Proposal of Circuit Description Language

Taku Noda
CRIEPI - Central Research Institute of Electric Power Industry
2-11-1 Iwado-kita, Komae-shi, Tokyo, 201-8511, Japan
takunoda@criepi.denken.or.jp

Abstract – **This paper proposes a language to describe the information of a circuit for computer-aided analysis and design. The syntax of the proposed language, named Circuit Description Language (CDL) in this paper, provides strong modularity which enables to build a circuit using previously defined other circuits (building-block construction of a circuit). This feature allows to build a library of standard power-system models using CDL. Example circuits are described using CDL in order to show the power of modularity.**

Keywords : circuit, language, circuit analysis program, modularity, encapsulation, self-documenting, maintenance, standard model library.

I. INTRODUCTION

Computer-aided analysis and design of a power system, such as transient, stability, and load-flow analysis, become common practice these days. In order to perform those simulations, the complete information of a circuit (being analyzed), that is the topology and the value of every circuit element, has to be described by a certain method to pass the information to a circuit analysis program. Two major methods have been used for the description.

One is to type the value and the both ends of every circuit element using a certain format in a text file. For example, "branch cards" of EMTP requires to type the above items in specified columns, and "netlist" of SPICE in a blank-separated format. This method is efficient, because the circuit analysis program can directly read the formatted text file with a little amount of coding. However, this method is not intuitive for human beings, and the ability of self-documenting is quite low. Thus, imaging the entire circuit from the formatted text file is very difficult. Describing a large circuit may also be troublesome due to the weakness of modularity, where "modularity" is used to indicate the ability to group a subcircuit and to reuse it (as a model) in different portions with different parameters. It could be said that this method is not for users but for computer programs.

The other method is the use of GUI (graphical user interface) that allows to draw the diagram of an analyzed circuit on screen using a graphical window system and a pointing device. This method is quite intuitive for human beings and provides an easy way to select several parts for grouping them as a subcircuit, and thus it provides a sort of modularity. One problem of this method is machine or OS (operating system) dependence. And the most important point is that even the GUI method requires another description method when passing the information to the circuit analysis program and also when saving the information as a disk file. In other words, the picture of a circuit diagram itself cannot be passed or saved without converting into another description.

Considering the above, the author proposes a language approach and has created the Circuit Description Language (CDL). The syntax of CDL provides strong modularity, and the features are summarized as follows :

1 It is allowed to define a subcircuit using previously defined other subcircuits. Subcircuit of subcircuit, i.e. the nesting of the subcircuit definition is also allowed.
2 A subcircuit can be used in different places with different parameters.
3 Internal node and variable names are encapsulated inside a subcircuit and cannot be accessed from outside.
4 Circuit description itself is documenting the circuit. The description is in a free format, and additional comments are allowed in any places.

Feature 1 allows the building-block construction of a circuit, i.e. a complicated model can be built level by level from a microscopic level to a macroscopic level. The development of a more general and reusable model is made possible by feature 2. The ability of model maintenance is improved by feature 3, because this prevents careless or other-users' change of internal variables and releases users from the troublesome concern of the duplication of node names. Feature 4 enables a user to intuitively image the inside of a circuit by a glance of the description. Due to the above advantages, a library of standard power-system models (subcircuits) can easily be built using CDL, and those models can safely be used by users who even do not know their inside. Also, CDL can be used as a standard output format of different GUI circuit editors, because CDL is cross-platform.

A translator "cdlparse", which translates from CDL to netlist-like format, has been developed. And the CDL code of example circuits is illustrated in order to show the power of modularity.

II. SYNTAX OF CDL

The syntax of CDL is mostly from structured programming languages such as Pascal [1], C/C++ [2,3] and so on,

especially from the C language. CDL uses a free format, and thus the end of a line is treated as same as a white space. Comments can be inserted in any places, and they are enclosed by /* and */. A name, used to identify a circuit element, node, or variable, is an alphanumeric string staring with an alphabet character. CDL makes a distinction between lower and upper case characters.

A circuit is defined in the following form :

```
circuit circuit name
  {
  circuit definition
  }
```

First comes keyword **circuit**, and the name of the circuit follows. The name is used to identify this circuit, when it is used in other circuits. If the name is **main**, the circuit is the outer-most circuit that contains all other circuits, and therefore the main circuit is usually defined at the bottom of a file. The definition of the circuit is described between braces { and }. The circuit definition consists of the following declarations and arithmetic value substitutions.

