DE GRUYTER OPEN

HUNGARIAN JOURNAL OF INDUSTRY AND CHEMISTRY
Vol. 44(2) pp. 85–91 (2016)
hjic.mk.uni-pannon.hu
DOI: 10.1515/hjic-2016-0010

HUNGARIAN JOURNAL OF INDUSTRY AND CHEMISTRY
HJIC

SIMULATION OF ELECTRICAL GRID WITH OMNET++ OPEN SOURCE DISCRETE EVENT SYSTEM SIMULATOR

MILÁN SŐRÉS AND ATTILA FODOR*

Department of Electrical Engineering and Information Systems, University of Pannonia, Egyetem út 10., Veszprém, 8200, HUNGARY

The simulation of electrical networks is very important before development and servicing of electrical networks and grids can occur. There are software that can simulate the behaviour of electrical grids under different operating conditions, but these simulation environments cannot be used in a single cloud-based project, because they are not GNU-licensed software products. In this paper, an integrated framework was proposed that models and simulates communication networks. The design and operation of the simulation environment are investigated and a model of electrical components is proposed. After simulation, the simulation results were compared to manual computed results.

Keywords: energy transmission, electrical networks simulation, distributed energy systems

1. Introduction

The simulation of electrical networks is important before network planning, development, servicing, etc. is conducted. There are many planning and simulation software solutions on the market, which can simulate electrical networks and grids, e.g. MATLAB, EPLAN, WSCAD.

The problem with commercial simulation software is that such software packages cannot be ported to a new system. For example, if the development of a cloud-based electrical network and grid simulation system is required, standard simulation software products and methods cannot be used. To solve this problem, a simulation engine was used, which possesses a GNU licence. The OMNeT++ Discrete Event Simulator (DES) [1] was chosen. Mets *et alia* [2] have previously used the OMNeT++ simulator environment.

OMNeT++ is an extensible, modular, component-based C++ simulation library and framework, primarily for building network simulators. OMNeT++ is not supported directly by the simulation of an electrical grid. To solve this incompetency, a model of the most important electrical components was constructed and as a result, the general simulation engine can be used for cloud-based electrical grid simulation. Two methods were implemented in our OMNeT++ software, which investigated their mathematical foundations, as well.

*Correspondence: fodor.attila@virt.uni-pannon.hu

2. Simulated Network

According to load distribution the electrical distribution system can be classified as:

- Fed at one end with one load at the other end;
- Fed at one end with more loads;
- Fed at both ends;
- Radial;
- Ring;
- The distribution system.

To test the OMNeT++ and model of the developed components, a network architecture was chosen, which is convenient and compatible with normal methods for planning of electrical networks. A system has been simulated, which is fed at both ends and consists of three loads (*Fig.1)*.

Of course, this OMNeT++ simulation project can simulate any distribution systems, though exact values will only be calculated in this case. To simplify our calculations, the wire parameters shown in *Table 1* were

Table 1. Wire parameters

ϱ	$1.85 \cdot 10^{-8}$ Ωm
q	1 mm^2
l_1	27.027 m
l_2	54.027 m
l_3	27.027 m
l_4	27.027 m

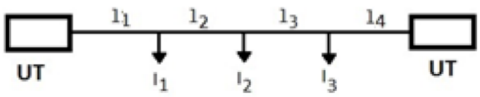

Figure 1. Distribution system fed at both ends with three loads.

considered.

The resistances of all sections of wire are computable using *Eq.(1)*. The calculated resistances of the sections of wire are $R_1 = R_3 = R_4 = 1\ \Omega$ and $R_2 = 2\ \Omega$, which were used in the OMNeT++ simulation.

$$R = 2\,\varrho\, l / q \qquad (1)$$

The selected resistance values are far from the resistance of wire used in real distribution systems, but the calculations are simplified and the illustration of the results more obvious. The values of current loads for each load were $I_1 = 10$ A, $I_2 = 16$ A, and $I_3 = 5$ A.

2.1. Classical Method of Calculating the Voltage Drop

First, the voltage drop of a distribution system was simulated, in order to calculate the voltage drops using the classical method. Our system consists of three loads and it is fed from both ends.

