OPENDSS MODEL OF ORIVESI MV-GRID, DOCUMENTATION

Table of Contents

Introduction

Model was created for yearly power flow calculation so that user would be able to define time series for transformer loads and then simulate grid voltages for predetermined time period. However the model can be modified for other types of analyses by user relatively easy. For example by specifying harmonic spectrums for transformer loads user could run harmonic sweep analysis or by attaching an EnergyMeter object into the model user could run a simulation for yearly grid energy losses.

Model is, in essence, a collection of dss-script files each representing an individual part of the model. The composition and general relations of the script files are depicted in figure 1.

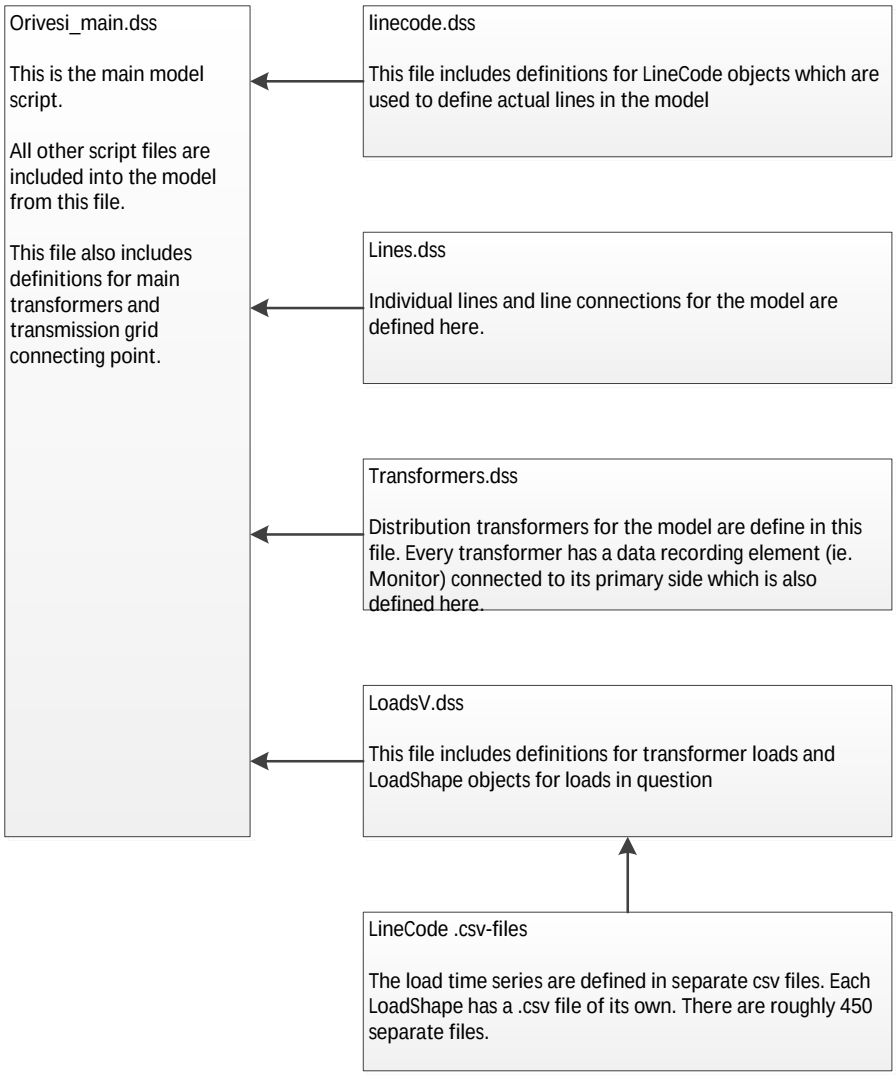

Figure 1: Composition of the model

After reading this documentation reader should be able to run yearly power flow simulations from MATLAB using this model.

Original network data

Network model is composed from data supplied by Elenia Oy, which was modified to format OpenDSS understands. Data was delivered in excel tables each representing the mv-grid behind an individual substation feeder. In this chapter the method for interpretation of network data is described.

Conductor type data

Data regarding conductor types was in an individual excel table. In this model this data is used to define LineCode objects. These objects are used in OpenDSS to define per length unit characteristics of any line configuration. For every conductor type a separate LineCode object is defined. LineCodes are written to file "linecode.dss" which is included to the model from the master script file. Separate LineCodes are called when the lines for the model are defined.

In the following example the method for creating a LineCode object is demonstrated.

Example:

Overhead line with aluminium conductors has following characteristics:

Tunnus	Johtolaji	R	X	R0	X0	RN	XN	BN	BL	IMAX	IKMAX
A54	A54 3x54 Al	0.537	0.391	0	0	0	0	1.916	2.93	280	4.7

"Tunnus" is used as name for LineCode object. R and X are series resistance and inductive reactance per kilometre, respectively. BN and BL are shunt and line-to-line susceptances (µS/km). IMAX is maximum loading current (A) and IKMAX is maximum short circuit current (kA).

