Implementation of Mobility Management Methods for MANET

Pavel Vajsar, Jiri Hosek, Milan Bartl and Karol Molnar

***Abstract*—The Mobile Adhoc Networks represent very perspective way of communication. The mobility management is on the most often discussed research issues within these networks. There have been designed many methods and algorithms to control and predict the movement of mobile nodes, but each method has different functional principle and is suitable for different environment and network circumstances. Therefore, it is advantageous to use a simulation tool in order to model and evaluate a mobile network together with the mobility management method. The aim of this paper is to present the implementation process of movement control methods into simulation environment OPNET Modeler based on the TRJ file. The described trajectory control procedure utilized the information about the route stored in the GPX file which is used to store the GPS coordinates. The developed conversion tool, implementation of proposed method into OPNET Modeler and also final evaluation are presented in this paper.**

***Keywords*— GPS, GPX, MANET, mobility, OPNET Modeler, simulation, software.**

I. Introduction

AD-HOC networks recently represent one of the most attractive research fields in telecommunication technology. The wireless ad-hoc network is a decentralized network that does not rely on any pre-existing communication infrastructure. Each wireless node is capable of movement and can participate in the routing process by forwarding data to other neighbouring nodes [1], [2].

Mobile Ad-hoc Network (MANET) is one of the technologies that belong to the wireless ad-hoc networks. MANET represents a system of wireless mobile nodes which can freely and dynamically self-organize into a dynamic network with temporary topology allowing users and devices internetwork seamlessly [3].

Because, the MANET is a communication system consisting of mobile nodes, one of the important factors in the simulation of these networks is the ability to control the movement of mobile nodes during the simulation. For this reason, the mobility control of individual nodes or entire networks becomes a key factor for the testing and simulation of communication systems [4]. While keeping the functionality of mobile nodes and networks, wireless communication allows changing their position based on the predefined trajectories, orbits and randomly selected routes [5].

The possibility to control the movement of mobile node allows more effective prediction and scheduling of network sources for individual stations, such as handover optimization in MANETs.

Manuscript received November 1, 2012. Manuscript accepted December 4, 2012. This work has been supported by the project CZ.1.07/2.3.00/30.0005 of Brno University of Technology.

Pavel Vajsar[1], Jiri Hosek[2] and Milan Bartl[3] are with Department of Telecommunications, Brno University of Technology, Brno, Czech Republic (e-mails: pavel.vajsar@phd.feec.vutbr.cz[1], hosek@feec.vutbr.cz[2], bartl02@stud.feec.vutbr.cz[3]).

Karol Molnar is with the Honeywell International, s.r.o. (e-mail: Karol.Molnar@honeywell.com).

II. Trajectory control in OPNET Modeler

In the environment of OPNET Modeler (OM), there are several ways to control the trajectory of mobile node. Firstly, the type of trajectory is chosen and then its assignment to the selected node is defined. There are more trajectory types and ways of assignment in the simulation environment OPNET Modeler, see following chapters [6].

A. Segment-based trajectory

In order to exactly determine the trajectory of mobile node in OM, user can use the segment-based trajectory. Thus defined trajectory consists of one or more traversal-time values and a set of three-dimensional (x, y, altitude) coordinates that define the mobile node's path. In addition, trajectories with variable-length segments have a set of angles (roll, pitch, yaw) that define the mobile node's orientation in space. Segment-based trajectories are stored in ASCII text files with a *.trj extension. For proper functionality of so defined trajectory, it is necessary to place the *.trj file into a folder created by the project and then assign this file with defined trajectory to a mobile node or network using the "trajectory" attribute [6].

1) Fixed-interval trajectory

Segment-based trajectories come in two varieties: fixed-interval and variable-interval. In a fixed-interval trajectory, only one value determines the traversal time for all segments, hence a node takes the same amount to time to traverse every segment, regardless of the segment's length. In addition, a single value generally determines the altitude for all points. The *.trj file has following structure:

```
<coordinate_count>
locale: <locale>
<sample_time_step> <position_unit> <coordinate_method>
<x_coord_0>, <y_coord_0>, <alt_0>
<x_coord_1>, <y_coord_1>, <alt_1>
...
<x_coord_n>, <y_coord_n>, <alt_n>
```