The current of the first and second feeding points should be I_I and I_{II}, respectively. By applying Kirchhoff's Law, *Eq.(2)* is defined as:

$$I_I + I_{II} = I_1 + I_2 + I_3\ . \qquad (2)$$

The total length of the wire is calculated as:

$$\Sigma\, l = l_1 + l_2 + l_3 + l_4\ . \qquad (3)$$

If I_I and I_{II} are known and the load currents are subtracted from one of them, a load that is fed by both ends is identified [4, 5]. Afterwards, the electrical network can be separated to obtain two networks fed at one end. The method of calculating, for example, I_I is as follows:

$$I_I\, \Sigma\, l = I_3\, l_1 + I_2\, (l_2 + l_3) + I_1\, (l_4 + l_3 + l_2)\ . \qquad (4)$$

Using *Eqs.(2)* and *(3)*, the results of the calculations of the total current, currents of feeding points, and the length of the wire are $I_1 + I_2 + I_3 = 31$ A, $I_I = 15.4$ A and $I_{II} = 15.6$ A, and $\Sigma\, l = 135$ m, respectively. From I_I and I_{II}, it can be determined that the voltage drops accotding to Ohm's Law. The voltages of the loads are $U_1 = 214.6$ V, $U_2 = 203.8$ V, and $U_3 = 214.4$ V.

2.2. A Method of Calculating the Voltage Drop Based on the Node-Potentials

The previously presented method can be easily used to calculate networks consisting of topology fed at one end as well as at both ends. However, our electrical grids are obviously not that simple, see *Fig.2* or they can even be more complex. Furthermore, in this kind of method implemented using OMNeT++ the presence of a small solar plant on a rooftop is hard to handle.

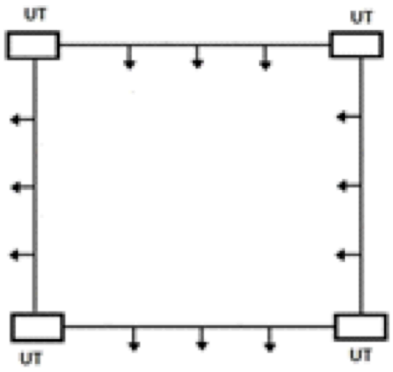

Figure 2. A more complex electrical network than shown in *Fig.1*.

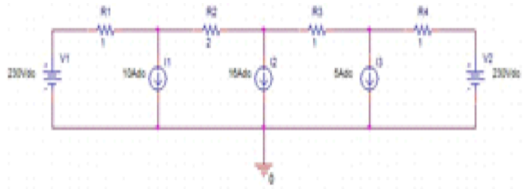

Figure 3. The network fed at both ends modelled with electrical elements.

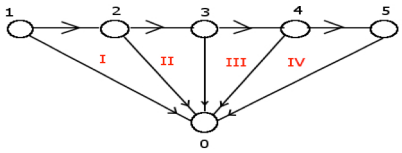

Figure 4. Directed graph of the modelled circuit.

Another method of calculating the voltage drops and currents was identified. Using the node-potential method, any parameter of an electrical network can be calculated. To apply such a method, the feeding points with voltage sources, the loads with current sources, and the wires with resistances were modelled (*Fig.3*).

With this method the feeding points are modelled with voltage sources exhibiting constant voltages, which results in a crucial consequence for more complex networks. If another feeding point is added to the system, some part of it or even the whole network will be parallel to the new feeding point, as it is directly connected to the ground. That would make the investigation of the system and the handling of complex grids easier. Of course this is only a theoretical method with many limitations and conditions, but it can be a good basis on which to start our investigation, plus the method can be developed. Later new elements, both linear and non-linear, can be added to the network. Although the focus of this paper was linear time-invariant systems.

The electrical circuit can be transformed into a directed graph, where the direction of the edges is the same as the direction of the current in calculations. In this case our network possesses six potentials. Two potentials of the feeding points, three potentials of the current sources and the ground potential (*Fig. 4*).

