# MAULWURF Documentation

***Release 0.1.11***

**Piotr Czaja**

2014 - 03 - 19

Contents

Main goal of this project is to make a set of objects for simulation of guided transit operation systems. To achieve this a python library will be made.

It is intended to be only an analytical tool without graphical user interface. I recommend Ipython for use with this library.

Feel free to use this library for any purpose you like. The only restriction is to add link to main page of the MAULWURF project in your paper and software documentation.

You are welcome to join this project. I'm ready to consider to grand you write access to the repository after checking some of your earlier proposals and code.

This document describes both: *data model for guided transit operation systems (DMS)* and its *software reference implementation library (SRL)*. Data model consists of network object definitions. Each of object descriptions contains:

- object name - the same as classes lowest order (DMS, SRL)
- input data - designated as **Parameters** (DMS, SRL)
- output data - designated as **Returns** (DMS, SRL)
- internal variables - designated as **Variables** (SRL)

The code library is available under license: *GPL*. The data model for guided transit operation systems i available under license: *MIT*.

This document applies to:

- version 0.1.0 (release: 0.1.11) of the software package (SRL)
- version 0.1.0 (release: 0.1.11) of the data model (DMS).

Indices and tables (mainly only in html): *genindex*, *modindex*

---

**Note:**

1. A link to the current version of MAULWURF project is always available through: http://ovzo.net/. To find it please navigate to Projekty/Project website.
2. The documentation is written in English to make it publicly available - it is not my native language. If you find any language mistakes or errors - don't complain, please send me a bug report or make the corrections yourself.

---

The document consists of three parts: user's manual, code documentation (python modules), attachments.

CHAPTER 1

# User's manual

Description of installation, configuration and maintenance of the library and tools.

## 1.1 Installation and configuration of the package

### 1.1.1 System requirements

The requirements of the package are:

- Python 3.3+
- Ipython 1.1+
- Sphinx 1.2+

### 1.1.2 Components and architecture

> **Warning:** The package includes only a few usage examples of the library, and it should be so.

After successfully installation please go to your work directory and run ipython engine to access the library.

You can find some examples of ipython-netbook session in the repository.

### 1.1.3 Sources

Sources of this project are maintened in GIT repository (public: GITHUB, private/development: gitolite). You can find the main website of the project on:

`https://sec.ovzo.net/dev/`

To clone the repository use Github (for releases or major development tasks) or the kaminari.ovzo.net private repository (after achieving read/write access to it) for minor tasks and conceptual work.

### 1.1.4 How to build the package

- Clone the repository to local folder
- Checkout into prefered version
- Move/copy documentations Makefile:

```
cp .../docs/etc/Makefile .../docs/Makefile
```

- Build documentation:

```
cd .../docs/
make html
```

- Build package:

```
cd root-repo-directory
python3 setup.py sdist
```

- You will find the new package in *sdist* directory

### 1.1.5 How to install MAULWURF library

You can use *pip-3.x* or do it manually:

```
python3 setup.py install
```

after uncompressing the package directory.

Please use the git repository - it will be not available on PyPi.

## 1.2 Introduction to the library (quick start)

You can use the library in several different ways.

The simplest one is:

1. Make your *virtualenv* with all the dependencies of the package
2. Prepare the package
3. Install the package
4. Go to your work directory and run ipython's instance.
5. Check out the installation: rerun ipython's sessions examples
6. Make config files for network you want to analyse.
7. Make your's simulations/experiments/...

## 1.3 Technical overview and detailed solutions description

You can simulate an network in many different ways. This part of documentation describes solutions taken for implementation of simulation model and contains main arguments which determined their choice.

For the network objects description, please see the code documentation.

### 1.3.1 Time and synchronization of events in network

Changed in version 0.1.2.

