Power Distribution Network Reliability Modeling Based on the Statistical Data and Multi-agent Simulation

Miroslav Prýmek[1)], Aleš Horák[1)], Tadeusz Sikora[2)]
[1)] Faculty of Informatics, Masaryk University Brno
Botanická 68a, 602 00 Brno, Czech Republic
E-mail: {xprymek,hales}@fi.muni.cz
[2)] VŠB – TU Ostrava, Department of Electric Power Engineering, 17. listopadu 15, 708 33 Ostrava, Czech Republic
tel: +420 596 999 307, E-mail: tadeusz.sikora@vsb.cz

ABSTRACT

We display the preparation and results of a real electric distribution network (EDN) simulation and its reliability analysis using the parameters calculated from the real database of EDN failures in Czech Republic from last years. An example network is simulated in the RiceVM environment.

Keywords: Electric Power Network, RiceVM, Simulation

1 INTRODUCTION

We have prepared the simulation environment of the Rice system to model a real electric distribution network described in the next section. The simulation of the network reliability was done using parameters calculated from the failure database [1]. The load nodes in the simulation are defined with dynamic power flow at the day basis, so the amount of energy-not-supply varies with the exact time of failure occurrence [2,3]. The real network is simulated in the RiceVM software environment [4].

2 THE REAL NETWORK DATA

The simulated network data come from a real scheme of middle-voltage distribution network in Ostrava (see the Figure 1). The network is supplied by 110 kV/10 kV substation MART. Among others there are two main loads in this grid - Fnem and Energ. These load nodes also interconnect particular supply branches.

Each load node is characterized by its maximum load P_{max} and the type of average daily load course. Based on practical measurements, three types of daily load course were determined: households, mixed load (both households and light industry) and light industry. Assigned types and maximum loads to nodes are in the Table 1. According to the maximum load, the typical load course was proportionally adjusted, so the peak on the load curves reaches P_{max} (see the Figure 2, all load courses were adjusted to P_{max} = 3 MW).

The modeled network is a typical urban cable system with no maintenance reliability parameters. The mean time to repair was set to 215 hours. The failure rate was calculated according to the cable length. The failure rate of the cable 4,85 1/Year/100 km was calculated from the failure database [1].

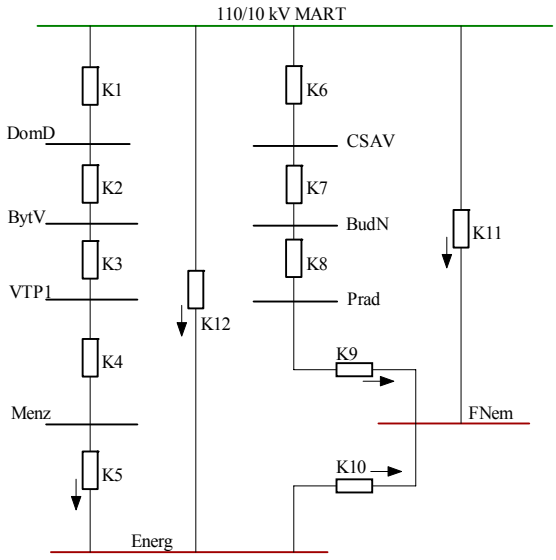

Figure 1: The scheme of the simulated network

Table 1: Load types and maximum loads of nodes

Node name	Type	P_{max} (MW)
MART	source	
Dom	households	0,2
BytV	households	0,25
VTP1	mixed	0,4
Menz	light industry	0,47
Energ	light industry	1,1
CSAV	light industry	0,15
BudN	light industry	0,22
Prad	mixed	0,26
Fnem	light industry	3

3 RICEVM - THE RICE SIMULATOR WORKING ENVIRONMENT

RiceVM is the virtual machine designed for the evaluation of the Rice power network simulation system. It is based on the simplified Arch Linux distribution [6] with X-server,

web server and fully functional Rice system. The main features of RiceVM are:

- Graphical user interface for controlling all the main system features
- Automatic updates
- Flexible agents behavior definitions
- Reports generation
- Decentralization and scalability support

The Rice simulator is designed as decentralized and modular [8] and this principle is kept even in the RiceVM distribution. It is possible to test the decentralized deployment of the system by running multiple version of the virtual machine on one or more physical computers interconnected with the network. This arrangement can be used for decentralized simulation of one power network.