2) Variable-interval trajectory

In a variable-interval trajectory, each point has its own specified altitude, wait time, traversal time (time from the previous point to the current point), and orientation. The wait

time causes a mobile object to pause at each point before it begins traversing the next segment. The *.trj file for variable-interval trajectory has following structure:

```
Version: 3
Position_Unit: <position_unit>
Altitude_Unit: <altitude_unit>
Coordinate_Method: <coordinate_method>
locale: <locale>
Calendar_Start: <start_time>
Coordinate_Count: <coordinate_count>
# X Position ,Y Position ,Altitude ,Traverse Time, Wait
Time, Pitch, Yaw, Roll
<x_coord_0>,<y_coord_0>,<alt_0>,<trav_time_0>,<wait_time
_0>,<pitch_0>,<yaw_0>,<roll_0)>
<x_coord_1>,<y_coord_1>,<alt_1>,<trav_time_1>,<wait_time
_1>,<pitch_1>,<yaw_1>,<roll_1)>
...
<x_coord_n>,<y_coord_n>,<alt_n>,<trav_time_n>,<wait_time
_n>,<pitch_n>,<yaw_n>,<roll_n)>
```

3) Relative movement

Because the mobile nodes can be nested (e.g. a mobile node within a mobile sub-network), the movement of the parent subnetwork can affect the movement of the child node. It must be considered when the segment-based trajectories are created. The effect on movement is as follows [6]:

- If the child node's position units are degrees, movement is considered in an absolute sense relative to the earth. Parent subnetwork movement has no effect on the child node.
- If the child node's position units are non-degree (such as meters or feet), movement is relative to the parent subnetwork. Thus, if the parent subnet is moving east at 10 meters per second (m/s) and the child node within it is moving west at 5 m/s, the motion of the node with respect to the earth is 5 m/s east.

4) Defining segment-based trajectories

The segment-based trajectory can be created and assigned manually in OPNET Modeler. The procedure for defining a trajectory varies slightly depending on whether the trajectory uses fixed intervals or variable intervals.

B. Random trajectory

Trajectories and orbits specify deterministic paths for mobile nodes. Random trajectory is defined by a rectangular region in which a node will move during a simulation. This region is specified by x-y coordinates or by using a mobility domain. During simulation, the node randomly selects a destination in the region and moves toward it at a specified or randomly chosen speed. Upon reaching its destination, the node pauses a configurable length of time before it repeats the process by selecting another random destination [6].

C. Direct manipulation of position attributes

In addition to trajectories and orbits, OPNET Modeler users can model movement of mobile sites by directly manipulating position attributes. If a trajectory is specified for a mobile node, the path of that site is predetermined for the entire simulation. However, if no trajectory is specified, the node's position can be directly updated by any process during simulation. A mobile node's x position and y position attributes specify its location in its parent subnetwork. A mobile node's altitude attribute specifies its elevation relative to sea level, the underlying terrain, or the parent subnetwork (depending on the site's altitude modeling attribute setting). A change to one of these attributes will cause an immediate change in the location of the mobile site [6].

Typically, one of two techniques is employed to dynamically change the location of a mobile node. In both cases, a user-defined process is responsible for modifying the position attributes of a mobile node. The first technique is a centralized approach, in which one process is responsible for updating the positions of all of the mobile nodes in a network model. The second technique is a decentralized approach. In the sense that each mobile node has a process executing within it that updates only its own position [6].

III. GPX AS INPUT DATA

The real path is obtained using the GPS (Global Positioning System) system and stored in the GPX (GPS eXchange Format) file subsequently. This file type can be captured from the navigation device or online maps servers. This GPX file with defined path is passed to the OPNET Modeler environment, where it is converted by special functions into a format that meets the requirements of OM for movement control.

A. GPX file format

The GPX file format is based on XML (eXtensible Markup Language), which means that the file contains predefined tags to represent the whole trajectory [7]. The structure of a GPX file format is described in Table I.

TABLE I
STRUCTURE OF THE GPX FILE FORMAT

XML Element	Description
<?xml version="1.0" standalone="yes"?>	File header placed at the beginning of the document.
<gpx>	Tag identifying GPX files.
<trk>	Tag representing a track.
<trkseg>	Tag containing the list of points representing the track.
<trkseg lat="49.5684025" lon="14.0124586" >	Tag containing the GPS coordinates of the point.