The index of the nodes is identical to the index of potentials. If we apply Kirchhoff's First Law to all nodes, six equations and in ordinary cases five unknown variable potentials from 1 to 5 (the ground potential is 0

V) will results so the linear equation system can be solved. They can be arranged them into a vector **Φ**.

$$\mathbf{\Phi} = \begin{bmatrix} \Phi_0 \\ \Phi_1 \\ \vdots \\ \Phi_5 \end{bmatrix} \tag{5}$$

From the potential, voltages can be calculated from *Eq.(6)*.

$$U = \Phi_i - \Phi_{i+1} \tag{6}$$

Obviously the voltages can be arranged into a vector **U**, similarly to vector **Φ**. From the resistances, a resistance matrix **R** or conductance matrix **G** can be can created. For the node-potential method, a special matrix was used. Once again, the central concept is Kirchhoff's First Law. The first row of the matrix contains the conductances associated with the first node. The direction of the voltage (likewise the direction of the current) will determine their sign.

$$\mathbf{G} = \begin{bmatrix} -G_1 & G_2 & 0 & \dots & 0 \\ G_1 & -(G_1+G_2) & G_2 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & -G_4 \end{bmatrix} \tag{7}$$

The potential vector can be reduced by omitting Φ_0 thus yielding a reduced potential vector $\mathbf{\Phi}_r$. Of course the current matrix labeled **I** can be used.

$$\mathbf{I} = \begin{bmatrix} I_I \\ I_1 \\ \vdots \\ I_{II} \end{bmatrix} \tag{8}$$

Ohm's Law helps to calculate the unknown values of the network based on *Eq.(9)*.

$$\mathbf{I} = \mathbf{G} \times \mathbf{\Phi}_r \tag{9}$$

In the method of node-potentials, the potentials are considered to be unknown, while the other parameters are given. As a result, matrix $\mathbf{\Phi}_r$ contains all parameters that should be calculated. In the present situation the two potentials of the feeding points are considered to be known, 230 V and the ground potential is 0 V. Each current value of the loads is known, but the currents of the feeding points are unknown. Therefore, the matrices should be modified so that all unknown variables will be present in one matrix, **X**, and all known parameters in another one, **C**, a constant matrix. The **G** matrix must be used as well and denoted by $\mathbf{G}_m$.

$$\boldsymbol{X} = \begin{bmatrix} \Phi_2 \\ \Phi_3 \\ \Phi_4 \\ I_I \\ I_{II} \end{bmatrix} \tag{10}$$

$$\boldsymbol{C} = \begin{bmatrix} G_1\Phi_1 \\ -G_1\Phi_1 + I_1 \\ I_2 \\ -G_4\Phi_5 + I_3 \\ G_4\Phi_5 \end{bmatrix} \tag{11}$$

$$\boldsymbol{G_m} = \begin{bmatrix} G_1 & 0 & 0 & 1 & 0 \\ -(G_1+G_2) & G_2 & 0 & 0 & 0 \\ G_2 & -(G_2+G_3) & G_3 & 0 & 0 \\ 0 & G_3 & -(G_3+G_4) & 0 & 0 \\ 0 & 0 & G_4 & 0 & 1 \end{bmatrix} \tag{12}$$

By arranging the three matrices into one equation, we get

$$\mathbf{C} = \mathbf{G}_m \times \mathbf{X} \tag{13}$$

Eq.(13) was solved using Gauss-Jordan elimination method and the same results presented earlier for $I_1 + I_2 + I_3$, I_I and I_{II}, $\Sigma\, l$, U_1, U_2, and U_3 were obtained. To make use of the Gauss-Jordan elimination method, the modified conductance matrix with the constant matrix had to be extended. With the help of a newly created matrix $\mathbf{G}_e$, the elimination process yielded the values for vector **X** directly.

The extended conductance matrix ($\mathbf{G}_e$) is as follows:

$$\begin{bmatrix} G_1 & 0 & 0 & 1 & 0 & G_1\Phi_1 \\ -(G_1+G_2) & G_2 & 0 & 0 & 0 & -G_1\Phi_1 + I_1 \\ G_2 & -(G_2+G_3) & G_3 & 0 & 0 & I_2 \\ 0 & G_3 & -(G_3+G_4) & 0 & 0 & -G_4\Phi_5 + I_3 \\ 0 & 0 & G_4 & 0 & 1 & G_4\Phi_5 \end{bmatrix} \tag{14}$$

2.3. Comparison of the Two Methods

Although both calculations yield exactly the same results, in our opinion the second one is preferred. The structure of the matrices indicates that it can be applied to even more complex grids, e.g. in radial topology. Another advantage of the node-potential-based method is that the direction of the load currents is irrelevant. For example, even a small solar plant on a rooftop can be simulated.