For LineCode object resistance and reactance values are used directly. Susceptance values however must be converted into per length capacitance values (nF/km). This can be done by applying following equation:

$$C\left[\frac{\text{nF}}{\text{km}}\right] = 1000\frac{B\left[\frac{\mu\text{S}}{\text{km}}\right]}{\omega} = 1000\frac{B\left[\frac{\mu\text{S}}{\text{km}}\right]}{2\pi 50\ \text{Hz}}$$

In linecode.dss file this conductor type would be defined in the following way:

```
new LineCode.A54 units=km R1=0.537000 X1=0.391000 R0=0.537000
~ X0=0.391000
~ Cmatrix="6.098817 | 9.326480 6.098817 | 9.326480 9.326480
~ 6.098817"
~ Emergamps=280
```

Feeder data

The actual network data for line connections and distribution transformers is divided for each system feeder. The data consists of definitions for transformer stations and individual transformers, lines and transformer station busbars, MV-isolators and feeder root connection point (i.e. Main transformer bus). Network connections are specified by using xy-coordinates. For lines coordinates for both beginning and the ending points of the line are specified.

Lines and busbars

Line and busbar data is used to define both bus connections and Line objects in the model. Bus connections are discussed further, here we focus on line data.

As discussed is previous chapter line objects are defined by using conductor data and line length, these are define in line data table in fields “Tunnus” and “Pituus” respectively. See appendix A for examples of line and busbar data.

Distribution transformers

Distribution transformer data is defined in two tables: distribution transformer table which has VA and impedance values for individual transformers and transformer station table which has a name and id for the transformer.

In the distribution transformer table values of interest for transformer are nominal VA, load losses, no-load losses, nominal primary and secondary voltages, vector group and short-circuit impedance. These values are found in fields “Mitoitusteho”, “Kuormitushäviöt”, “Tyhjäkäyntihäviöt”, “Ensiöpuloen mitoitusjännite”, “Toisiopuolen mitoitusjännite”, “Kytkentäryhmä” and “Oikosulkuimpedanssi” respectively. See Appendix B for an example in transformer data.

Main transformers

There are two main substation transformers which are defined separately in original network data. For substation transformers a bit more data is provided than for regular distribution transformer, namely minimum and maximum series resistance and short-circuit impedance values. For this model average impedance values are used. OpenDSS doesn’t differentiate substation transformers from distribution transformers. Therefore all transformers, including substation and distribution transformers, are modelled as Transformer objects.

Main transformer table also includes values for network impedances behind substation at transmission network side. These values are used to define Thévenin equivalent source feeding the model.

Composing the model

Composing the model from the network data introduced above is a fairly straightforward process. The lines in the network are modelled with Line objects and transformers with Transformer objects, both of which are

built in representations of lines and transformers in OpenDSS. Loads are connected directly to distribution transformers' secondaries and they are modelled with Load objects.

For purposes of clarity not every object is defined in a single script file. As show in chapter Introduction the complete model is composed of multiple .dss-files each containing definitions for different types of objects.

The most troublesome part of the modelling was the representation on connections in the original network data versus connection representation in OpenDSS. These are discussed in chapters to come.

Bus list

The need for a bus list arose from the way of connection representation in original network data vs. network connections in OpenDSS. As said before original network data specified connections using xy-coordinate system. This meant that if an arbitrary line A begun from coordinates (x1,y1) and ended in coordinates (x2,y2), and another arbitrary line B begun from coordinates (x2,y2) and ended in coordinates (x3,y3) these lines were considered to be connected so that the beginning on line B was connected to the end of line A. However in OpenDSS the connection points are specified by using bus names instead of coordinates.

This problem was solved by generating a bus list which bound an individual coordinate to an integer. This integer was then used in bus name, for example "Bus_34". Now, a certain coordinate connection could be expressed as a simple string. The syntax of the bus list is following:

$X\text{-coordinate}_1$	$Y\text{-coordinate}_1$	$Integer_1$
…		
$X\text{-coordinate}_n$	$Y\text{-coordinate}_n$	$Integer_m$

Note that $n \geq m$ because multiple coordinates may refer to same connection point.

The generation of bus list begun from listing the busbar coordinates. If two coordinates were connected together via busbar they were considered to be same connection point (ie. bound to same integer). Essentially this meant that connections made with busbars were reduced from the model and objects connected with busbars were just simply connected directly together. This is illustrated in figure x.