To keep all of the virtual machines up to date it is essential to have an automatized system for the distribution of the actual system components. in the RiceVM automatic updates are based on the Arch Linux package manager Pacman [7]. The packages with the actual versions of the system components are stored in the package repository on the central server. Installation of the new versions of the components is fully automatized and can be initialized from the graphical interface (GUI) or remotely.

The GUI is designed to easily control all the main features of the simulator system - not only Rice components but also the software updates and basic configuration such as the name server used. You can see the GUI on the Figure 3. The left part of the figure displays the controller GUI and the right part contains the Viewer application showing the simulated network.

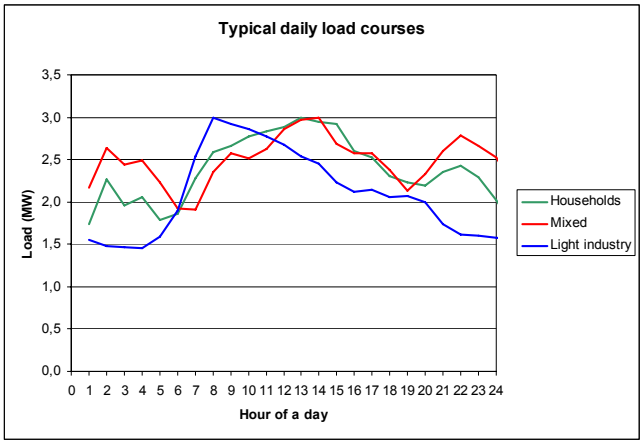

Figure 2: Typical load courses

The Rice simulator is based on the interchange of the small pieces of information between the agents which represent the components of the power distribution network. The behavior of the agents constitutes the overall behavior of the simulator. The agents behaviour itself is based on the small code pieces associated with the concrete information interchange acts. This way the overall simulation definition is strictly event-driven and highly modular.

In [4, 10] we have discussed the ways of using the XML format for the definition of the behavior of the agents. We have proposed the format which uses declarative approach combined with the definition of the event-driven actions in the Python programming language.

Using this definition format, we can describe not only the components to be simulated (declarative portion) but also the power distribution network features to be simulated and means of such a simulation (actions portion). Both is defined in the ways which can be processed and generated automatically without the need of the source code modification or extension.

In the case of the RiceVM, the simulation definition is placed in one XML file which can be loaded into the system using the web interface.

The web interface is also used for the presentation of the results of the simulation. The values of the variables to be watched are gathered in the special database according to the instructions in the simulation definition file. After the simulation run, these values can be visualised, statistically analyzed or exported for another processing. The RiceVM itself contains the module for charts-making again driven by the definition present in the simulation definition file. The generated charts are presented in the web interface of the system. The main export format is the comma-separated-values (CSV) file.