3. Simulation with OMNeT++

The OMNeT++ 4.x Integrated Development Environment is based on the Eclipse platform, which has been extended with new editors, views, wizards, and additional functionality. Although OMNeT++ is not a network simulator in itself, it has gained widespread recognition as a network simulation platform in the scientific community as well as in industrial settings, and has built up a large community of users.

The most common area of application of OMNeT++ is the simulation of telecommunication networks. The simulator itself is message-based, so our electrical distribution system had to "communicate" *via*

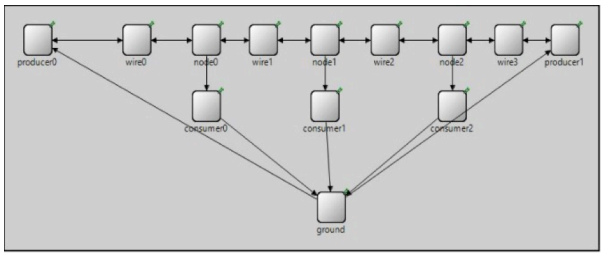

Figure 5. The schematic of the network using OMNET++.

Table 2. Summary of modules used in OMNET++ simulations.

Parts of the system	Module names
Feeding point	Producer
Load	Consumer
Connections	Wire
Node	Node
Ground point	Ground

messages, which is rather unusual in terms of physics or electrical engineering. OMNeT++ provides component architecture for models. Components (modules) are programmed in C++, then assembled into larger components and models using a high-level language (NED).

OMNeT++ simulation comprises three different files, which are (i) Source code (CPP), (ii) Network file (NED), and (iii) Configuration file (INI). In OMNeT++ modules are defined, which can communicate *via* messages. Both modules and messages are special C++ objects. In the source code, we can specialize our modules, and define their tasks precisely. The network file contains the actual topology of the network system, and the position and names of the used modules as well as their connections. With the aid of the configuration file, other network settings can be specified, though none were used in our simulations.

All modules have four functions, which can be defined by users. In our simulation, only two of them were used; the *initialize()* and *handleMessage()* functions. In the initialize function the initial parameters of the modules can be set, in the *handleMessage()* function the action in case of an incoming message can be declared. With these functions, completely different modules can be developed.

The main idea was to show the current or energy flow *via* messages in our system. It worked quite well in the case of a system fed at one end. Although because of the message and handling of messages inclinations of OMNeT++ in systems fed at both ends, the energy flow was not so easy to show.

3.1. Implementing the Distribution System in OMNeT++

The distribution system shown in *Fig.1* had to be implemented. It is obvious that we have at least three different types of modules exist in this network as follows: (i) feeding point, (ii) load, and (iii) connection between them. The implemented electrical distribution network is shown in *Fig.5*.

By analysing the topology more closely, we could identify another module could be identified, connecting two wires and a load, the node. Nodes are important parts not only of grids with their topology, but also of radial grids.

The previously used module is not shown in *Fig.1* but there should be a ground point to make our simulation easier and clearer. This ground point module is the most important as it carries out the main calculations and connects the loads to the producers. *Table 2* shows the modules and the names used in our source code and files.

3.2. Modules and Messages

The class of messages possesses several variables, including a void-type pointer called *contextPtr,* which is a user-defined pointer. In our simulation, this pointer was used to send data to modules. It points to a class, which contains the used variables such as voltage, current, effective power, reactive power, resistance, etc. Of course, some of the variables are used in only one module (e.g. the resistance in wire module), and some in all of the modules (e.g. voltage, current). In the constructor, all variables were set to zero.