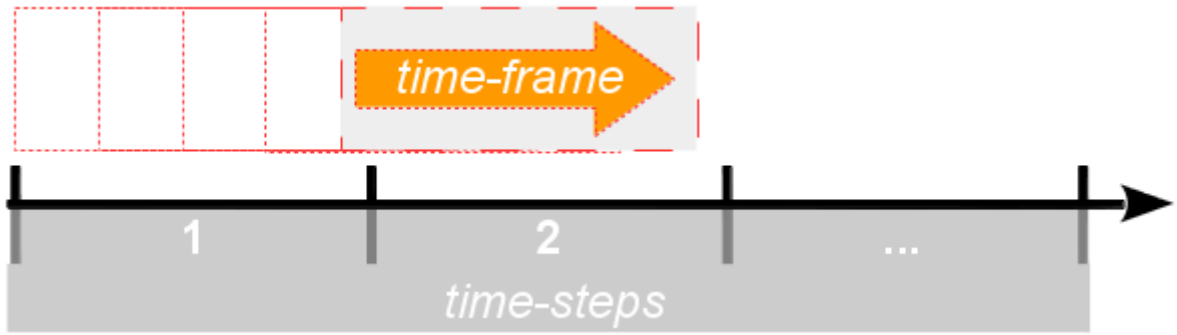


Figure 1.1: Time-frame vs time-step

*time-frame* is a period of time which establishes the framework for prediction and determination of parameters of the model objects.

*time-step* is a set of moments in time for prediction and determination of changes in the network infrastructure.

**Infrastructure** facilities are determined at intervals specified by the time frame and effects in time-steps. The properties of the technical objects results of the events that took place in earlier periods (time-steps). The current time frame does not affect the behavior of the infrastructure objects (This simplification is due to excessive correlation between events and properties of the network).

Events related to **vehicles** are forecasted for the duration of the time frame. We solve only the first event during the time frame. Then we update movements of vehicles for the moment. We move then the beginning of the time frame to this moment.

On start of simulation time-frame coincide with the time-step and this repeats every time-step moment.

Other implementations of network behavior used in the past where: Petri Nets like model, a queue like model, static time frame model. Some solutions of network behavior from previous models are still there, after adjustment.

The most simple and natural solution is to divide each segment of the network into smaller equal pieces. The procedure resembles in some ways **Petri Nets**. The main problem with this approach is the low accuracy in reflecting lengths. The accuracy of measurement is almost the length of the subsegment. If two vehicles move along the same path length but a different number of sub/segments, they can not even meet at a designated place on the network. If so - you can not take into consideration the exchange of conflict on the network, which result from crossing the tracks.

Treatment pieces of the network as **a queue** leads to problems if, in a single calculation step vehicle traverses many sections. Synchronization and registration of many events, including the dependency between the moving vehicles at different speeds on sections of the network becomes often impossible.

The next solution is the use of **static time frames**. The time frame is a period in which events are considered. The time frame coincides with a major step in the calculation. Each time step contains all network events which must be solved (including: the sequence of events in a single time-frame, dependency between events).

Calculation of each static time-frame consists of the following steps:

1. Defining events associated with the operation of the infrastructure on the sections between nodes, which are based on the results of the previous periods.
2. Determination of the maximum distance that can traverse each of the vehicles
3. Defining of events and related logical nodes.
4. Movement of the vehicles when the range goes beyond the following node and within single sections in the events order.

In each of these cases, the ability to block and delay by slower vehicles, have to be considered.

The need of network section's part reservation for each of the vehicles in conjunction with the dependency in release of space from preceding vehicles, causes the correct timing and order of processing events importance. The introduction of short-time frames to calculations can help to simplify the scenarios.

Calculation of the static time frames does not reflect all the changes occurring during their lifetime. The omitted changes results from the dependencies between paths of different vehicles.

### 1.3.2 Speed (Vehicles)

New in version 0.1.0.

Speed of the vehicles is a major determinant of mapping the dynamic behavior of the network at the designated time. Constant, average speed of vehicles is sufficient to analyze most of the networks. Each vehicle can still move at different average speed and be slowed by another, preceding vehicle.

Vehicle speed can not exceed:

- maximum speed of analyzed vehicle,
- current maximum speed on network section (respectively: maximum, average, indicated),
- the speed of directly preceding vehicle,

and its value is the minimum of the three.

The use of variable speed for single vehicle is not effective in the analysis of the network. It significantly increases the computational effort and does not increase the adequacy of the model to the actual network behavior.

### 1.3.3 Events and network nodes

Changed in version 0.1.1.

Each event corresponds to the appropriate node on the network, physical or logically - depending on the type of event. Each physical node corresponds to a event associated with it.

Physical nodes are inserted where there are:

- crossovers
- crossings
- changes in the operating characteristics of track

The events are generated mostly by the physical nodes when the end of the vehicle can potentially pass the node during its movement on the network.

The logical nodes are placed where there are identified events.