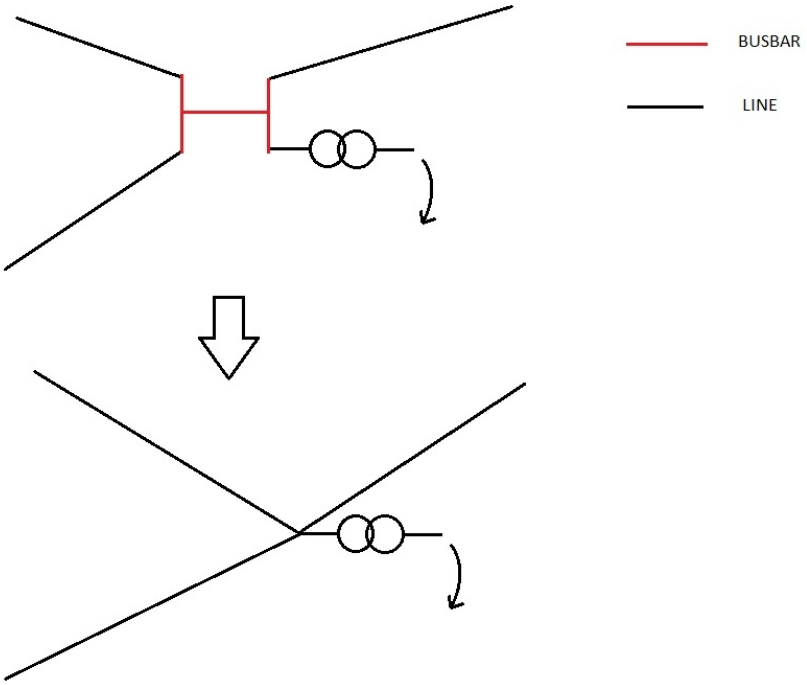

Figure 2: Busbar reduction

After busbar coordinates were added to the list, the line coordinates were added. If a line was connected to a busbar the coordinate already existed in the list and there was no need to add it again. Also if line didn't connect to a busbar, but connected to some other coordinate already in the list, the coordinate was not added. On the other hand if the line didn't connect to a busbar and did not connect to any existing coordinate the coordinate was added with a new integer reference.

Finally, when the list was completed we could use the coordinate-integer relations in defining the actual network objects.

Bus naming conventions

The before mentioned bus list is created separately for every feeder. If just integer values were to be used in bus naming, it could (and most probably would) result in unwanted interconnections between feeders. This is avoided in the model by using bus name which consists of the feeder name and the reference integer. For example, let's consider naming the bus for coordinate in "Teollisuus" feeder which reference integer is 10. The bus name in the model would be "kj_bus_teollisuus10". In general the bus name can be divided into 3 parts. First, the prefix "kj_bus_" is used when the bus in question is a MV bus. "teollisuus" refers to the feeder the bus is associated and last "10" is the reference integer obtained from the bus list. So in general the MV-side bus name syntax is "kj_bus_'*feeder name*''*reference integer*'".

The bus naming system introduced above is used when bus in question is associated with MV-side of the model. Generally most of the buses in the model are named using this system. Currently only exceptions to this convention are distribution transformer secondary buses. With these buses following bus name syntax is used: "scd_'*transformer station id*'". For example the secondary bus of the example transformer in appendix B is named "scd_101570".

Lines

The power lines are represented with Line objects in OpenDSS model. These objects are defined simply by stating line length, conductor type and bus connections. For example the line in appendix A is defined in OpenDSS script followingly:

```
new Line.1290115 units=m Linecode=Pg99 bus1=kj_bus_teollisuus214
~ bus2=kj_bus_teollisuus2 length=546.126787
```

Few notes from this definition are in order. Firstly, the bus names are obtained by searching the bus list for reference integer for both end point and beginning point coordinates. Then the bus name is constructed as instructed in previous chapters. Secondly, LineCode units for per length values were defined as per kilometre and here we use meter as line length. This is no problem as OpenDSS converts the units internally as long as they are explicitly defined for both objects.

For the model a file lines.dss is created. In this file every line in the model is defined.

The modelled network is partly built meshed for purposes of secondary feeding if some part of network would fail. Normally network would operate radially. This causes a problem in the model because under normal connection conditions there would be open isolators in meshed parts to prevent meshed operation. This however is not the case with the model which assumes that all defined lines would be connected from both ends. In order to make model correct the open isolators must be somehow added to the model.

One possibility to model open isolators in OpenDSS is to define extremely low impedance lines to the places in model where an isolator should be and then define the line in question disabled. These lines would then effectively serve as switches in the model. There is however a problem in this solution. OpenDSS converts impedance models to admittances to calculate power flow. For near-zero impedance this would mean near-infinite admittance which could cause computational errors. Therefore this is not implemented.

As the number of normally open isolators is low, it was decided that the most practical way in dealing with this problem was to open the isolators manually in the lines.dss file. This was done by declaring those line terminals open where an open isolator should be.

In test simulations it became apparent that some parts of the network were left unconnected to the rest of the model. This was due to errors in original network data (most likely missing busbars) and it was corrected by manually entering correct bus definitions to lines.dss file.

Transformers

All transformers including substation and distribution transformers are represented with Transformer objects in OpenDSS. The Transformer object definition is bit more complex than Line object's. Generally one would specify transformer by entering phase count, number of windings, vector group, bus connections, nominal winding VA's and primary and secondary voltages. Also, to correctly model transformer voltage loss and real power losses, one must enter short-circuit reactance between primary and secondary windings and per cent values for load and no-load losses.