1 *Node Declarations* declare the names of nodes used in the circuit. Nodes accessible from outside, i.e. nodes which can be connected to ones outside of the circuit or of which the voltages and/or current can be referred as an output, are declared as **terminal**. The other nodes, which are just used for internal connection, are declared as **node**. (e.g. `terminal T1, T2; node N;`)

2 *Variable Declarations* declare the names of numerical variables used in the circuit. Variables accessible from outside, i.e. of which the value can be changed from outside of the circuit, are declared as **parameter**. The other variables, which are just used for internal calculation, are declared as **variable**. (e.g. `parameter a, b, theta; variable x;`)

3 *Element Declarations* declare the names of circuit elements used in the circuit. Elements such as **resistance**, **inductance**, **capacitance**, **voltage source**, and **current source** are prepared as predefined elements. A previously defined other circuit can be used as a circuit element in the form of **circuit** *circuit name*:. (e.g. `resistance Rx; circuit line_model: line1, line2;`)

4 *Connection Declarations* describe the connection of elements in the circuit in the form of **connect** *element name* { *node and value substitutions* }. A node substitution is described as *node name of connected element* -> *node name of this circuit*. In case of a non directional two-terminal element or a one-terminal element, the from can be abbreviated as -> *node name of this circuit*. A value substitution is described as *variable name* = *arithmetic expression*. (e.g. `connect Rx { value = 10.0; -> T1; -> N })`

5 *Arithmetic Value Substitutions* are in the form of *variable name* = *arithmetic expression*.. (e.g. `x = a + b*sin( theta*pi/180 );`)

The above declarations and substitutions are placed in arbitrary order between the braces, but the name of a node, a variable, or a circuit element has to be declared before used. Usually, a file contains one or more circuit definitions. If circuit A uses circuit B, then B has to be declared before A is declared, and therefore the main circuit comes at the bottom of a file. The name of a CDL file usually ends with extension .cdl.

CDL has a mechanism to include a file at a specified place of another file. As used in the C language, **#include** is used. If `#include "file2.cdl"` is found at a place in `file1.cdl`, then file2.cdl is inserted at the place of file1.cdl. This is very useful as follows. Assume that a transformer model **circuit transformer** has been developed and described in file "trans.cdl". Then, one can use the model by writing `#include "trans.cdl"` just before it is used. Furthermore, if a set of general models used in substations are described together in file "ss.cdl", one can use them by writing `#include "ss.cdl"` at the top of a file.

In CDL, **pi**, **eps0**, and **mu0** are predefined constants of which the values are $\pi = 3.1415\ldots$, $\varepsilon_0 = 8.854 \times 10^{-12}$, and $\mu_0 = 1.257 \times 10^{-6}$ in MKSA units respectively. **GND** is reserved for the node name of the ground that has absolute zero potential. CDL also has basic arithmetic functions, such as **sqrt**, **exp**, **sin**, **cos**, and so on, which can be used in arithmetic expressions.

III. SAMPLE DESCRIPTION

This chapter shows sample description of example circuits in order to show the syntax of CDL and the power of modularity. The first example is a rather simple circuit to illustrate the concept of modularity. And the second one is for the illustration of practical use.

A. Example 1 – Double Resonance Circuit

Fig. 1 (a) shows a simple series *RLC* circuit. Using CDL, this simple circuit is described as

```
circuit series_RLC
  {
  terminal T1, T2;
  node N1, N2;
  parameter Rval, Lval, Cval;
  resistance R;
```

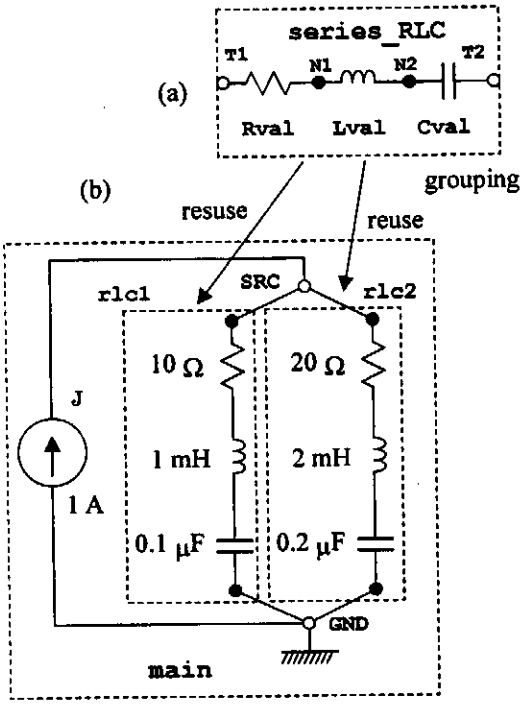

Fig. 1 Example 1 – double resonance circuit

```
inductance L;
capacitance C;
connect R { value = Rval; -> T1; -> N1; }
connect L { value = Lval; -> N1; -> N2; }
connect C { value = Cval; -> N2; -> T2; }
}
```

Two terminals T1 and T2 and two internal nodes N1 and N2 are declared, and resistance R, inductance L, and capacitance C are connected between T1 and N1, N1 and N2, and N2 and T2 respectively. The values of R, L, and C are remained as parameters Rval, Lval, and Cval.