Events that result in the introduction of logical nodes to the network are:

- catching the vehicle when one of them is moving and the other is stationary
- catching the vehicle when they move at different speed
- damaged section of track

Physical and logical nodes are on two different layers. Physical nodes adjacent to sections (modules) - logical nodes are assigned to these sections.

Both of physical nodes and logical distinguish their specific types because of their different impact on network behavior.

For the logical nodes a control point may be both start and end of the vehicle during its movement with or without a corresponding additional section before and after it.

The general principles are:

- Events include a time and place.
- Identification of the site is followed by the node.
- Identification of time is followed by an event.
- Events define the points in time at which the network state is updated during the simulation

What happens in the network between events is for the logic of a network of minor importance, and only through the impact on the events itself.

In the implementation of the model we use only the physical nodes and events. We leave the selection of logical nodes as they result from the events.

### 1.3.4 Simulation steps

Changed in version 0.1.2.

The simulation is done by examining changes in the network for the moment of selected events.

Time steps (their beginnings) generates additional events (see: *time-step*).

Proper simulation generally consist of the following steps:

1. If the beginning of the time-frame falls on moment of a time-step, we analyze the changes in the sections of the network (damaged track, ...) for period to the next time-step
2. We check maximum range during the current time-frame for each vehicle
3. We identify the first physical node overcome on the road by each vehicle and mark it as an event
4. We identify all logical nodes on the road and the first from them mark as an event for each of vehicle
5. We create a list of all the events which were identified and take the nearest event
6. We find solution for the taken event, depending on the type
7. We update the location of vehicles

8. We move the beginning of the time-frame to the just analyzed event time

Proper simulation also includes enabling and disabling vehicles moving at the end of their routes.

In the moments defined by the time-step are generated all the changes on the sections of track and they are also saved for future use during time to next time-step. From a technical point of view, the time-step is a kind of global event covering both infrastructure and movement by the vehicle.

Since the operations events on sections of the network are generated in advance for time to next time-step, the updates that go in the future to the currently analyzed event are ignored in reports associated with the use of vehicles in the network.

### 1.3.5 Global network events

New in version 0.1.2.

Global events that may occur:

- time-step
- introduction of vehicle/s to network
- dropping network by the vehicle/s

Global events are plain events with the difference that, in general relate to more than one place in the network.

### 1.3.6 The impact of the environment on the network

In the simulation, it is important to give the bonds of the system with the environment. Environment affects the behavior of the system through the network generated events. These events are associated with nodes and modules.

The nodes represent potential points in the network where vehicles can collide with objects outside.

The collision may occur at longer distance and not be associated with the intersection directions of traffic but their parallelism. Then a potential collision is not determined by a network node, but the calculated probabilities of an event on module.

Events on modules of the network coming from the environment are mainly caused by the fact that these sections are not separated from the environment (eg, damage which is not caused by a network vehicle).

CHAPTER 2

# Code documentation

Implementation of the model consists of dedicated code modules (Python):

- modules that define the behavior of the model objects
- modules that implement the logic and functionality of the model
- modules defining the runtime environment and how to cooperate with other software
- modules that allow the data logging and reporting for later analysis

Currently implemented code modules:

## 2.1 `netmod`

Module NETMOD defines global objects for representation of network's sectors between its nods or allows you to enter and exit the vehicle from the network (see: *sector*):

- TracMod *(A)* - is a transit sector with FIFO queue
- BufrMod *(B)* - is a buffer sector, with algorithmically generated leaving order
- InOuMod *(C)* - is a enter/exit sector, for introducing to or leavieng a vehicle the network


Figure 2.1: Types of model modules

In addition to these basic types of modules, it is possible to to distinguish the network modules with fixed or variable direction movement.

Events that define the behavior of the network are assigned to each sector (module). We can distinguish the following basic types of events on modules which generates the data as result of related events:

- vehicle entry to the network section ($MA$)
- exit the vehicle from the network section ($MB$)

- occupation section of the network to enter ($MA + MAC$)
- stop on the route ($MS_X$)
- checkpoint on the route ($MS_X$)
- damage of section ($MS_2 - MS_1$)
- blocking section by a damaged vehicle ($MS_2 - MS_1$)
- blocking section by a moving vehicle ($MS_2 - MS_1$)
- change the operating parameters (eg slowing) of track section ($MS_2 - MS_1$)

Some of these events are being logged by the vehicle and some by the section of the track (module).