1) Haversine formula

One of the most important operations during the conversion of a GPX file to an OPNET Modeler trajectory file is to calculate the distances between two consecutive points of the track. This calculation is realized according to the mathematical equation called haversine formula [8]. This formula is defined as:

$$haversin\frac{d}{R} = haversin(\varphi_1 - \varphi_2) + \cos(\varphi_1) + \cos(\varphi_2) + \cos(\Delta\lambda), \quad (1)$$

where

- *haversin*() is the haversin function:

$$haversin(\theta) = sin^2\left(\frac{\theta}{2}\right) = \frac{1-\cos(\theta)}{2} \quad (2)$$

- d is the distance between two points,
- R is the diameter of the Earth,
- φ_1 is the latitude of the first point,
- φ_2 is the latitude of the second point,
- $\Delta\lambda$ is the difference in longitude of two points.

IV. Implementation of Online Conversion Tool

Detailed knowledge of both formats (GPX and TRJ) taking part in conversion is compulsory, as well as painstaking validation of perfection of the conversion process. At first, a web interface was developed for conducting this process. This interface was capable to operate with desired formats and provided more comprehensible environment than OPNET Modeler.

The first stage of the work was accordingly focused on implementing an application for conversion of GPX files to TRJ format which is supported by OM for the sake of insertion and testing of a real trace within OM environment. As for technological solution, the simplest approach has been chosen. The entire system is designed as a web application. The basic architecture of this application is shown in Fig. 1.

The user interface was programmed using plain HTML (HyperText markup Language) without any additional libraries or frameworks. For mere input validation was used simple JavaScript. The very conversion and parsing is conducted on the server side. UI provides options for choosing a file to be converted and uploads this file to the server where all the necessary operations are performed. A technology of PHP (Hypertext Preprocessor) was used to develop the server side of the application.

PHP scripting language was chosen for several reasons. GPX files are based on XML (eXtensible Markup Language) language and are practically identical. SimpleXML libraries were used for the reason of being supported by majority of contemporary webhosting providers. This means that no required installation or additional libraries, which results in simplification of development of the application. Another reason may be an extension like Google Maps API or any other map basis that may stand for data source in hypothetical extension of the application, as well as viewing a trace stored in a GPX file. At least but not last, in case of emplacement on public web page, the application could be available to all Internet users.

A. PHP Scripting Language

The PHP (Hypertext Preprocessor) is a widespread multipurpose scripting language. Especially, it is used for web application development, so it consequently supports an encapsulation into HTML. When used with dynamic web pages, scripts are being processed at server side and mere results of this process are transmitted to the target browser.

B. JavaScript

The JavaScript is multiplatform object-oriented scripting language. It is used, nowadays, mostly as programming language for web pages. Its code is often inserted directly into HTML code of a page. JavaScript usually controls interactive elements like buttons or special text fields, or can be used for creating animations and image effects.

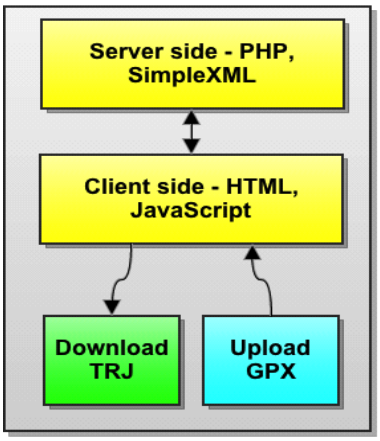


Fig. 1. Conversion application architecture

Word Java in the title of this language is of mere marketing character, though, and the language itself has nothing in common with the Java language besides similar syntax.

Unlike the other programming languages (e.g. PHP or ASP), JavaScript code is usually processed after the page is loaded in a browser (i.e. at the client side). This fact results in several security restrictions for protection of user's privacy. For example, JavaScript cannot process files stored on the disk [11].

C. Conversion Interface

Individual inputs of the application are controlled by JavaScript so that only a number can be inserted and a blank value is invalid. In a case when user disables JavaScript in his web browser, the inputs are subsequently checked by PHP as well. The graphical user interface of the application is shown in following Fig 2.