The simulation consists of cycles. Each cycle begins with the producers sending messages, and ends when they receive their messages from ground modules. In simulations, the Event Logging (EV) function can be used to log the parameters or result(s) of the calculations. There are two feeding points in all simulations, though loads are user-defined (N) in pre-processor instruction where both two- and three-point-loaded networks can also be considered.

The first module is called *Producer*. It exhibits a constant voltage value set in pre-processor instruction #define UT 230. In advanced simulations, it should be a user-defined value, and it is not necessarily constant. It could be a function as well, but in this simple simulation, it will remain constant throughout the whole process. At the start of the simulation, only the voltage of the producer is known, but the current and power are unknown parameters until the end of the turn. Thus, these values are set to -1 at the beginning. All three parameters are set in the *initialize()* function. The next stage of the module is handling an incoming message. The incoming message is actually a pointer to the message. At first, in all modules, the values of the pointer were stored as local variables were identified. This is very useful as during simulation some memory allocation problems. The message contains information only about the currents (the voltage remains constant). The current of the producers is set to the value from the message. The power is calculated from the current and voltage values. The module is connected to the ground and one-wire modules. The producer module logs the voltage, current, and power.

The second module simulates the load and referred to as *consumer*. It exhibits a constant current and power factor. Both users are defined. Current values are the same as discussed above in *Section 2*. From these and the voltage from the incoming message, the module calculates the power and reactive power. This block sends the current value to the *ground* block for further calculations. Consumer modules are connected to one node and the ground module, which logs voltage, current, power, reactive power, and the power factor.

In real networks, the connecting wires exhibit resistances as well, causing a voltage drop in the system. In our example, this is desirable, almost 1 Ohm, but our calculations are simple. In the *initialize()* function, the lengths from *Table 1* and R_1 to R_4 resistances from *Section 2* are set. The wire module calculates the voltage drop simply with Ohm's law. Obviously, it shows a useful value from the second cycle, as the current is negative until that turn. Wires are connected to one *producer* and one *node* or between two nodes. Wires log the voltage drop, current, resistance and length.

The Node module is especially useful in radial topology, but also implemented here as well. The aim of this module is to distribute current by applying Kirchhoff's First Law to our network. It works only after the first 'initial stage'. The module is connected to two wires and one consumer module. It shows the value of currents.

The ground module conducts the main calculations of the simulation. It determines the exact current values for each producer module. The calculations are based upon *Eqs.(2)* and *(3)*. The module takes into consideration the length of each wire and the currents from the consumer modules and distributes them. The ground module is connected to all consumer and producer modules in addition to logging the sum of currents and the currents for each producer.

3.3. Simulation

As stated before, this OMNeT++ simulation can be divided into three different stages. At first, the OMNeT++ simulation engines build the network with the user-defined values for each module. This is referred to as stage 0, and there is no logging occurs here. In other simulations, logging is possible here as well, but in this case this opportunity is omitted.

The next stage, or the first cycle, starts when the producers send their first message, and ends when they get their message from the ground module. This part is not necessary in other kinds of simulations. In this part all modules have their own user-defined values, but they do not have any effect on the other modules, e.g. producers exhibit their own voltages, but their current and power are both -1 or the voltage and current of wires are both unknown (-1), as shown in *Figs.6* and *7*.

At this stage, the voltage of the consumers is 230 V, and its powers are calculated with this voltage, shown in *Fig.8*. The first cycle of the simulation is examined in *Fig.9*. Currents are "delivered" between the

```
** Event #1  t=1  enetwork.wire0 (Wire, id=5), on `{}' (cMessage, id=0)
Wire U = -1
Wire I = -1
Wire R = 1
Wire l = 27.027
```

Figure 6. Wire logging during the first cycle.

```
** Event #3  t=2  enetwork.node0 (Node, id=7), on `{}' (cMessage, id=0)
I in = -1 I out = -1
```

Figure 7. Node logging during the first cycle.

```
** Event #10  t=3  enetwork.consumer2 (Consumer, id=13), on `{}' (cMessage, id=13
Consumer voltage: 230
Consumer current: 5
Consumer power: 1115.5
Consumer reactive power: 279.571
Consumer power factor: 0.97
```

Figure 8. Consumer logging during the first cycle.