Next, a double resonance circuit shown in Fig. 1 (b) is described using the previously defined circuit series_RLC as follows.

```
circuit main
  {
  terminal SRC;
  current source J;
  circuit series_RLC: rlc1, rlc2;
  connect rlc1 {
      Rval = 10.; Lval = 1.E-3;
      Cval = 0.1E-6; -> SRC; -> GND; }
  connect rlc2 {
      Rval = 20.; Lval = 2.E-3;
      Cval = 0.2E-6; -> SRC; -> GND; }
  connect J {
      value = 1.0;   -> SRC; -> GND; }
  }
```

Because this is the outer-most circuit, the circuit name is main. Node SRC is declared as a terminal for voltage output. Two instances of series_RCL are generated as rlc1 and rlc2, and used in different portions with different parameters. The values of R, L, and C of rlc1 are set to 10 Ω, 1 mH, and 0.1 μF respectively. On the other hand, those of rlc2 are 20 Ω, 2 mH, and 0.2 μF. And both of them are connected between SRC and the ground. This can be described as T1 -> SRC; T2 -> GND; without abbreviation. Additionally, current source J is connected in parallel.

It should be noted here that the "modularity" of CDL is effectively used to describe the present circuit. First a simple series *RLC* circuit is described and grouped as series_RLC, and then two instances of the circuit are used as different portions of the main circuit with different parameters. This building-block construction of a circuit makes easy to build a large complicated circuit. It should also be noted that internal node names N1 and N2 are encapsulated in series_RLC and no care is required when building the main circuit.

B. Example 2 – Pi Line Model

Fig. 2 shows one-section pi representation of a single phase transmission line. Using CDL, the line model is described as follows.

```
circuit pi_line
  /* transmission-line modal,
     one-section pi representation */
  {
  terminal      /* sending and receiving  */
    SND, REC;   /* end terminals          */
  node MID;     /* internal node          */
  parameter     /* radius r and height h  */
    r, h,       /* of conductor in [m]    */
    Rs, l;      /* series loss and length */
                /* [ohm/m]        [m]     */
  variable X;   /* common log term        */
  resistance R;  /* series resistance     */
  inductance L;  /* series inductance     */
  capacitance
    C1, C2;       /* shunt capacitance    */
  X = log( 2*h/r );
  connect  R { value = Rs*l;
    -> SND; -> MID }
  connect  L { value = 1*mu0/(2*pi)*X;
    -> MID; -> REC }
  connect  C1 { value = 1*pi*eps0/X;
    -> SND; -> GND }
  connect  C2 { value = 1*pi*eps0/X;
    -> REC; -> GND }
  }
```

The sending- and receiving-end nodes of the line model are declared as terminals SND and REC, and an internal node MID is declared. The radius r [m], height h [m], series loss

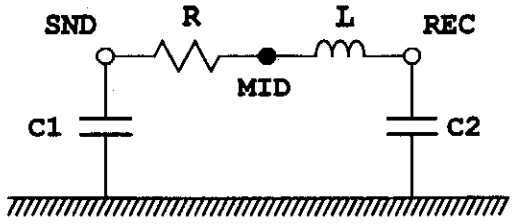

Fig. 2 Example 2 – pi line model

`Rs` [Ω/m], and length `l` [m] of the line are declared as parameters. Those parameters are the geometrical information of the line (`Rs` is the earth-return loss and thus can be considered also as geometrical information). The series resistance `R` and inductance `L` and the shunt capacitance `C1` and `C2` are calculated by the following formulas. A variable `X` is used to calculate the values of `L`, `C1`, and `C2` from the geometrical information.

$$R = R_s l \tag{1}$$

$$L = \frac{\mu_0}{2\pi}\log\left(\frac{2h}{r}\right)\cdot l \tag{2}$$

$$C_1 = C_2 = \frac{\pi\varepsilon_0}{\log\left(\frac{2h}{r}\right)}\cdot l \tag{3}$$

The values of the circuit elements `R`, `L`, `C1`, and `C2` are internally calculated, and the calculation is encapsulated inside the line model. Therefore, a user, who even do not know the internal calculation, can use the model just by giving the geometrical information `r`, `h`, `Rs`, and `l`. This is also one of the advantages of modularity. It should also be noted that one (who is familiar with the C language) can easily understand the circuit description with the aid of comments, and thus the code is self-documenting.