In the first stage, the application loads selected file and validates its appendix (must be .gpx). Size of the file is checked afterwards. Its maximal value is set for 10MB. The file is subsequently stored at the server side in a directory named gpx. Further, the file is loaded by function simplexml_load_file. This function converts the file into an object in memory. This step also includes validation of structure of the file, whether it corresponds with XML/GPX standards. With these requirements fulfilled, the object stored in memory is searched for an element <trkpt>. This element contains parameters of geographical latitude and longitude. These parameters are then stored in one field. If the object contains further information about time or altitude, these data are stored in the field as well. After storing all points of the track into the field, a count of these points is computed. Furthermore, if background is set on Campus, values in the field are converted from angular coordinates to coordinates of selected background. With a check button Center checked, a center of the trace is placed exactly into the middle of the coordinate system.

A new file *trajectory.trj* is created subsequently in a directory called export. A header is inserted into this file according to chosen metrics of the trace and to the count of trace points computed earlier. After this import, the field containing coordinates, time and altitude is read and the values are written into the file in a format specific to TRJ files.

During the last stage the old file from the *gpx* directory is deleted and the new TRJ file is available to download.

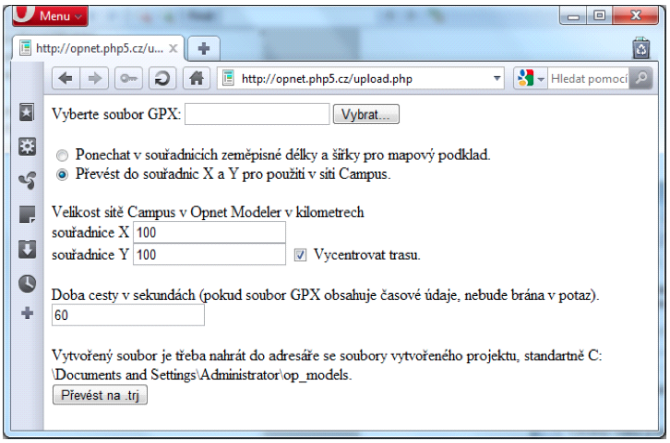


Fig. 2. Conversion application – graphical user interface

V. GPX TO TRJ CONVERSION IN OPNET MODELER

The C/C++ programming language can be used in OPNET Modeler to implement new and modify existing functions. All functions related to the work with GPX format have been stored in the *gpx.h* header file. Using the *#include "gpx.h"* command this header is inserted to the Header Block of the process model created in OPNET Modeler [6].

The coordinates obtained from the GPX file are stored in a structure containing 4 parameters, as it is shown in the following fragment of source code. The first two parameters (latitude and longitude) represent the location, the third parameter defines the altitude and the fourth parameter stores the timestamp.

```
struct coordinate {
        double longitude ;
        double latitude ;
        double altitude ;
        int time ;
};
```

The first step of converting a GPX file to a TRJ format is loading the GPX file from the given directory. Next the file must be opened for reading and processed by a XML parser, e.g. by functions available in the TinyXML library. The data obtained will be inserted into a vector. In our model this process is realized by the *CoordToVector* function:

```
vector < coordinate > CoordToVector ( char gpx_file
[128] , bool isTimeMan, double timeMan , double speed ,
bool * err )
```

This function is directly called from the process model. The input parameters of the function are the following:

- **gpx_file** – the name of the GPX file which has to be loaded.
- **is_time_manual** – a flag, which takes value 1 if the time is configured manually via model attributes or 0 if the time is obtained directly from the GPX file.
- **travel_time_manual** –manually defined time value, entered via an attribute of the model.
- **speed** - manually defined speed of movement, entered via an attribute of the model.
- ***err** – a pointer type flag, which takes value 1 if an error occurs during the process, i.e. if the file with given name is not present or if the format is not supported. Consequently this flag terminates the simulation in the process model, as it is described later.

By executing the CoordToVector function, it creates a vector of coordinate structures, defined earlier.

```
Vector< coordinate > v;
```

This vector is the return value of the CoordToVector function and contains the coordinates of each point together with its altitude and timestamp. In the next step the vector is converted into a TRJ format. This conversion is provided by the GPXtoTRJ function.

```
void GPXtoTRJ ( char site_name [128] , vector <
coordinate > v , bool * err )
```

The input parameters specify the vector of coordinate values and the name of the target TRJ file. First, the header is inserted into the file. Next the input vector is processed step by step and the values calculated are entered into the TRJ file. The timestamp of the first entry equals to 0. The timestamp of the following entries equals to the difference between the timestamp of the current and previous entry.