For example the transformer described in appendix B would be defined in transformer.dss file followingly:

```
new Transformer.Siltaoja Phases=3 Windings=2 kVs=[20.000000
~ 0.400000 ] XHL=3.998321 kVAs=[315.000000 315.000000]
~ %Loadloss=1.158730 %Noloadloss=0.193651 conns=[Delta Wye]
~ buses=[kj_bus_teollisuus2 scd_101570]
```

Values for kVs and kVAs arrays are taken directly from original network data. XHL, %Loadloss and %Noloadloss must be calculated. XHL, the per unit short-circuit reactance from high voltage side winding to low voltage side winding can be calculated from short-circuit impedance, z_k:

$$z_k = r_k + jx_k$$

$$x_k = \sqrt{z_k^2 - r_k^2}$$

$$x_k = \sqrt{z_k^2 - \left(\frac{P_{Loadloss}}{S_n}\right)^2}$$

And %Loadloss and %Noloadloss followingly:

$$\%\text{Loadloss} = \frac{P_{Loadloss}}{S_n} * 100$$

$$\%\text{Noloadloss} = \frac{P_{Noloadloss}}{S_n} * 100$$

Transformer vector groups are modelled in OpenDSS by specifying transformer winding connections accordingly. For example the transformer above connection Delta-Wye would result in vector group Dyn11 as given in original network data. By specifying bus connections "buses=[kj_bus_teollisuus2 scd_101570.2.3.1.0]" user could create vector group Dyn9. Similarly all vector groups with either star on delta connected windings can be modelled.

In original network data there are three different vector groups for distribution transformers: Dyn11, ZNzn0 and Yzn11. Modelling Dyn11 vector group is easy as shown before, as for the other two groups modelling is not that simple. OpenDSS lacks internal winding definition for Z-connected (Zigzag, Interconnected star)

windings. This may be due to the fact that Z-connected transformers are nearly non-excitant in US distribution networks. The only way to model these transformers is to use three single phase, 3 or 4-winding transformer models which are then interconnected accordingly. Modelling an Yzn11 connected transformer is explored in appendix C.

For each transformer a voltage recording element is also defined. In OpenDSS Monitor elements are used to record data during multistep simulations. Monitors record a sample per step of wanted quantity, in this case voltage magnitude, from the circuit element terminal they are attached. For the example transformer above a following Monitor element is defined:

```
New Monitor.mon_Siltaoja element=Transformer.Siltaoja terminal=1 mode=32
```

The parameter terminal=1 refers to transformers high voltage side connection terminal and mode=32 refers to operation mode for the monitor. In this case operation mode is to record voltage and current magnitudes for each phase (see OpenDSS Reference Guide, page 143, for other operation modes). Accessing the post-simulation data recorded by the Monitor is discussed in chapter “Reading simulation data”.

Loads and time series

For this model network loads are connected directly to distribution transformers’ secondary circuits. Loads are modelled whit Load and LoadShape objects, first of which represents the actual load and second of which represents variance in load during time. All loads are assumed symmetrical, grounded star connected with unity power factor.

All Loads and LoadShapes are defined in loadsV.dss-file. Because of the way OpenDSS handles linking these objects LoadShape for a certain load must be defined before the actual Load object.

For example a load for the transformer discussed in previous chapter would be defined followingly. First we define the LoadShape:

```
new LoadShape.LS_V_101570 npts=8784 csvfile=".\LS_V_101570.csv"
~ useActual=Yes
```

Name of the defined LoadShape is the transformer station id (also used in LV-side bus naming) with “LS_V_” prefix. The parameter npts refers to a number of elements in the time series and the parameter csvfile refers to the file in which the actual loading values are. Lastly, useActual=Yes mean that OpenDSS uses the values in the .csv file as actual values for the load, not as multipliers for base load value as it would normally do.

Then we define the actual load:

```
new Load.Load_101570 Phases=3 kv=0.4 kw=0 pf=1 conn=wye
~ bus=scd_101570.1.2.3.0 Yearly=LS_V_101570
```

With declaration “Yearly=LS_V_101570” the defined load uses the predefined LoadShape’s power values for yearly simulation studies.

The Loads and LoadShapes in file loadsV.dss are composed from AMR measurement data collected from customers of Orivesi MV-grid from time period from November 2011 to October 2012. From this measured data the .csv files for each load are calculated by summing the loads of every customer behind a certain transformer station.

If for some reason this model is to be distributed without the measured loadings another file for load definitions is also created: blank_loads.dss. In this file Load and LoadShape objects are defined similarly but with zero load values for every hour. In order to run simulations using this load file user must define these values. See chapter “Modifying time series” how to insert load values for LoadShape objects from MATLAB.

Finalizing the model (master script file)

In order to make the model complete a master script file must be created. In this file Thevenin source feeding the system and main substation transformers are defined, and after that rest of the model files are included and compiled into the program.

The master script file is relatively short compared to rest of the model script files. It only has c. 30 lines whereas rest of the files have between 1000 and 3000 lines. The master script file is depicted in figure 3.