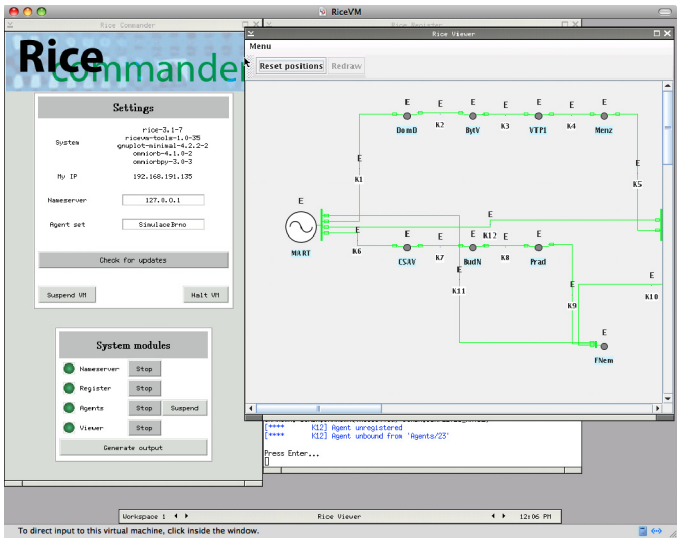

Figure 3: RiceVM

4 Example Simulation Using the Real-World Data

In this section we will describe how to use the proposed simulation-description format for the implementation of a simulation which will be based on the real-world gathered data. We will simulate the part of a real distribution network with ten nodes and twelve lines. We will focus on the power consumption and failure rates.

First, we must define the configuration options concerning the whole simulation:

```
<configuration>
 <time>
  <start value="080101"/>
  <stop value="100101" />
  <coefficient value="3600"/>
 </time>
</configuration>
```

In this code we declare the time-frame of the simulation. The Rice system defines time of the simulation as the interval between two given virtual-time moments. Virtual time is defined by the coefficient used to multiply the real time elapsed. In this case we define the simulation time-frame as the interval between January 1st 2008 and the same day in 2010. The coefficient is 3600 which means that one hour of the virtual time will elapse during one second of the real time.

Our main focus in this example is the energy consumption. We will use the consumption values identified in the real power network we are simulating. These data are presented on the hour basis for the period of one day as illustrated in the daily consumption chart of the "CSAV" node of the network, see the Figure 4.

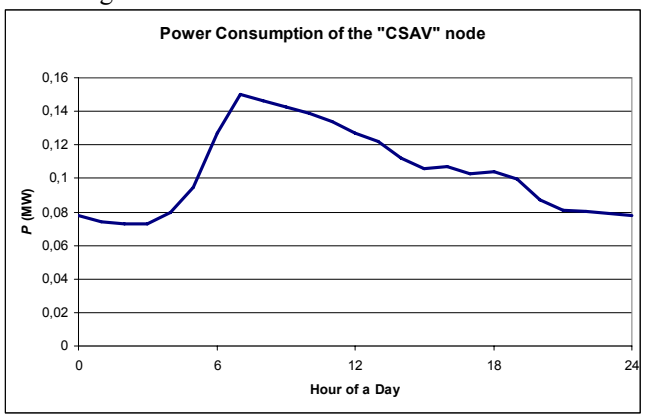

Figure 4: Consumption values example

The consumption values will be periodically loaded into the agent's knowledge slot named myConsumption and the consumption change will be summed to the overall power flow through the node (`consumption` knowledge). Implementation of the loop timer will be as follows:

```
<loopTimer triggerTime="0.3">
<action type="python">
def action(data):
 agent=data['agent']
 hourNow=Utils.timeNowConverted()['hour']
 myConsumption=
agent['dailyConsumption'][hourNow]
 myConsumptionChange   =   myConsumption-
agent.get('myConsumption',0)
 agent['myConsumption'] = myConsumption

 agent['consumption']                    =
agent['consumption']+myConsumptionChange
 return 1
</action>
</loopTimer>
```

This code will be common for all node agents. We will make a template named consumption-simulation and will include it into all agents definitions. The consumption values table is different for each agent and thus must be inserted straight into the agent definition:

```
<init>
 <knowledge name="dailyConsumption"
   value="py:[0.571,  0.545,  0.542,  ...,
0.591, 0.591, 0.582 ]"/>
</init>
```