The TRJ file also contains information about the total number of coordinates. This parameter is required during the configuration of a mobile node in the OPNET Modeler environment. The total number of coordinates is determined by the CountFunc function:

```
int CountFunc ( vector < coordinate > v ),
```

which is called by the vector of coordinates as an input argument. Additionally, the length of the whole trajectory is calculated. This calculation is provided by the *DistanceInMeters* function:

```
double DistanceInMeters ( const coordinate & from ,
const coordinate & to )
```

This function returns a value of type double, which contains the length of the whole track in meters. The distance is calculated according to the haversine formula introduced in section 3.

VI. SIMULATION PROCESS IN OPNET MODELER

Firstly, it is necessary to set up the simulation environment after the implementation of functions for the conversion of GPX file into TRJ file. The first step is the import of designed functions into environment. This process is important and allows using new functions during the simulation process. The next step is the design of simulation scenario, in which nodes trajectory will be changed according to GXP file.

A. *Set up OPNET Modeler simulation environment*

All necessary functions for the conversion of GPX file into TRJ file are implemented in header file called gpx.h. The OPNET Modeler simulation environment can read standard header files from this location:

```
\{INSTALL_DIR}\{OPNET_VERSION}\models\sd\include
```

Therefore, the gpx.h file can be stored into this location or a folder, which contains the gpx.h file, can be linked into simulation environment through the menu item Edit-Preferences-Compilation Flags for All Code. This menu item contains:

```
/ W3 / D _ C R T _ S E C U R E _ N O _ D E P R E C A T E
/ IC :\ {INSTALL_DIR} \ {OPNET_VERSION} \ models \ std \
include
```

To add another record, it is necessary to modify the menu item. The value PATH_TO_FOLDER will be replaced by real path to folder, which contains gpx.h file. This modification defines another search path for header files.

```
/ W3 / D _ C R T _ S E C U R E _ N O _ D E P R E C A T E
/ IC :\ {INSTALL_DIR} \ {OPNET_VERSION} \ models \ std \
include / IC : \ {PATH_TO_FOLDER}
```

B. *Initial configuration of simulation scenarios*

Firstly, a new project has to be created in OPNET Modeler. The creation of the mobile node's model is the necessary step. This node's model can by obtained by duplicating the default model. The duplicated model of mobile node was modified and extended for ability to manage its own trajectory, which depends on the GPX file. The duplication of mobile node's model is necessary, because the modification of the existing mobile node's model would be reflected in all applications of this model, i.e. in other independent simulations.

The duplicated mobile node's model is extended by new processor, which is called Mobility. This processor is based on standard processor called Sink, which contains two states - Init (forced) and Discard (unforced), see Fig. 3.

C. *Extended attributes of mobile node*

In the OPNET Modeler environment, it is possible to extend existing objects attributes by new custom attributes. The aim is to manage nodes movement by the route defined in GPX file, therefore it is necessary to extend the node's attributes by few new attributes:

- **GXP File** - specifies the name and path of GPX file which defines the route.
- **Manual Time** - the time can be entered manually (enable) or can be used from GPX file (disable). If this attribute is disabled and GPX file does not contain information about time, then the time 1s is used for a transition from one point to another.
- **Travel Time** - total time (in minutes) required for completion of the journey.