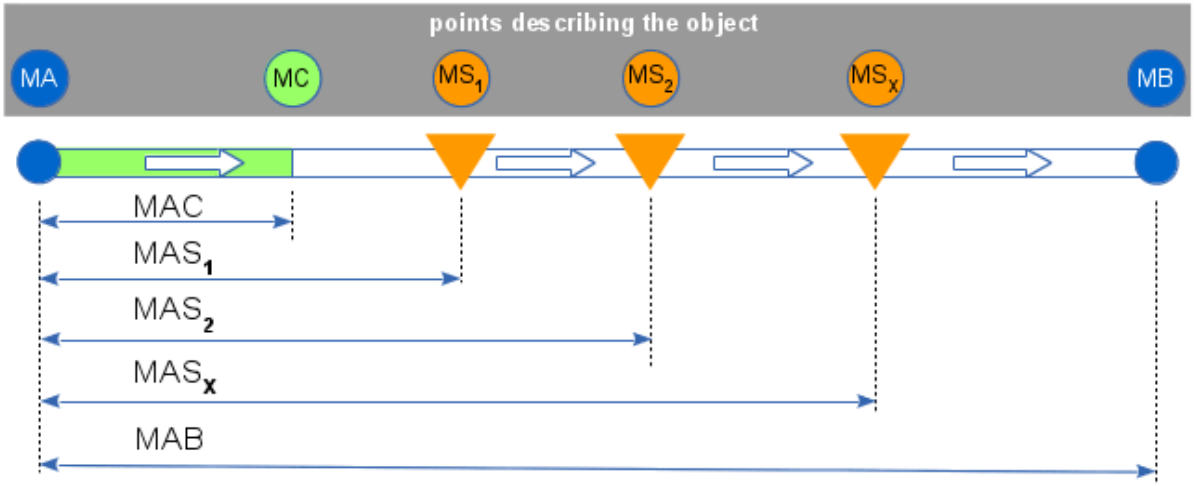


Figure 2.2: The main elements of the module which generates the data as a result of related events

If it is necessary to introduce track sections which only one vehicle can enter at a given time that may be achieved by establishing a permanent MC = MB. Sometimes the next module is shorter than the vehicle, then the vehicle is moved once a reservation on succeeding module is done where the vehicle fit in its entirety. Then intermediate modules and nodes on the route are also blocked.

The entrance to the module can be blocked by:

- result of blocking MAC section,
- as a result of events in the section,
- setting the lock on net section.

The name of the module consists of three parts: the first node, the intermediate module, the second node. Characteristics is described separately for each direction and the direction is determined by the starting node, eg: module generally [10, 23, 11], the module with the selected direction [10, 23, =]. Module: maulwurf.netmod

**class** `netmod.`**`Sector`**(*name*, *session*)

This main class defines all basic behavior of network sectors between junctions (or other network nodes)

**`name`**

class instance property (tuple) => allowed methods: get

**class** netmod.**TracMod**(*name*, *session*)
Bases: netmod.Sector

Transit network sector with FIFO queue.

**__init__**(*name*, *session*)
creates blank sector object ; all parameters **required**

**Parameters**

- **name** – (id) of the section object
- **session** – id of the session, values: (a) 'current' sets value from global variables, (b) any correct given previously created session id

**class** netmod.**BufrMod**(*name*, *session*)
Bases: netmod.Sector

Buffer network sector

**__init__**(*name*, *session*)
creates blank sector object ; all parameters **required**

**Parameters**

- **name** – (id) of the section object
- **session** – id of the session, values: (a) 'current' sets value from global variables, (b) any correct given previously created session id

**class** netmod.**InOuMod**(*name*, *session*)
Bases: netmod.Sector

Enter/exit sector, for introducing to or leaving a vehicle the network

**__init__**(*name*, *session*)
creates blank sector object ; all parameters **required**

**Parameters**

- **name** – (id) of the section object
- **session** – id of the session, values: (a) 'current' sets value from global variables, (b) any correct given previously created session id

## 2.2 netnod

NETNOD module defines the network nodes (see: *node*). There are three main cases to insert a node to the network:

- the intersection with a road(to show the influence of the environment),
- splits the route into sections which can occupy only one vehicle (including MAB = MAC),
- the mapping of switches in the network.