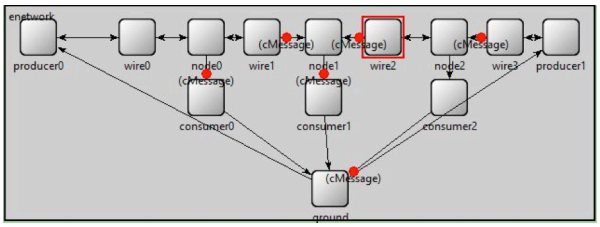

Figure 9. The first cycle of the OMNeT++ simulation.

```
** Event #22  t=23  enetwork.ground (Ground, id=11), on `{}' (cMessage, id=13
I GND = 31
I1 = 15.4002
I2 = 15.5998
```

Figure 10. Current distribution.

```
** Event #23  t=24  enetwork.producer0 (Producer, id=2), on `{}' (cMessage, id=13)
Producer voltage: 230
Producer current: 15.4002
Producer power: 3542.04
```

Figure 11. Voltage, current and power of the first feeding point.

```
** Event #25  t=25  enetwork.wire0 (Wire, id=5), on `{}' (cMessage, id=13)
Wire U = 15.4002
Wire I = 15.4002
Wire R = 1
Wire l = 27.027
** Event #27  t=26  enetwork.node0 (Node, id=7), on `{}' (cMessage, id=13)
I in = 15.4002 I out = 5.40016
** Event #34  t=27  enetwork.consumer2 (Consumer, id=13), on `{}' (cMessage, id=48)
Consumer voltage: 214.4
Consumer current: 5
Consumer power: 1039.84
Consumer reactive power: 260.609
Consumer power factor: 0.97
```

Figure 12. Logging of wire, node, and consumer modules in the second cycle.

modules throughout the whole network system. However, at the end of the initializing stage, the ground module calculates the sum of currents of the consumers and the currents of each feeding point *(Fig.10)*. Thus, the producers receive their currents, and the correct calculated values, e.g. voltages, powers, etc. will be obtained without any unknown parameters *(Fig.11)*.

```
** Event #27  t=26  enetwork.node0 (Node, id=7), on `{}' (cMessage, id=13)
I in = 15.4002 I out = 5.40016

** Event #30  t=26  enetwork.node2 (Node, id=14), on `{}' (cMessage, id=42)
I in = 15.5998 I out = 10.5998
```

Figure 13. Applying Kirchhoff's First Law to the Node modules.

The previously shown modules can be examined and it can be concluded that our voltage and current values are known *(Fig.12)*. It could be interesting to see how our nodes work after the first stage, when applying Kirchhoff's First Law *(Fig.13)*. We have to mention that in the case of node modules I_{out} means the current of the consumer subtracted from I_{in}.

3.4. Validation of the Simulation Results

It has been seen that the results of the OMNeT++ simulation and the results of the classical method of calculating are the same. As an example, the results of the voltage drops are shown in *Fig.14*.

3.5. Topology

All electric systems have unique topologies the position of the components or element(s) and the wires that connect them. By considering a network fed at both ends with only one node, it conatins one node module and two connecting modules between the feeding points and the node. In the case of a similar network with two loads there are two nodes and three connecting channels, as in the previous case. From this point of view the simulated network can be checked, this one contaists of three loads and four channels. If a gred fed at both ends with N loads (where N is a positive integer) exists then the number of wire modules, w (also a positive integer) can be calculated as follows:

$$w = N + 1 \tag{15}$$

Eq.(15) is only true for this type of topology. For example for a topology in which there is only one feeding point, the number of loads obviously is equal to the number of wires (and nodes). A proven formula can be applied to more complex topology variations. The advantage of these formulae are that they automate the creation of NED files either from another piece of software or implemented from OMNeT++.

4. Summary of Simulation Experiences

It can be concluded that our DES program works properly as shown in *Section 3.4*. Obviously there are some advantages and disadvantages of this method.

