (optional)
    if self.parentSite.vaccineNow:
        S -= self.parentSite.vaccov*S
```

```
# Model
Lpos = beta*S*((I+theta)/(N+npass))**alpha #Number of new cases
Ipos = (1-r)*I + Lpos
Spos = S + B - Lpos
Rpos = N-(Spos+Ipos)

# Update stats
self.parentSite.totalcases += Lpos #update number of cases
self.parentSite.incidence.append(Lpos)

# Raise site infected flag and add parent site to the epidemic history list.
if not self.parentSite.infected:
    if Lpos > 0:
        self.parentSite.infected = self.parentSite.parentGraph.simstep
        self.parentSite.parentGraph.epipath.append(
            (self.parentSite.parentGraph.simstep,
             self.parentSite, self.parentSite.infectors))

self.parentSite.migInf.append(Ipos)

return [0, Ipos, Spos]
```

Let's analyze the above code. The first thing to notice is that an Epigrass custom model is a Python program. So anything you can do with Python in your system, you can do in your custom model. Naturally, your knowledge of the Python programming language will define how far you can go in this customization. There are a few requirements on this Python program in order to make it a valid custom model from Epigrass's perspective.

1. **It must define a global function named Model. This function will be inserted as a method on every node object, at run**

self: reference to the model object.

- *vars*: A list with the values of the model's state variables in time t-1 in the same order as returned by this function.
- *par*: The parameters of the model. Listed in the same order as defined in the .epg file.
- *theta*: Number of infectious individuals arriving from neighboring sites. For disconnected models, it is 0.
- *npass*: The total number of passengers arriving from neighboring sites. For disconnected models, it is 0.

- (b) In the beginning of the function you define a list of strings (`self.parentSite.vnames`) which will be the names used when storing the resulting time-series in the database. Choose strings that are not very long and are meaningful. You only need to do this once, at the beginning of the simulation so put it inside an *if* statement, which will be executed only at time-step 1 (see code above).
- (c) After defining variable names, set their initial values in the same *if* clause. An *else* clause linked to this one will set variables values for the rest of the simulation.
- (d) Define local names for the total population *N* and fixed parameters.
- (e) Proceed to implement your model anyway you see fit.
- (f) **Feed some site level variables (*incidence*), with the result of the simulation.**
 - *incidence*: list of new cases per time step.