Fig. 3 shows a circuit consisting of two transmission lines

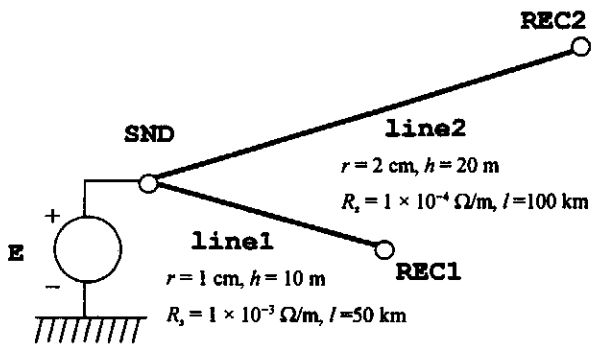

Fig. 3 Two transmission lines connected to a voltage source

connected to one voltage source. The name of the shorter line is `line1`, and its length is l = 50 km (r = 1 cm, h = 10m, $R_s = 1 \times 10^{-3}$ Ω/m). The other line is `line2`, and l = 100 km (r = 2 cm, h = 20m, $R_s = 1 \times 10^{-4}$ Ω/m). This circuit is described using CDL as follows.

```
#include  "pi_line.cdl"

circuit main
  {
  terminal SND, REC1, REC2;
  voltage source E;
  circuit pi_line: line1, line2;
  connect E { value = 1.0: -> SND; -> GND }
  connect line1 {  /* 50 km line */
    r = 1.E-2, h = 10.;
    Rs = 1.E-3; l = 50.0E3;
    SND -> SND; REC -> REC1 }
  connect line2 {  /* 100 km line */
    r = 2.E-2, h = 20.;
    Rs = 1.E-4; l = 100.E3;
    SND -> SND; REC -> REC2 }
  }
```

The above code assumes that circuit `pi_line` is described in file "pi_line.cdl". Three terminals `SND`, `REC1`, and `REC2` are first declared. Two instances of circuit `pi_line` are generated as `line1` and `line2`. `line1` is connected between `SND` and `REC1` as a 50-km line, and `line2` between `SND` and `REC2` as a 100-km line. A line model `pi_line` is used in two different places with different parameters. As mentioned in the first example, this is the power of modularity. A voltage source `E` is connected between `SND` and the ground.

From those two examples, the advantages of CDL are summarized as follows. General, reusable, and maintainable models can be described using CDL due to its strong modularity, which provides the mechanism of the building-block construction of a circuit and the encapsulation of internal node, variable, and element names. Line model `pi_line` is a good example to show that CDL has an enough power to describe a library of standard power-system models.

IV. TRANSLATOR "cdlparse"

A translator called "cdlparse" has been developed by the author. The program parses a CDL file, evaluates expressions and node connections to build the entire circuit, and finally translates into netlist-like branch output. The following is the output of cdlparse when the first example – double resonance circuit – is parsed.

```
R, 1, 2,  1.00000E+01
L, 2, 3,  1.00000E-03
C, 3, 0,  1.00000E-07
R, 1, 4,  2.00000E+01
```

```
L, 4, 5,  2.00000E-03
C, 5, 0,  2.00000E-07
I, 1, 0,  1.00000E+00
number of independent nodes = 5
```

First comes element type, and two node numbers to which the element is connected follows. Last comes the value of the element. One line corresponds to one element. The last line shows the number of independent nodes for a circuit analysis program (e.g. for determining the size of the nodal admittance matrix). This sort of branch output can easily be read by existing circuit analysis programs.

V. CONCLUSIONS

A new circuit description approach – language approach – has been proposed : Circuit Description Language (CDL). The syntax of CDL provides strong modularity which allows to build a library of standard power-system models. The power of modularity has been illustrated using the sample code of example circuits.

The incorporation of mutually coupled elements and ideal transformer into the CDL syntax is considered as the next important work. Also, interface with programming languages may be a future work so that a user-developed model can be incorporated as a circuit in CDL.

VI. AKNOWLEDGMENTS

The author is grateful to Laurent Dubé of DEI Simulation Software for his lead to this field of research.

REFERENCES

[1] K. Jensen, N. Wirth, A.B. Mickel, "Pascal User Manual and Report : ISO Pascal Standard, 4th edition," Springer Verlag, 1991.
[2] B.W. Kernighan, D.M. Ritchie, "The C Programming Language, 2nd edition," Prentice Hall, 1988.
[3] B. Stroustrup, "The C++ Programming Language, 3rd edition," Addison-Wesley, 1997.