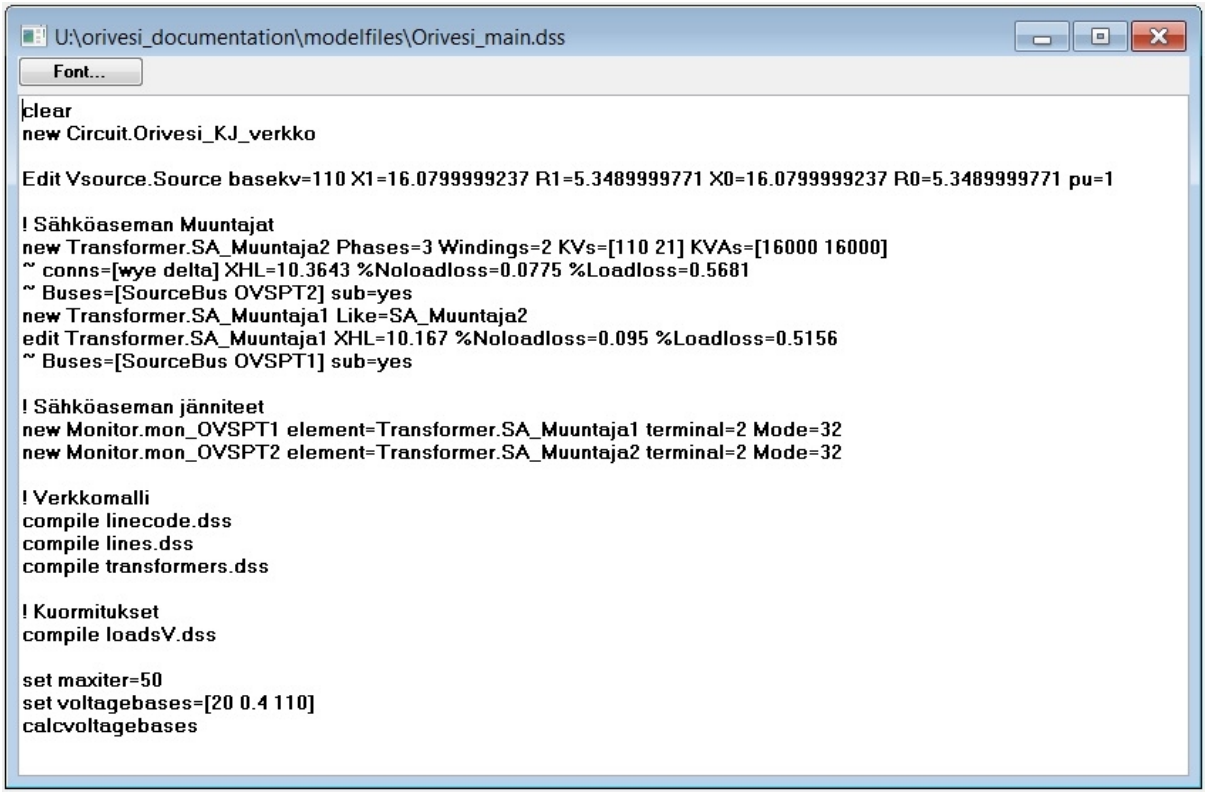

Figure 3: Master script file

The master script begins with “clear” command. This clears the program from any previous circuit definitions and it is considered to be good practice to begin every master script file with this command. After clearing a new circuit is defined. Defining new circuit automatically results in definition of system feeding voltage source “Source”. This source is intended to represent distribution networks connection point to transmission network. In order to accurately represent the feeding transmission network voltage and series impedance values for the source must be defined. Here it’s assumed that voltage of the feeding system is constant at its nominal value 110 kV.

In the next section main substation transformers are defined. Parameter values for definition are calculated similarly as in distribution transformer definition. Here it is assumed that substation transformers do not have tap changers and they operate at their nominal voltage ratios. Also, two monitor are defined which record voltage magnitudes of transformers’ secondary circuits.

After transformer and monitor definitions rest of the network model (lines and transformers) is included and compiled to the program. After this the transformer loads are included. If the user wishes, this file can be changed to “blank_loads.dss”. Here it’s assumed that rest of the script files are in the same directory as the master script file.

In the last section maximum iterations per solution are defined. Here the number of allowed iteration is 50. The value is relatively high, normally being around 15. This is because it would seem that circuits containing Zigzag transformers are sometimes particularly difficult to solve. This however has no significant effect on simulation time as OpenDSS only does as many iterations as required. Normally solution converged in 3-4 iterations during test simulations. After defining maximum iterations base voltages for the model are defined. After the allowed voltage bases are defined calcvoltagebases-command is issued. This invokes a zero-power flow solution in the circuit which OpenDSS uses to define base voltage for every element.

Using the model with MATLAB through COM-interface

Preparations

When using OpenDSS user may choose whether to run simulations using the OpenDSS executable or using OpenDSS simulation engine from 3rd party software (for example MATLAB or Excel) through COM-interface. If the system which is simulated is simple the executable is adequate in most cases. However when simulating complex systems with multiple recording elements (like in this system), it is only reasonable to use OpenDSS simulation engine from the software one would use for the result processing. In this chapter using OpenDSS simulation engine from MATLAB is discussed.