For all other cases, insertion of a node must be very cautious, since the node affects the network traffic. In many cases it is more appropriate to place events on the module with the same consequences for the behavior of a particular piece of the network.

The model allows the use of only two kinds of physical node:

- DiffNod *(A)* - node dividing section into two modules,
- TracNod *(B)* - node between three modules.

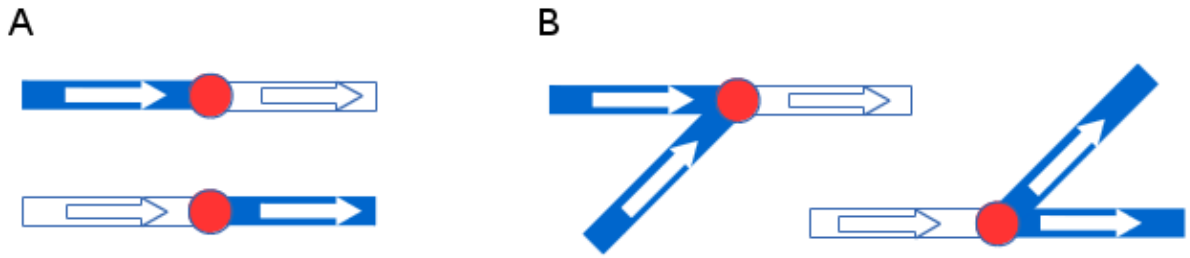


Figure 2.3: Types of model nodes

The vehicle, reaching to the node asks for further route even if has saved the whole route. Node decides whether or not to pass the vehicle.

The network node knows only its direct neighbors. However, its neighbor node can recommend its further neighbor to answer a request.

The network node can be blocked. Putting a lock on the node stops the vehicle despite the fact that the node is only a point on the route.

Network nodes decide on network traffic. All the mechanisms that determine the conduct of network traffic are introduced in the network nodes.

## 2.3 netveh

The vehicle moves in a well-defined track (see: *laufer*). Therefore, the representation simplifies to the section of the route.

Due to the fact that the network has a number of vehicles and the fact that they move on the network a vehicle is not a one simple determinated segment *(VAB)*.

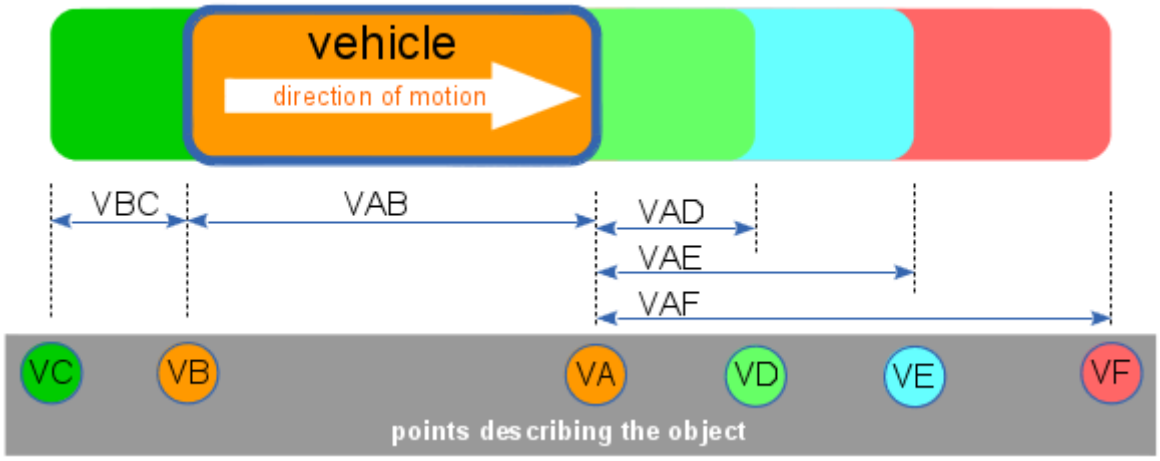


Figure 2.4: Points defining vehicle

For safety, when the vehicle is not moving, we take into account the additional distance before *(VAD)* and after him *(VBC)*. These sections should not be affected by any other vehicle. If the vehicle is moving

two addisional distances in the direction of the movement are taken into account *(VAE, VAF)* .

Always tracked are two points of the vehicle: the beginning *(VA)* and the end *(VB)*. This is due to the fact that some event are initiated by the start point of the vehicle and some by its end point. These points are therefore recorded on a regular basis. If the vehicle is moving additional points *(VE)* and *(VF)* can occur as source of events. The particular event may be related to only one point of the vehicle.

