CAPE Creation of Doble State-Simulation Data Files for End-to-End Relay Testing

Russell W. Patterson

Tennessee Valley Authority
Chattanooga, Tennessee USA

Presented before the

52nd Annual Georgia Tech
Protective Relaying Conference
Atlanta, Georgia
May 6th – 8th, 1998

Abstract

This paper discusses the development of programs in the CAPE (Computer Aided Protection Engineering) environment that are used to create Doble state simulation data files for exhaustive end-to-end testing of transmission line relaying. The programs were created using the CUPL (CAPE Users Programming Language) and run interactively in the CAPE environment.

End-to-end testing allows for the complete testing of the protective relaying system. The program output contains fault currents and voltages that are as accurate as the network model used for protective relay setting. The resulting data files can be used directly by the test engineers whereas they previously had to create them by hand from fault study data provided by the relay engineer. This "by hand" method was error prone and the faults included were limited by the amount of time required for data file creation.

The CUPL programs default case applies the following list of faults to six different locations (four in-zone and two reverse locations):

- A-phase to ground
- B-phase to ground
- C-phase to ground
- A-phase to B-phase fault
- B-phase to C-phase fault
- C-phase to A-phase fault
- Three-phase fault

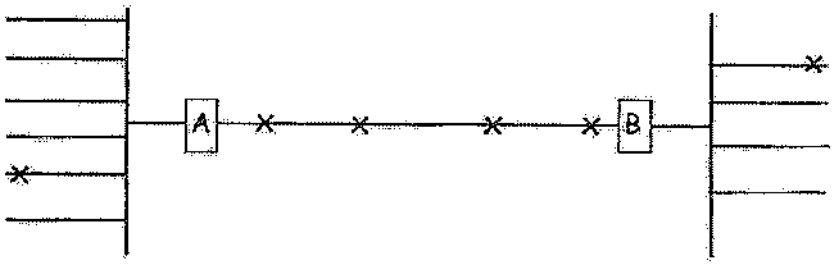

Figure 1. Six default locations for fault application.

Doble State Simulation Files

These data files are plain text files (see appendix A for sample listing) containing voltage and current source data as well as simulation ordering information. These simulations are run on the Doble test equipment (current and voltage sources) using Doble's Protest III software.

Each data file can contain many simulations. Each simulation can contain several different states. For example, the simulation data file 356.SS1 may contain 42 different simulations (there are 42 simulations created in a default run). The first simulation might be an A-G fault applied 10% down the line from bus 356 toward bus 357. This simulation would most likely consist of three states; pre-fault, fault, and post-fault. Each of these three states would contain the voltage and current source magnitudes and angles expected for that state (the pre-fault might be expected load currents and voltages etc.). As well as voltage and current source values the data file contains information needed for simulation control and timing to coordinate with tests being run at other locations for simultaneous end-end tests.

CAPE (Computer Aided Protection Engineering)

CAPE is a grouping of various related and interacting "modules" for power system protection and analysis. These include:

- Data Base Editor
- Short Circuit
- One-Line Diagram
- Coordination Graphics
- Relay Setting
- Relay Checking
- Line Constants
- Order Production
- Short Circuit Reduction
- Power Flow

CAPE is the environment in which the CUPL code described in this paper is executed. This CUPL code is dependent on the short circuit engine of the Short Circuit module of CAPE. CAPE runs in the PC environment on the Windows 95 or Windows NT operating systems.

More information on CAPE can be found at the Electrocon homepage on the world-wide-web (http://ic.net/~eii/).

CUPL (CAPE User's Programming Language)

CUPL includes commands normally found in high-level computer languages, commands such as IF-THEN-ELSE, DO-WHILE, and DO loops such as DOBUSES and DOLINES that are specifically designed for power system problems. CUPL commands are similar to commands in the BASIC programming language.

CUPL lets you customize the use of any CAPE module to perform an almost limitless number of useful, repetitive tasks without extensive user intervention. For example, CUPL macros are easily prepared to perform automated fault studies. Elsewhere in CAPE, macros are already available to compute fault locations, reaches of instantaneous overcurrent and distance elements, transformer circulating currents, the total complex impedance of multi-segment lines, and relay settings, to name a few.

The Code

The CUPL programs discussed in this paper run solely in the Short Circuit module of CAPE and only make use of the network fault calculation abilities of the Short Circuit module. This design path was taken due to the fact that when the need for this program was identified only portions of the TVA system relay information had been entered into the database. The goal then was to be able to create the Doble simulation files without the particular relay data being in the database. As long as the system impedance elements (lines, transformers etc.) had been modeled simulation cases could be created.