These two pieces of code together ensures that every agent has the correct value of the consumption according to the hour of the virtual time present in the knowledge `myConsumption`. But we must deal with the interchange of this value between agents. We will use the subscription mechanism. Every agent will subscribe for the value of the consumption of his outputs as soon as the outputs are connected to it. I.e. it is a reaction to the change of the output knowledge. We will write this handler for it:

```
<handler valueName="output" name="output-
handler">
<action type="python">
def action(data):
 agent= data['agent']
 newOutputId = data['newValue']

 # append the new output to the list of
outputs
 if agent.get('outputsList',None)==None:
  agent['outputsList']=[newOutputId]
 else:
 agent['outputsList'].append(newOutputId)

 # subscribe for the output's consumption
  agent.sendMessage(agent.makeKqmlMessag
e('subscribe',outputAid, '',

agent.makeTrivialMessage({'receiver':outp
utAid,'performative':'ask','con-
tent':'consumption=?',
      'reply-with':'consumption  of  %s'%
newOutputId}),  ''))

 return True
</action>
</handler>
```

The subscription means that as soon as the value of the consumption knowledge changes in the output, it will send the message to our agent. The message will be labeled with the in-reply-to tag "consumption of X" where X is the ID of the output agent.

The last piece needed is the procedure which will sum the consumption values present in the incoming messages (from outputs) and the agent's own consumption (myConsumption) and place the sum into the knowledge consumption (which can be propagated further in the same manner). This procedure must be triggered by the incoming messages. Its code looks like this:

```
<incommingMessages>
<handler inReplyToFilter="consumption of"
bypassKb="yes"
 name="consumption-summation">
<action type="python">
def action(staticData,**fireData):
  agent = staticData['agent']
  consumption          =          fire-
Data['messageSemantics']['consumption']
  message = fireData['message']
  sender = message.sender
  # store the cons. value and compute the
change
```

```
  oldConsumption                    =
agent['outputConsumptions'].get(sender,0)
  agent['outputConsumptions'][sender]    =
consumption
  consumptionChange = consumption - old-
Consumption
  # change the overall consumption of the
node
  agent['consumption']                    =
agent['consumption']+consumptionChange
  return True
</action>
</handler>
</incommingMessages>
```

We have developed the whole chain of actions which are needed to transport the consumption values from the energy consumer to the source. These simple one-purpose actions cooperate in such a way that every agent representing a node in the network has the actual value of the energy going through it in the knowledge consumption. Values are computed automatically with every change of the consumption in any of the nodes.

Of course the above-described set of actions also ensures that the values of the energy consumption are summed according to the actual network topology in spite of we have no central data structure describing the topology. All the actions are performing their tasks on a strictly local basis and according to agent's knowledge about the surrounding world.

5 Values Logging and Charts Generation

The last part of the simulation implementation is the definition of the desired result. Let's suppose we want to see the chart of the overall network power consumption and we want its values be exported for the further processing. We have the network with one source point - the 'MART' node. We also know that every node has the value of its overall power consumption stored in the consumption knowledge. The task is to dump every change of this knowledge, make the chart and export the timeline into the CSV file.

As have been said, the Rice power network simulator has primitive functions to store values (as the timelines) into the database. The first part of the task is to develop a change handler of the knowledge consumption. A change handler is a function which is triggered every time the knowledge value changes. In our case we must save the new value of the knowledge into the database for further processing. The code will look like this:

```
<handler          valueName="consumption"
name="uloz-consumption">
    <action type="python">
def action(data):
 agent=data['agent']
 agent.logValue('consumption')
 return True
    </action>
</handler>
```

The primitive function `logValue` correctly saves the desired value along with the virtual time value.

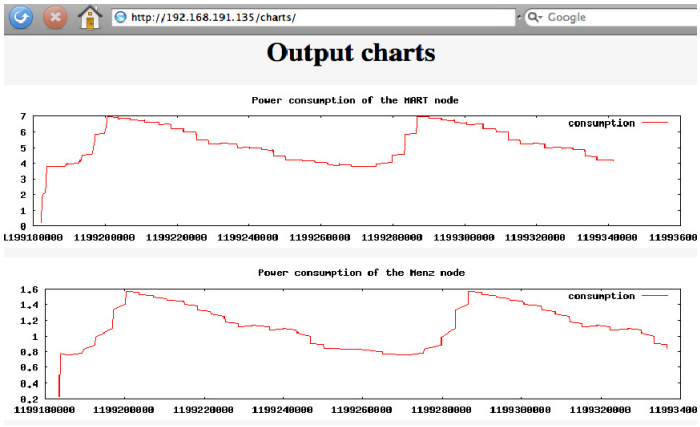

Figure 5: Charts page of the web interface

The next step is the chart generation. The RiceVM web interface has built-in support for this. We must declare which stored knowledge values should be used in the chart:

```
<output>
   <chart db="/tmp/evdbs/MART"
      title="Power consumption of the
MART node">
         <knowledge name="consumption"/>
   </chart>
</output>
```