 - *infected*: Boolean stating if the site has been infected, i.e., it has had an autoctonous case.
 - *epipath*: This variable is at the graph level and contains the path of spread of the simulation.
 - *migInf*: Number of infectious individuals in this site per time-step.
- (g) Finally, this function must return a list/tuple with the values of the state variables in the same order as received in *vars*.

Warning: The strings in `self.parentSite.vnames` must be valid *SQL* variable names, or else you will have a insert error at the end of the simulation.

After defining this function with all its required features, you can continue to develop your custom model, writing other functions, classes, etc. Note however, that only the *Model* function will be called by Epigrass, so any other code you add to your program must be called from within that function.

Note: Since `CustomModel` is imported from within Epigrass, any global code (unindented) in it is also executed. So you may add imports and other initialization code.

Warning: The name `CustomModel.py` is case-sensitive and cannot be changed. The same is true for the *Model* function.

4.2 The Environment

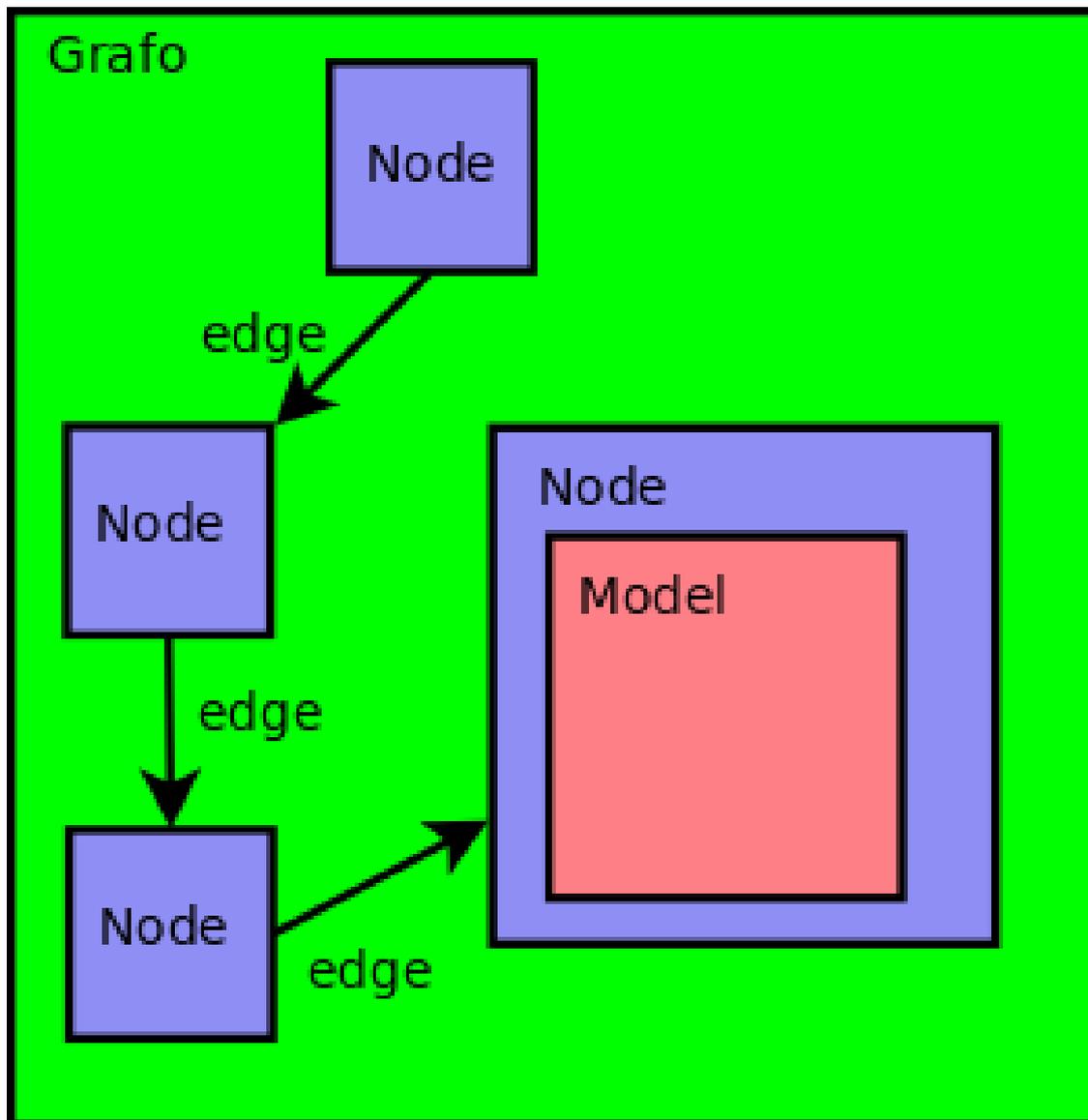


Figure 4.1: Nesting of the objects inside a Simulate object.

From quickly going through the example Custom model above it probably became clear, to the Python-initiated, that you can access variables at the node and graph levels. This is possible because *Model* becomes a method in a node object which in its turn is contained into a graph object (see figure).

Besides being nested within the *graph* object, *node* and *edge* contain references to their containers. This means

that using the introspective abilities of Python the user can access any information at any level of the full *graph* model and use it in the custom model. In order to help you do this, Let's establish an API for developing custom models.

4.2.1 Model Development API

All attributes and methods (functions) from all around the simulation must be references from the model's perspective, denoted by *self*. The parent objects can be accessed through the following notation:

- ***self.parentSite*** Is the Site (node) containing the model.
- ***self.parentSite.parentGraph*** Is the Graph containing the parent site of the model.

The following attributes and methods can be accessed by appending them to one of the objects above. For example:

```
self.parentSite.parentGraph.simstep
```

Site Attributes and Methods

Not all attributes and methods are listed, only the most useful. For a complete reference, look at the source code documentation.