4.1. Advantages

First of all it is an open-source platform, thus it can be modified and developed easily. The use of other auxiliary pieces of software can be added, too. As the program includes a graphical viewer, individual

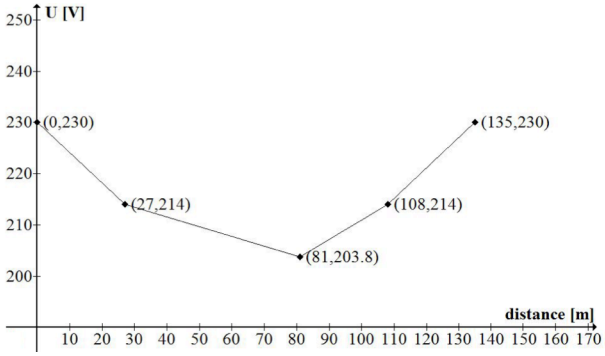

Figure 14. Voltage drop of the loads.

simulation results can be followed. The message direction is the same as the current direction, thus a negative current means failure in the program, or negative values can be defined as unknown parameters. Our system can be observed for values of interest. With this definition the model later may be used in transient analysis to consider the reinitialization cycles each time.

4.2. Disadvantages

On the other hand, as an open-source platform it is still in the development phase. Sometimes the program gave us crash reports during the development phase. The message direction is the same as the current direction; however, according to the message sending system of OMNeT++ extra message(s) can be obtained.

Another issue is the network description file (NED). Each time our network system is modified, the whole code needs to be modified, including the connection definition part, as well as the gate declaration. Topology statements are omitted. Although the time of the simulation is user-defined, all the steps are to be followed the simulation may be rather lengthy. In this case, the simulation of another type of electrical distribution system was attempted, e.g. for a system fed at only one end, a totally new program has to be written. This program can be solved only if a formula is created for each different type of topology. Real electrical grids may be more complex than the situations our DES can handle at the present.

5. Conclusion

The models of the electrical components have been developed in this paper based on engineering principles that are able to describe the behaviour of an electrical grid. It seems that the OMNeT++ discrete event simulator is suitable for the simulation of electrical grids. At first we need to fix some problems mentioned in *Section 4.2*.

A future task would be to implement new electrical network components, e.g. photovoltaic power plants, wind turbines, and different loads, etc. to the simulation environment and subsequently the OMNeT++ would be suitable to simulate smart grid networks.

SYMBOLS

DES	Discrete Event Simulatoion
EV	Event Logger
I_1, I_2, I_3	currents of the loads
$I_\mathrm{I}, I_\mathrm{II}$	currents of the feeding points
l_1, l_2, l_3, l_4	length of the wires connecting loads and feeding points
N	number of loads
w	number of wires
q	diameter of the wire
U_1, U_2, U_3	voltages of the loads
U_T	voltage of the feeding point
ϱ	resistivity
NED	Network Description File
INI	Initialization file
Φ	potential
$\mathbf{\Phi}$	potential vector
$\mathbf{\Phi_r}$	reduced potential vector
$\mathbf{R}$	resistance matrix
$\mathbf{G}$	conductance matrix
$\mathbf{G_m}$	modified conductance matrix
$\mathbf{G_e}$	extended conductance matrix
$\boldsymbol{I}$	current vector
$\mathbf{X}$	vector of variables
$\mathbf{C}$	vector of constants

Acknowledgement

We acknowledge the financial support of this work by the Hungarian State under the VKSZ_12-1-2013-0088 project.

REFERENCES

[1] OMNeT++ Discrete Event Simulator omnetpp.org

[2] Mets, K.; Verschueren, T.; Develder, C.; Vandoorn, T.L.; Vandevelde, L.: Integrated simulation of power and communication networks for smart grid applications, *Proc. IEEE 16th Int. Workshop, Computer Aided Modeling and Design of Communication Links and Networks (CAMAD),* pp. 61-65, 2011

[3] Niemi, R.; Lund, P.D.: Decentralized electricity system sizing and placement in distribution networks, *Applied Energy* 2010 **87**(6), 1865-1869 DOI: 10.1016/j.apenergy.2009.11.002

[4] Gonen, T.: Electrical power transmission system engineering: analysis and design (CRC Press, Boca Raton, FL USA) 2011

[5] Jamniczky, Á.: Electric engines, (University of Veszprém Press, Veszprém, Hungary) pp. 31–49, 1994 (in Hungarian)