The CSV file creation is similar:

```
<output>
  <csv db="/tmp/evdbs/MART">
    <knowledge name="consumption"/>
    <knowledge name="rlStatus"/>
  </csv>
</output>
```

By this code we tell the system that it should create CSV file containing the values of: virtual time (implicit), power consumption and the state of the node (its potential failure codes).

After the simulation run, we click on the "Generate output" button of the RiceVM GUI and the web interface will contain the chart and link to the generated CSV results export. The output page with two charts is presented in the Figure 5.

6 Conclusion

The simulation of the simple power distribution system has proven RiceVM to be useful for calculation of energy not supply and it seems also promising for calculation of the energy supply reliability. Nevertheless for the full applicability, the RiceVM must be further tested with other power network configurations to meet all the required criteria for modelling of the distribution network reliability.

ACKNOWLEDGEMENTS

This work is supported by the Czech Science Foundation - project No: GAČR – 02/09/1842.

REFERENCES

[1] Krátký, M., Rusek, S., Goňo, R., Dvorský, J.: A Database Framework to an Analysis of Data of Failures in

Electrical Power Networks. In Proceedings of ELNET 2005, Ostrava:VŠB-Technical University of Ostrava, 2006, pp. 66-77, ISBN 80-248-0975-3.

[2] Prokop, L., Hradílek, Z.: Results of reliability analysis and Energy not supply Estimation. In Proceedings of ELNET 2006, Ostrava : VŠB-Technical University of Ostrava, 2006, pp. 9-14, ISBN 80-248-1216-9.

[3] Prokop, L., Hradílek, Z.: Theoretical Bases of Energy not Supply Estimation. In Digital Technology Journal, 2007.

[4] Prýmek, M., Horák, A.: XML as an agent behaviour definition language. In the Proceedings of ElNet 2007, VSB Technical University of Ostrava, Ostrava, 2007.

[5] Prýmek, M., Horák, A.: Multi-Agent Framework for Power Systems Simulation and Monitoring. In the Proceedings of ICICT 2005, Cairo, Egypt : Information Technology Institute, Giza, pp. 525-538, 2005.

[6] Judd Vinet and Aaron Griffin: Arch Linux distribution, www.archlinux.org, 2008.

[7] Using Pacman, http://archux.com/page/using-pacman, 2008.

[8] Prýmek, M., Horák, A.: State-oriented Maintenance of Electrical Power Systems with Dynamic Multi-agent Network. In Proceedings of CSIT 2006, pp. 310-315. Amman, Jordan : Applied Science Private University, Amman, 2006.

[9] Russel, S. and Norvig, P.: Artificial Intelligence, A Modern Approach, Pearson Education, Inc. : Upper Saddle River, New Jersey, 2nd edition, 2003.

[10] Prýmek, M., Horák, A.: Optimizing KQML for Usage in Power Distribution Network Simulator. In Proceedings of 3rd International Symposium on Communications, Control, and Signal Processing (ISCCSP 2008), pp. 846-849. St. Julians, Malta : IEEE, 2008.

[11] Shafranovich, Y.: RFC 4180: Common Format and MIME Type for Comma-Separated Values (CSV) Files. The Internet Engineering Task Force. http://www.ietf.org/rfc/rfc4180.txt, 2008.Drapela, J., Toman, P. Interharmonic – Flicker Curves of Lamps and Compatibility Lever for Interharmonic Voltages. 2007 IEEE Laussane Powertech, Switzerland. IEEE PES, 2007, 6pp.