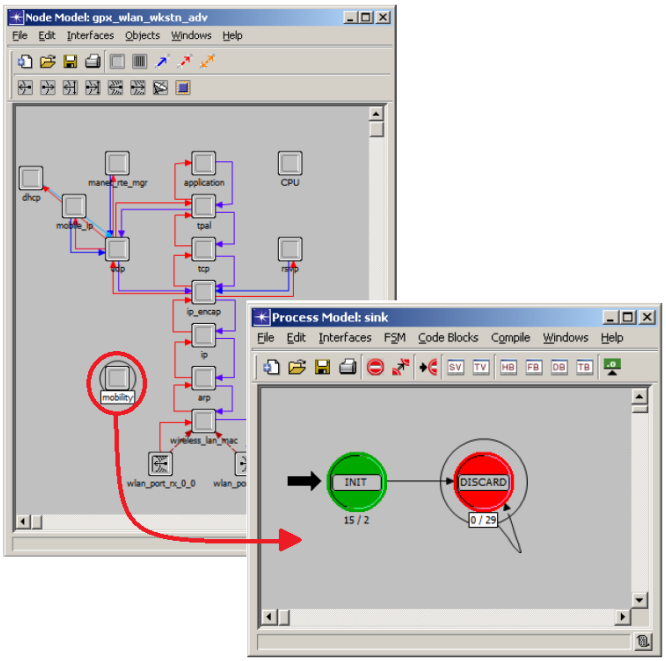


Fig. 3. Mobility processor and child process model

- **Speed** - speed of movement in km/h.
- **Generate TRJ** - if this attribute is enabled, TRJ file is created during simulation, which can be displayed in OPNET Modeler environment or it can be reused for another simulation process. If value is set up to disable, TRJ file is not created.
- **Movement Report** - determines whether the reports about mobile node's movement will be displayed in the OPNET simulation console.

These attributes can be set during the mobile node configuration in user interface part of OPNET Modeler or can be set by program during the simulation process. New attributes can be seen in Fig. 4.

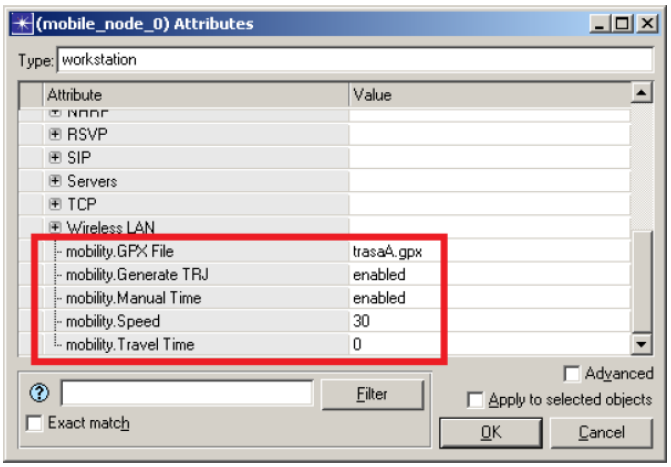


Fig. 4. New defined attributes for mobile node

D. *Extending process model*

The process model described above was based on a standard process model called Sink. The Sink process model is not

suitable for a trajectory management, therefore the Sink process was extended by an unforced state called Coordinates. It can be seen in Fig. 5.

1) INIT state

The INIT state is an initial state in every process model in OPNET Modeler environment. This state contains operations to get node identifier (ID), nodes attributes and to set coordinates vector.

```
my_obj_id = op_id_self();
parent_obj_id = op_topo_parent(my_obj_id);
op_ima_obj_attr_get (my_obj_id , "gpx file", site_gpx);
…
mymap[parent_obj_id]=CoordToVector(site_gpx,  time_manual
,travel_time , speed , &err );
distance = PathDistance ( mymap [ parent_obj_id]);
count = CountFunc(mymap[parent_obj_id]);
GPXtoTRJ (site_name, mymap[parent_obj_id], &err);
```

At first, an ID of the Mobility process is loaded using *op_id_self()* function. After that is loaded an ID of the parent process. Subsequently, appropriate attributes of the mobile node are loaded using function *op_ima_attr_get()*. In the example above there is only a sample showing loading of the gpx attribute. Other attributes are loaded similarly. Function *CoordToVector()* is called afterwards. It fills the *mymap* vector with coordinates. Following functions determine length of the loaded route and count of coordinates of the entire route.

If the attributes of the node are set to create TRJ file, fiction *GPXtoTRJ()* is called. Its output is a TRJ file, which is located in the project folder. After performing all tasks program transits into the Coordinates state.