Future enhancements will include actual relay connection information taken from the CAPE database (made possible by recent enhancements to CUPL). Use of the database will allow for the creation of data files with a minimum of user knowledge of the actual relay and ct/pt configuration data (if the relay connection data is complete and accurate in the database). This will allow correct modeling of polarization quantities when they are created from several different sources (neutrals of various transformers etc.). This approach will allow for the exact quantities that CAPE "thinks" the relay will see to be put in the simulation files, even if several layers of auxiliary ct's or remote pt's are used. If it can be modeled and simulated in CAPE it can have a Doble simulation file created.

Example blocks of the CUPL program code are given in Appendix B.

Dynamic Testing

The Doble test setup can be used effectively to simulate system conditions to check the relay and a board wiring, relay operation, relay settings, coordination of auxiliary relays, and coordination of pilot, permissive, and other relaying schemes between both ends of a transmission line. The following is a brief list of some problems found during dynamic testing in the TVA system:

- Broken wire in a reclosure circuit
- Incorrect timer setting causing coordination problem in an electromechanical to microprocessor relaying scheme
- Incorrect carrier jumper setting on microprocessor relay on new installation
- Sneak circuit through electromechanical relaying package
- Improper scheme selection setting on microprocessor relay

- Intermittent grounded output on a carrier circuit board
- Grounded switch on a current circuit
- Microprocessor relay setting problem
- Timer circuit design problem
- Incorrect option setting on microprocessor at new installation
- Contact bounce on a carrier relay

The dynamic tests are performed with Doble tests sets (voltage and current sources) at each end of a transmission line terminal. The test sets are controlled by Doble's Protest software. Time synchronization (for end-end simultaneous testing) is achieved with a GPS (Global Positioning Satellite) computer card. This allows the test setups at each end of the line to start and run in synchronism to simulate the actual voltages & currents the relaying scheme will see as a whole. This simulates actual voltages and current flows during a fault simultaneously at both line ends.

Dynamic testing with the data files generated with these CUPL programs in CAPE allows exhaustive testing of the relaying scheme and relay settings as well as complete scheme equipment (carrier relays, carrier equipment, auxiliary relays etc.).

Conclusion

The virtues of dynamic testing are clear. Dynamic testing coupled with the ability to compute large numbers of standard system simulation cases gives the field test personnel the ability to thoroughly test new and existing installations. These standard cases can be augmented with specific "what if" cases to give the field person the ability to extensively troubleshoot schemes when problems arise. Simulation data files can be (and are) e-mailed back to the TVA field personnel within minutes of receiving the request for the files. This flexibility and speed in generating the data files greatly aids in quickly troubleshooting equipment to get critical transmission lines back in service as soon as possible. In addition to dynamic end-end testing the field personnel can run single-ended tests to verify actual relay settings applied to the relays enable the relays to perform as desired.

Appendix A Sample Listing of Doble State Simulation Data File

Note: The actual data files do not include line numbers. They have been included here so each line of code could be subsequently described.

```
 1. VER;0003
 2. Created with CUPL macro in CAPE (by RWP) on Feb 16 1998 11:00:49.
 3. 1
 4. A-G  0.10 pu from 356 to 357.
 5. Y
 6. 0
 7. 0
 8. 0
 9.
10. V
11. V
12. 60.000000
13. 7
14. 3
15. VA
16. 1
17. VB
18. 1
19. VC
20. 1
21. I1
22. 1
23. I2
24. 1
25. I3
26. 1
27. H1
28. 1
29. Pre-Fault
30. TIME
31. 69.30000
32. 0.0000
33. 69.30000
34. 240.0000
35. 69.30000
36. 120.0000
37. 0.25000
38. -75.0000
39. 0.25000
40. 45.0000
41. 0.25000
42. 165.0000
```

43. 0.00000
44. 0.00000
45. 300
46. 60.000000
47. 60.000000
48. Fault
49. TIME
50. 16.5302
51. -2.3658
52. 77.5789
53. -132.00
54. 80.7943
55. 129.859
56. 21.0354
57. -79.979
58. .32900
59. -97.143
60. .31143
61. -90.314
62. 21.6567
63. -80.384
64. 10
65. 60.000000
66. 60.000000
67. Post-Fault
68. Time
69. 69.30000
70. 0.0000
71. 69.30000
72. -120.0000
73. 69.30000
74. 120.0000
75. 0.000000
76. 0.000000
77. 0.000000
78. 0.000000
79. 0.000000
80. 0.000000
81. 0.00000
82. 0.00000
83. 200
84. 60.000000
85. 60.000000
86. 1
87. VA
88. C
89. O

```
 90. P
 91. 06/07/67 01:01:01
 92. MS
 93. 1
 94. I1
 95. C
 96. O
 97. T
 98. 06/07/67 01:01:01
 99. MS
100.     1
101. H1
102. C
103. C
104. T
105. 0.000000
106. MS
107. **XX**F
```