In order to use OpenDSS simulation engine the OpenDSS software must be first installed. Currently there are two distributions of the software: standalone executable and installer pack. If the installer pack is used no other preparations in addition to running the installer are required. And if the standalone executable is used the OpenDSS engine must be registered. There are comprehensive documentations on OpenDSS wiki site for the registration (http://sourceforge.net/apps/mediawiki/electricdss/?source=navbar). For most systems registration can be done in a following way:

1. Run the command prompt (as administrator in windows XP, Vista and 7) and go to the directory you have the OpenDSS's .dll files.
2. Run command: regsvr32 OpenDSSEngine.DLL

After this the OpenDSS simulation engine is ready to be used from other softwares.

Running standard (OpenDSS native) simulations

In order to run simulations from MATLAB an instance of OpenDSS engine must be created, then the simulation engine must be started and lastly the model must be compiled. After this user may use the OpenDSS command interface to issue simulation commands pretty much like one would use the executable's interface. In MATLAB the creation of the engine instance is done by commanding

```
DSSObj = actxserver('OpenDSSEngine.DSS');
```

And the engine is started by invoking the starting method:

```
DSSStart = DSSObj.Start(0);
```

This method should return an integer 1. If not then for some reason the simulation engine failed to start. Now, in order to issue commands to OpenDSS engine an interface for text command must be created. This is done by commanding:

```
DSSText = DSSObj.Text;
```

This interface has two properties Command and Result. The Command property is used to issue the text commands and the Result property is used in reading the results of some commands (for example after issuing an export command the exported filename will appear in the Result property). Next step is to compile the model. This is done by compiling the master file which should if properly created contain compilation commands for the rest of the model. From MATLAB commands is issued followingly:

```
DSSText.Command = ...
    'compile U:\orivesi_documentation\modelfiles\Orivesi_main.dss';
```

Now, here it's important to specify the full file path of the master script file otherwise the command may not have the wanted result.

Now finally the simulation engine is ready to take simulation commands. In the following chapter running a yearly power simulation is explored.

Example

We have now run the basic steps to enable simulation running. Let's consider that we have compiled the model with loadsV.dss file so that we have some meaningful values for transformer loads. We would now like to run a yearly power flow simulation so that we would have transformer station voltage magnitudes for every hour. As the model and the loading conditions are symmetrical we are only interested in the voltage magnitudes of the first phase in the system (we will assume that voltage magnitudes for other phases are same). The MATLAB code for running this simulation is depicted in the figure 4.

simulointiajo.m

```
1   clear all;
2
3   DSSObj = actxserver('OpenDSSEngine.DSS');
4   DSSStart = DSSObj.Start(0);
5   DSSText = DSSObj.Text;
6
7   if DSSStart ~= 1
8       error( 'DSS engine failed to start\n' );
9   end
10
11  DSSText.Command = ...
12      'compile U:\orivesi_documentation\modelfiles\Orivesi_main.dss';
13  DSSText.Command = 'set mode=yearly';
14  DSSText.Command = 'set number=8784';
15  DSSText.Command = 'set hour=0';
16  DSSText.Command = 'solve';
17
18  fprintf( 'Simulation Done\n' )
19
20  DSSMons = DSSObj.ActiveCircuit.Monitors;
21
22  Mon_names = {};
23  Voltages = [];
24  ind_mons = DSSMons.First;
25  % luetaan monitoreista a-vaiheen jännitteet yhteen matriisiin
26  while ind_mons > 0
27      mon_data = lueMonitoridata( DSSMons.bytestream );
28      % ei oteta nimeen mukaan monitorin tunnusta 'mon_'
29      mon_name = {DSSMons.name(5:end)};
30
31      Mon_names = cat( 2 , Mon_names, mon_name );
32
33      % Monitori mittaa vaihejänniteitä ja talletaa jännittet voltteina
34      % Muunnetaan jänniteet (vaihejännite, V) -> (pääjännite, kV)
35      Voltages = cat(2, Voltages, mon_data.data(:,3)*sqrt(3)/1000);
36
37      ind_mons = DSSMons.Next;
38  end
```

Figure 4: MATLAB code for yearly power flow simulation.

The initializing steps described in previous chapter are taken on line 1 to 12. On lines 13 to 16 Simulation parameters are set (multistep yearly simulation, 8748 simulation steps with default step length of one hour, beginning from hour 0) and solve command is issued.

After the simulation is done an interface to recording elements (Monitors) is issued and matrixes for results are initialized. Then the data recorded by the monitor is read from the monitor bytestream and the monitor name is inserted into the name matrix. From the data recorded by the monitor the matrix column representing line-to-neutral voltage of the first phase is selected, all of its values multiplied by √(3)/1000 (in order to

represent line to line voltages in kV), and the resulting vector is inserted to the voltage matrix. This is done to all monitors in the circuit.

At the end of the simulation we have to two matrixes containing result values: “Voltages” and “Mon_names”. “Voltages” is *number_of_hours* by *number_of_monitors* matrix which contains the line-to-line voltage values for every transformer station at every hour in the simulation. “Mon_names” contains the monitor names in single row. They bind the column in the Voltage matrix to a certain monitor name.