2) COORDINATES state

This state browses the vector of coordinates that was created in the INIT state. These coordinates are assigned to the mobile node. Assignment of individual coordinates is performed by the function called *op_ima_obj_attr_set_db1():*

```
op_ima_obj_attr_set_dbl ( parent_obj_id , "x position ",
mymap[parent_obj_id][i].longitude);
```

In the case where the time value is loaded from the GPX file, calling the function *TimeDiffFunc()* is necessary:

```
timeD [ parent_obj_id ] += TimeDiffFunc (i, mymap [
parent_obj_id ]);
```

The function mentioned above returns a difference of times between two points. The acquired time value is incremented into the *timeD* variable at each pass through the *Coordinates* state. In the case where the time value is set manually through station's attributes, the *timeD* value is acquired differently.

```
timeD[parent_obj_id] += mymap[parent_obj_id][i-1].time;
op_intrpt_schedule_self(timeD[parent_obj_id], 0);
```

The function *op_intrpt_schedule_self()* is a standard function of the OPNET Modeler environment. It performs an interruption of the state at the very time that is saved in the *timeD* variable. This results in mobile node's transition to the desired location at the requisite time. This procedure repeats until there are no coordinates available.

3) DISCARD state

This state performs actions associated with terminating functions of the process model and thus terminating the superior Mobility processor element. From the simulation point of view, the DISCARD state provides termination of station's movement.

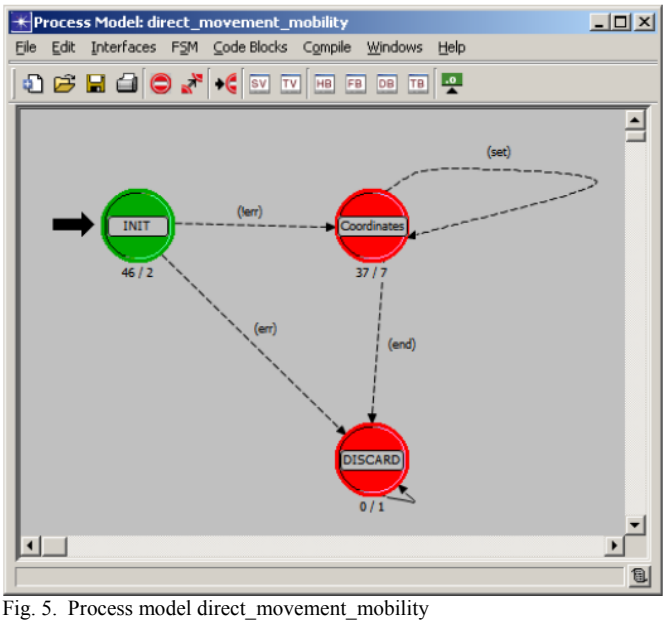


Fig. 5. Process model direct_movement_mobility

VII. ANALYSIS OF THE SIMULATION RESULTS OBTAINED

The simulation was performed for two mobile nodes. Each mobile node obtained slightly different settings. The adapted model, supplemented with the new Coordinates state and process parameters, was used as the mobile node's model. Settings of parameters of individual mobile nodes correspond to the following Table II.

TABLE II
PARAMETERS OF INDIVIDUAL MOBILE NODES

Attribute	mobile_node0	mobile_node1
GPX file	traceA.gpx	traceB.gpx
Generate TRJ	enabled	disabled
Manual Time	enabled	disabled
Speed	20	0
Travel Time	0	0

The OPNET Modeler environment contains an advanced debugger that allows observation of chosen simulation parameters, simulation progress control and display an animation of individual object's movement. The movement animation is an essential simulation parameter. It allows monitoring of the mobile node if it follows the trace that was defined in the GPX file assigned (see Fig. 6).

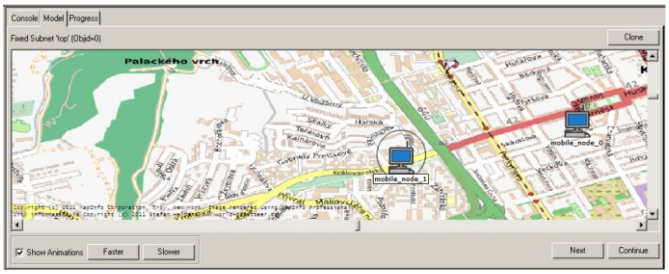


Fig. 6. The sample of mobile node's movement around the map background based on a trajectory defined in a GPX file

VIII. Conclusion

In the beginning, the paper states options of station model's control in the OPNET Modeler environment. Particularly, a TRJ file and its structure is essential. The file provides trajectory control and the possibility of moving the station using direct manipulation of the attributes. The next part of the paper deals with implementation of functions for a real trajectory simulation. All these functions are implemented in an individual file and create an interface between the GPX file defining a real trajectory and the OPNET Modeler environment. This system enables to control movement of a station around a real trajectory. This was impossible in the OM until now. It is possible to use more fractional GPX files for a trajectory definition. The functions defined in the OM are loading fractional files and connecting individual trajectories into one continuous trajectory. The station moves around the resultant trajectory, then. This solution brings new opportunities for network simulations to get closer to the real environment.