The Doble Protest software expects an SS1 file extension, e.g. "356.SS1"

Explanation of each line:

```
 1. Version number of the Doble Protest Software.
 2. Text information descriptive of the entire data file.
 3. Number of individual simulations in the data file.
 4. Text description of the current simulation.
 5. Simulation timing (System or Source)
 6. DC Voltage (0, 48, 125, 250)
 7. Sense Delay
 8. Sense Duration
 9. No Alternate Freq (Src 1 of Master)
10. Variable Frequency on Src 1 of Master
11. Variable Frequency on System
12. Base Frequency
13. Number of sources (voltage + current)
14. Number of states (prefault, fault, postfault etc.)
15. Name of first source.
16. Source frequency (1=fundamental, 2=second harmonic etc.*)
17. Name of second source.
18. Source frequency (1=fundamental, 3=third harmonic etc.*)
19. Name of third source.
20. Source frequency (same as *)
21. Name of fourth source.
22. Source frequency (same as *)
23. Name of fifth source.
```

24. Source frequency (same as *)
25. Name of sixth source.
26. Source frequency (same as *)
27. Name of seventh source.
28. Source frequency (same as *)
29. Name of first state (Pre-fault)
30. Timing information (Time, Delay, Go At, PS UT)
31. Magnitude of first source.
32. Phase angle of first source (± 0-360° **)
33. Magnitude of second source.
34. Phase angle of second source (same as **)
35. Magnitude of third source.
36. Phase angle of third source (same as **)
37. Magnitude of fourth source.
38. Phase angle of fourth source (same as **)
39. Magnitude of fifth source.
40. Phase angle of fifth source (same as **)
41. Magnitude of sixth source.
42. Phase angle of sixth source (same as **)
43. Magnitude of seventh source.
44. Duration of first state (milliseconds)
45. Base Frequency of Src 1 of Master - S1
46. Base Frequency of System in State 1
47. Name of second state (Fault)
48. (similar to items 29-46 above)
49. (similar to items 29-46 above)
50. (similar to items 29-46 above)
51. (similar to items 29-46 above)
52. (similar to items 29-46 above)
53. (similar to items 29-46 above)
54. (similar to items 29-46 above)
55. (similar to items 29-46 above)
56. (similar to items 29-46 above)
57. (similar to items 29-46 above)
58. (similar to items 29-46 above)
59. (similar to items 29-46 above)
60. (similar to items 29-46 above)
61. (similar to items 29-46 above)
62. (similar to items 29-46 above)
63. (similar to items 29-46 above)
64. (similar to items 29-46 above)
65. (similar to items 29-46 above)
66. Name of second state (Post-Fault)
67. (similar to items 29-46 above)
68. (similar to items 29-46 above)
69. (similar to items 29-46 above)
70. (similar to items 29-46 above)

71. (similar to items 29-46 above)
72. (similar to items 29-46 above)
73. (similar to items 29-46 above)
74. (similar to items 29-46 above)
75. (similar to items 29-46 above)
76. (similar to items 29-46 above)
77. (similar to items 29-46 above)
78. (similar to items 29-46 above)
79. (similar to items 29-46 above)
80. (similar to items 29-46 above)
81. (similar to items 29-46 above)
82. (similar to items 29-46 above)
83. (similar to items 29-46 above)
84. (similar to items 29-46 above)
85. Time 2 is not used
86. Source sync
87. Contacts
88. Open to Close
89. Timer Mode
90. Date stamp of file creation
91. Screen Display in MS
92. (similar to item 85-91 above)
93. (similar to item 85-91 above)
94. (similar to item 85-91 above)
95. (similar to item 85-91 above)
96. (similar to item 85-91 above)
97. (similar to item 85-91 above)
98. (similar to item 85-91 above)
99. (similar to item 85-91 above)
100. (similar to item 85-91 above)
101. (similar to item 85-91 above)
102. (similar to item 85-91 above)
103. (similar to item 85-91 above)
104. (similar to item 85-91 above)
105. (similar to item 85-91 above)
106. End of file marker (**XX**F)