Modifying time series

In order to run different simulations user has to be able to modify the power values in the time series for loads. For this purpose a MATLAB function ModifyLoadShapeArray was written, and it is to be supplied with the model script files. The syntax of the function is:

```
function return_bool = ModifyLoadShapeArray( DSSObj ,
Array_of_Powers, LSname )
```

In which, the parameters are:

- DSSObj, the interface to the OpenDSS engine.
- Array_of_Powers, n by 1 array of real powers for the load time series.
- LSname, the name of the LoadShape object the time series is to be inserted.

The function returns logical true if the LoadShape with the given name was found, else logical false.

It’s important that the user specifies correct sized power array. For example, if user wishes to run simulation for 1000 hours the user must specify 1 by 1000 array for every LoadShape in the model and then specify the number of solutions in the simulation parameters to be 1000. If for some reason user specifies this number of simulations to be something else, like 2000 then OpenDSS would start to repeat the time series after the first 1000 solutions.

In order to modify the power factor of a load user must issue an Edit command trough DSSText interface. For example, the command:

```
DSSText.Command = ‘Edit Load.Load_101570 pf=.9’
```

would edit the power factor of ‘Siltaoja’ transformer’s load to be 0.9.

Reading simulation data

In order to efficiently read the data recorded by the monitors a MATLAB function lueMonitoridata was written. This function is also to be supplied with script files. The syntax for function is following:

```
function output = lueMonitoridata( bytestream )
```

In which, the parameter is a bytestream from Monitors interface.

The function returns an output structure which contains following fields:

- output.signature, signature byte for the monitor (must be 43756)
- output.version, program version
- output.recsize, number of recorded quantities
- output.mode, operation mode of the monitor
- output.header, string containing names for recorded values
- output.data, a matrix containing the values recorded by the monitor in same syntax as in exported .csv file

See the example simulation in the Example chapter for proper use for this function.

Appendixes

A: An example of line and busbar data

Table 1: A single line on line data table.

ID	Laji	x1	y1	x2	y2	Pituus (m)	Kytkimen tila 1	Kytkimen tila 2	Tunnus
1290115	Pg99 3x85/14 Al/FE Pigeon	6842551.105	2518085.41	6842983.598	2518216.281	546.1267865	0	0	Pg99

Table 2: A single line on busbar data table.

ID	Laji	x1	y1	x2	y2	Pituus (m)	Kytkimen tila 1	Kytkimen tila 2	Tunnus
1319847	MMO-kisko-osa näkymätön SJ	6842983	2518216	6842983	2518216	0.03125	0	0	MMO-kisko-osa

B: An example of transformer data

Table 3: A single line on distribution transformer data table

ID	Laji	Father ID	x	y	Mitoitusteho (kVA)	Kytkentäryhmä	Kuormitushäviöt (W)	Ensiöpuolen mitoitusjännite (V)	Tyhjäkäyntihäviöt P0 (W)	Toisiopuolen mitoitusjännite (V)	Oikosulkuimpedanssi (%)
1319849	Jakelumuuntaja	1319838	6842983	2518216	315	Dyn11	3650	20000	610	400	4

Table 4: A single line on transformer station data table

ID	Laji	x	y	Tunnus	Nimi
1319838	2-pylväsmuuntamo	6842984	2518216	101570	Siltaoja

Note: the value in the FatherID field binds an individual distribution transformer to transformer station

C: Modelling Yzn11 transformer in OpenDSS

Let's consider an Yzn11 transformer with following nameplate values:

U1/U2	20/.4kV
S_N	100 kVA
$P_{Loadloss}$	2.51 kW
$P_{Noloadloss}$	269 W
z_K	6 %

Winding configuration and phasor group for this transformer would be as shown in figure 5:

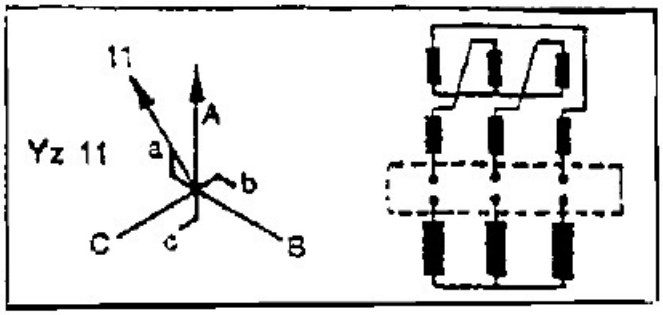

Figure 5: Phasor group and winding connections of Yzn11-transformer

As OpenDSS doesn't have internal model for Z-winding configuration we must use interconnected single-phase transformers for modelling. The winding configurations for these transformers is depicted in figure 6.