References

[1] ILLYAS, M. The Handbook of Ad Hoc Wireless Networks, CRC Press, 2003.

[2] BOUKERCHE, Z. Algorithms and Protocols for Wireless and Mobile Ad Hoc Networks, Wiley, 2009.

[3] SKOŘEPA, M.; ŠIMEK, M.; MAHDAL, O. Mobile Ad-Hoc Network routing protocols - performance analysis. In Proceedings of the 32nd International Conference Telecommunications and Signal Processing. 2009. s. 1-4. ISBN: 978-963-06-7716- 5.

[4] CAMP, T.; BOLENG, J.; DAVIES, V. A survey of mobility models for ad hoc network research. In Wireless Communications and Mobile Computing, 2002, vol. 2, no. 5, pp. 483-502.

[5] HOŠEK, J.; MOLNÁR, K.; JAKÚBEK, P. Map-Based Direct Position Control System For Wireless Ad- Hoc Networks. In Proceedings of the 34th International Conference on Telecommunication and Signal Processing, TSP 2011. Budapest: Asszisztencia Szervezo Kft., 2011. s. 195-200. ISBN: 978-1-4577-1409- 2.

[6] OPNET Technologies. OPNET Modeler Product Documentation Release 16.1 .2011.

[7] FOSTER, D. GPX: the GPS Exchange Format, [online]. 2004. URL: http://www.topografix.com/gpx_for_developers.asp.

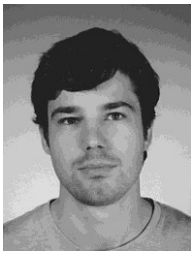
[8] GIS FAQ Q5.1: Great circle distance between 2 points, [online]. 2010. URL: http://www.movable-type.co.uk/scripts/gis-faq-.1.htm.

[9] PHP: Preface - Manual [online]. 2010. URL: http://cz.php.net/manual/en/preface.php.

[10] Hrebenar, Jiří. SimpleXML - jednoduše na XML v PHP 1.díl [online]. 2009. URL: http://programovani.blog.zive.cz/2009/12/simplexml-jednoduse-na-xml-v-php-1dil.

[11] About JavaScript - MDC Doc Center [online]. URL: https://developer.mozilla.org/en/JavaScript/AboutJavaScript.

Pavel Vajsar completed the Master degree at Faculty of Electrical Engineering in Communications and Informatics specialization in Brno University of Technology. He is currently 4th-year postgraduate student at the Department of Telecommunications of the same faculty. His research work has been concentrated on routing in MANET networks with regard on quality of services. Recently he has also been concerned with wireless sensor networks and developing of application for monitoring of these networks.



Jiri Hosek received the B.S. and M.S. degrees in Electrical Engineering from Faculty of Electrical Engineering and Communication at the Brno University of Technology in 2005 and 2007, respectively. He is currently an assistant professor at the Department of Telecommunications of the Faculty of Electrical Engineering and Communication at the same university. His research work has been concentrated on the design of new communication protocols and services for the wireless networks.

Milan Bartl is currently a 3rd-year master student at the Department of Telecommunications of the Faculty of Electrical Engineering and Communication, BUT. His master thesis is focused on the issue of cooperation between external systems and simulation environmet OPNET Modeler and its utilization in QoS assurance area.

Karol Molnar received his MSc. degree in Electronics and Communications (1997) and Ph.D. degree in Teleinformatics (2002) at Brno University of Technology (BUT), Czech Republic. He is with the Honeywell International, s.r.o. In his scientific work he focuses on modern network technologies, especially on topics of QoS support in both fixed and mobile network technologies. During the last several years he actively participates in theoretical and research works closely related to the technology of Mobile Adhoc Networks.