Various types of vehicles may be present in the network. Each one may have different characteristics.

Vehicles can move at different speeds. The value of the average speed on a section is always taken into account. Acceleration and braking are not recorded. Instead, the value of the average speed over some distance is revised (therefore it is important to track the *(VF)* point). This procedure is sufficient even if one vehicle catches up another.

Distances *VAD* and *VBC* are fixed and determined at the beginning of the simulation. Distances *VAE* and *VAF* are determined and updated on a regular basis. They depend, inter alia, by vehicle type and current speed.

## 2.4 events

The basic types of events in the simulation are:

- events associated with the entry on the module *(EA)* and leave it *(EB)*
- change of the operating parameters on the network section (with its lock included) - *static: (EDD)*
- section reservation for the moving vehicle (sometimes including the consequent locking of the whole module) - *dynamic: (EEE)*
- events at a point - *static: (EC)* :
  - stop
  - checkpoint

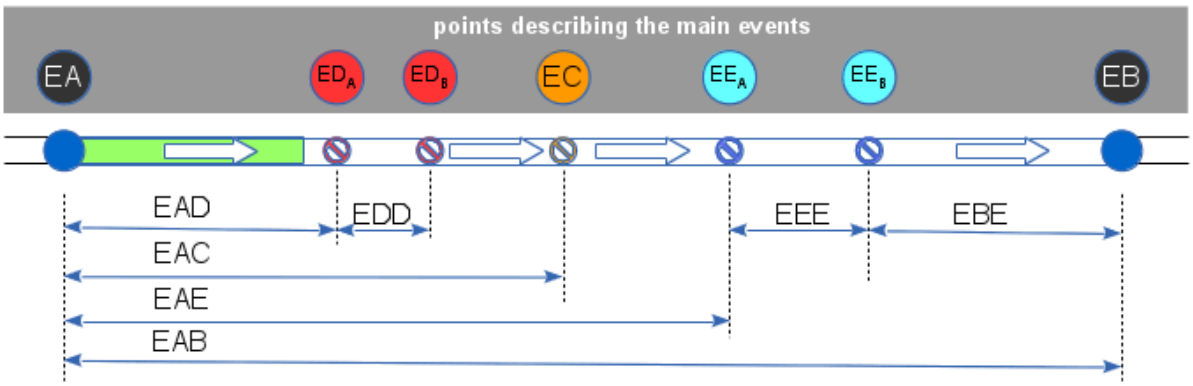


Figure 2.5: Types of events occurring on the module

Event times decide when the network status updates during the simulation.

Static events are calculated at the beginning of the simulation or during time-step initialization. Dynamic events are updated on a regular basis during the simulation.

So the cause of the event can be a vehicle and infrastructure (eg, as a result of the module EFECTS) or current simulation herself (eg, to allow monitoring).

Generally, events are logged by network modules but logging through network nodes or vehicles is also allowed.

## 2.5 netini

The simulation consists of establishing all the events in a certain period of time. The events are mainly related to the routes of vehicles. There are two main types of routes:

- route with closed loop *(A)*
- a simple route between two points *(B)*

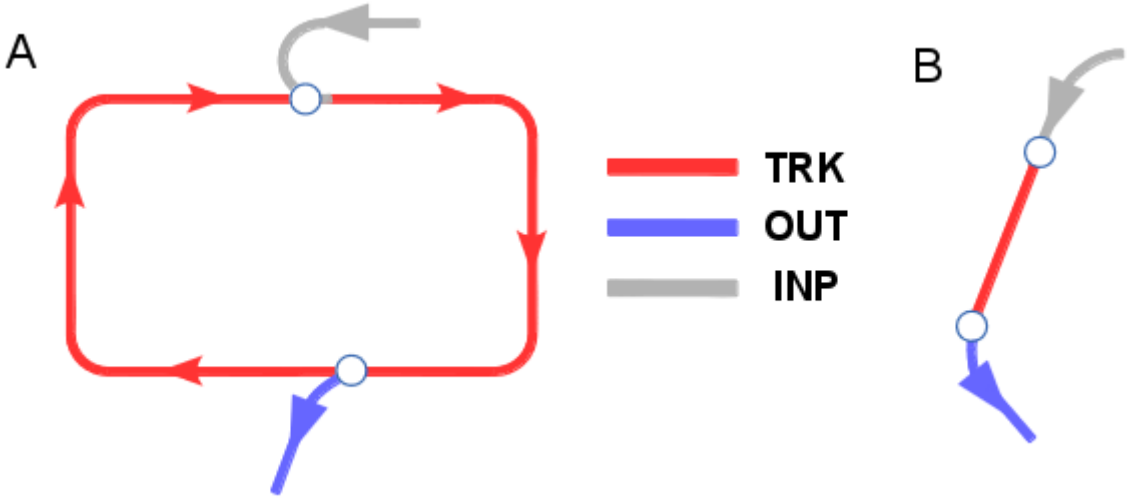


Figure 2.6: Route of a vehicle

Each of these types consists of three stages:

- introduction to the network *(INP)*,
- the movement on main route *(TRK)*
- leave the network *(OUT)* .

Point of entry to the network may be different from the place where vehicle leaves the network. Sometimes you have to take into account the earlier movement between points of entry / exit nodes before vehicle is setting off on a route.

To start moving vehicles all network objects must be pre-initialized in the correct order. It is a task of NETINI module, consisting of:

- enable the introduction of load distribution between nodes of a computing cluster
- enable the introduction of load distribution between processor cores
- creation of all network objects
- perform configuration of network objects based on the class of these objects
- enable the distribution of objects on remote hosts, and ensuring the operation synchronization with the root computing node
- supervision and execution of all other tasks related to the preparation of the network to simulate

After completing all the tasks on the preparation of a simulation module NETINI gives control over the simulation back to SIMSET module.

## 2.6 simset

SIMSET module is superior to the other. Allows you to run, control and supervise the simulation. Provides current simulation data about the status of objects modeled network. With this module, it is possible to introduce visualization during simulation.

## 2.7 efects

EFECTS module contains the whole simulation logic associated with the operation and maintenance of vehicles and infrastructure. This module calculates all the events related to the technical degradation of simulated objects.

## 2.8 loging

During the simulation informations that should be noted are generated. Important informations are generated by very different network objects and events. To unify the form of recorded informations and allow analysis based on logs LOGING module is implemented.

## 2.9 report

Data analysis is based on information collected during the simulation. REPORT module provides access to relevant information from logs and organizes the data recorded during simulation.

## 2.10 conect

Objects can be implemented in a single or multiple processes. Large load during simulation can also be the cause of a simulation run on a cluster.

To ensure communication during simulation between objects regardless of differences in:

- architecture
- runtime environment

and to allow parallel processing CONECT module is introduced.

CONECT module is used whenever there is a need for the exchange of information between objects in the simulation. This allows objects do not have to deal with an appropriate addressing of other network objects.

This module allows communication in each case, even the exchange of data between processes on different hosts can be implemented.

## 2.11 record

The purpose of the module is to mediate in writing and reading information during the simulation. For network objects does not matter what is used to read and write the data. These can be files, database, or you can use an ORM.

CHAPTER 3

# Attachments

## 3.1 Glossary

**event** .... (see: *Global network events*, *Events and network nodes*, *Time and synchronization of events in network*)

**laufer** ... (see: *netveh*, *Speed (Vehicles)*)

**node** .... (see: *Events and network nodes*, *netnod*, *The impact of the environment on the network*)

**sector** ... (see: *netmod*)

**time-frame** .... (see: *Time and synchronization of events in network*)

**time-step** .... (see: *Simulation steps*)

## 3.2 Releases, changes, roadmap

This document applies to version 0.1.0 (release: 0.1.11) of the project.

### 3.2.1 v0.5.x

- Tools for installation testing:
  - Input network definition examples
  - Input simulation description examples
  - Ipython sessions examples

### 3.2.2 v0.4.x

- Ipython session example for user's manual
- Improving user's manual.
- Bugfixes and code optimization.

### 3.2.3 v0.3.x

- First full functional and executable code.

### 3.2.4 v0.2.x

- Release of all objects and basic functionality description.
- Source code is still hidden.

### 3.2.5 v0.1.x

- The model assumptions are established.
- The library functionality is full defined.
- The structure of the project and library is stable.
- Initialization and configuration of project tools.

## 3.3 Python modules

*GENERATE: modindex*

## 3.4 Index

*GENERATE: genindex*

## c



## e



## l



## n



## r



## s