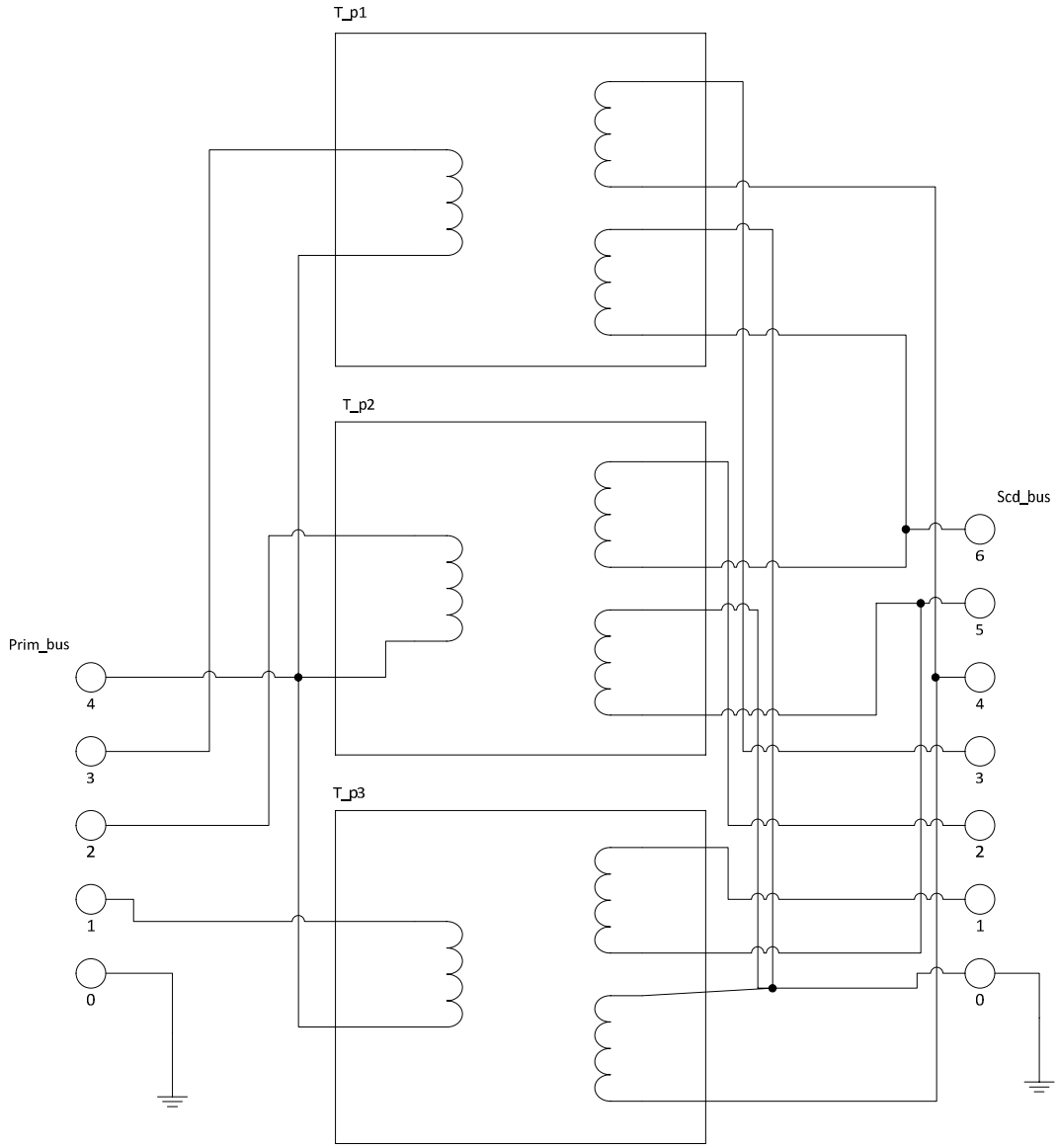

Figure 6: Winding configuration for Yzn11 transformer.

And in script this would look:

```
new Transformer.T_p1 Phases=1 Windings=3 kVs=[11.547005 0.133333 0.133333]
~ kVAs=[33.333333 16.666667 16.666667] XHL=5.994748 XHT=5.994748 XHT=5.994748
~ %Noloadloss=0.269000 %Rs=[1.255000 1.255000 1.255000]
~ wdg=1 bus=prim_bus.1.4 wdg=2 bus=scd_bus.1.5 wdg=3 bus=scd_bus.0.4
new Transformer.T_p2 Like=T_p1
new Transformer.T_p3 Like=T_p1
Edit Transformer.T_p2 wdg=1 bus=prim_bus.2.4 wdg=2 bus=scd_bus.2.6 wdg=3
```

```
~ bus=scd_bus.0.5
Edit Transformer.T_p3 wdg=1 bus=prim_bus.3.4 wdg=2 bus=scd_bus.3.4 wdg=3
~ bus=scd_bus.0.6
New Monitor.mon_T element=Transformer.T_p1 terminal=1 mode=32
```

The parameters values are calculated similarly as in chapter “Composing the model \ Transformers” but here 1-phase powers and line to neutral voltages are used. The three transformers are declared to ha same parameter values and then the winding connections are specified to be as in figure 6.

The list of required files:

Orivesi_main.dss	Main model script
lines.dss	Line definitions
loadsV.dss	Load definitions, from measurement data
blank_loads.dss	Load definitions, zero loading
transformers.dss	Transformer definitions
linecode.dss	Conductor type definitions
modifyLoadShapeArray.m	MATLAB function for load time series modification
lueMonitoridata.m	MATLAB function for monitor bytestream reading