```
class Site ()
    self.parentSite. Actually named siteobj in the source code.
    bi
        Dictionary with initial values for all of the model's state variables. Keys are the variable names.
    bp
        Dictionary with initial values for all of the model's parameters. Keys are the parameter names.
    totpop
        Initial total population
    ts
        List containing the model output time series (variables in the same order of the model)
    incidence
        Incidence time series
    infected
        Has the site been already infected? (logical variable)
    sitename
        Site's name (provided in the .csv)
    values
        Tuple containing extra-variables provided by .csv file
    parentGraph
        Graph to which Site belongs (see class Graph)
    edges
        List containing all edge objects connected to Site
    inedges
        List containing all inbound edges
    outedges
        List containing all outbound edges
    geocode
        Site's geocode
    modtype
        Type of dynamic model running in Site
```

vaccination

Time and coverage of vaccination event. Format as in .epg

vaccineNow

Flag indicating that it is vaccine day (0 or 1)

vaccov

Current vaccination coverage

vaccinate (*cov*)

At time *t*, the population is vaccinated with coverage *cov*

getOutEdges ()

Returns list of outbound edges

getInEdges ()

Returns list of inbound edges

getNeighbors ()

Returns a dictionary of neighbor sites as keys and distances as values

getDistanceFromNeighbor (*site*)

Returns the distance in km from a given neighbor

getDegree (*site*)

Returns degree of this site, that is, the number of sites connected to it

Graph Attributes and Methods

Not all attributes and methods are listed, only the most useful. For a complete reference, look at the source code documentation.

class Graph ()

self.parentSite.parentGraph

simstep

Time-step of the simulation. Use it to keep track of the simulation progress.

speed

The speed of the transportation system

maxstep

Final time-step of the simulation

epi_size

Current size of the epidemic, graph-wise.

site_list

Full list of nodes in the graph. Each element in this list is a real node object.

edge_list

Full list of edges in the graph. Each element in this list is a real edge object.

getSite (*name*)

Returns an site object named *name*

Indices and tables

- *Index*
- *Module Index*
- *Search Page*

INDEX

B

bi (Site attribute), 26
bp (Site attribute), 26

C

custom models, 23

D

Database

epigrass database, 5
results table, 5

E

edge, 4
edge_list (Graph attribute), 27
edges (Site attribute), 26
epysize (Graph attribute), 27

G

geocode (Site attribute), 26
getDegree () (Site method), 27
getDistanceFromNeighbor () (Site method), 27
getInEdges () (Site method), 27
getNeighbors () (Site method), 27
getOutEdges () (Site method), 27
getSite () (Graph method), 27
Graph (class), 27

I

incidence (Site attribute), 26
inedges (Site attribute), 26
infected (Site attribute), 26

M

maxstep (Graph attribute), 27
models, 4
epidemiological models, 4
network models, 4
modtype (Site attribute), 26

N

node, 3

O

outedges (Site attribute), 26

P

parentGraph (Site attribute), 26

S

shapefile, 3
simstep (Graph attribute), 27
Site (class), 26
site, 3
site_list (Graph attribute), 27
sitename (Site attribute), 26
speed (Graph attribute), 27

T

totpop (Site attribute), 26
ts (Site attribute), 26

V

vaccinate () (Site method), 27
vaccination (Site attribute), 27
vaccineNow (Site attribute), 27
vaccov (Site attribute), 27
values (Site attribute), 26