Appendix B Excerpts from the CUPL Code.

```
% Comment lines begin with the '%' symbol.

% This excerpt prompts the user for fault location information and CT
% ratio.

    message 'Enter to_bus of starting line segment (pick from list):  '
    read(0) second_bus

    message 'Enter circuit number of starting line segment (1):  '
    read(1) startckt

    message 'Enter ct ratio this branch (default 240/1):'
    read(240) ct_A
    save ct_A as ct_save

% Conversion from perunit to relay amps.

    save base_mva*1000/1.7321/base_kv_A/ct_A AS ct_A

% This code creates a default file name for output.  The 'strcat'
% command does a string concatenation, the ntoa converts a number
% to an ascii string.  If the users doesn't elect to enter a name
% the program uses this default file name.

    message 'Enter remote bus number B (other end of total line):  '
    read(0) BUS_B

    save strcat('v:\',ntoa(BUS_B),'.SS1') as B_name

    message 'Enter name (in single quotes with path and extension) for
             B-end data file (defaults to bus #):  '
    read(B_name) B_name


% This block of code saves the fault quantities in variables for output.
% It is called after each fault is applied.
% The $1, $2, etc. are variables passed to the 'function' when it is
% called.  e.g. 'save_fault_quantities 345 231 1 22'

ma(save_fault_quantities,

        save 0 as IRes

        save IA $1 $2 $3 IA_1
          save (IRes + IA_1) as IRes
          save IA_1*$4 as IA_1
            IF (ABS(IA_1) < 0.1) THEN save 0.0000 as IA_1 ENDIF

        save IB $1 $2 $3 IB_1
          save (IRes + IB_1) as IRes
          save IB_1*$4 as IB_1
            IF (ABS(IB_1) < 0.1) THEN save 0.0000 as IB_1 ENDIF

        save IC $1 $2 $3 IC_1
```

```
        save (IRes + IC_1) as IRes
        save IC_1*$4 as IC_1
          IF (ABS(IC_1) < 0.1) THEN save 0.0000 as IC_1 ENDIF

        save IRes*$4 as IRes
          save ABS(IRes) as IResmag
          save ARG(IRes) as IResang

          IF (ABS(IRes) < 0.01) THEN
             save 0.0000 as IResmag
             save 0.0000 as IResang
             ENDIF

          IF (ABS(IRes) > RES_LEVEL) THEN save RES_LEVEL as IResmag
       ENDIF


      save VA $1 VA_1
        save VA_1*69.3 as VA_1
          IF (ABS(VA_1) < 0.01) THEN save 0.0000 as VA_1 ENDIF

      save VB $1 VB_1
        save VB_1*69.3 as VB_1
          IF (ABS(VB_1) < 0.01) THEN save 0.0000 as VB_1 ENDIF

      save VC $1 VC_1
        save VC_1*69.3 as VC_1
          IF (ABS(VC_1) < 0.01) THEN save 0.0000 as VC_1 ENDIF
)

% This code sample writes fixed quantities (same for all cases) in the
% text file.

ma(insert_prefault,

    dis 'Y'          % Y for System - O for Source
    dis '0'          % Battery Voltage
    dis '0'          % Sense Delay
    dis '0'          % Sense duration
    dis ' '          % No Alternate Freq (Src 1 of Master)
    dis 'V'          % Variable Frequency on Src 1 of Master
    dis 'V'          % Variable Frequency on System
    dis '60.000000'  % Base Frequency
    dis '7'          % Six Sources are defined
    dis '3'          % Three States are defined
    dis 'VA'         % Source 1 Source Name
    dis '1'          % Source 1 Harmonic
    dis 'VB'         % Source 2 Source Name

...

    dis '1'          % Source 7 Harmonic
    dis 'Pre-Fault'  % State 1 Name
    dis 'TIME'       % State 1 Type
    dis '69.30000'   % Source 1 Amplitude during State 1
    dis '0.0000'     % Source 1 Phase during State 1
    dis '69.30000'   % Source 2 Amplitude during State 1
```

```
    dis '240.0000' % Source 2 Phase during State 1
    dis '69.30000'  % Source 3 Amplitude during State 1
    dis '120.0000'  % Source 3 Phase during State 1
    dis '0.25000'   % Source 4 Amplitude during State 1
    dis '-75.0000'  % Source 4 Phase during State 1
    dis '0.25000'   % Source 5 Amplitude during State 1
    dis '45.0000'   % Source 5 Phase during State 1
    dis '0.25000'   % Source 6 Amplitude during State 1
    dis '165.0000'  % Source 6 Phase during State 1
    dis '0.00000'   % Source 7 Amplitude during State 1
    dis '0.00000'   % Source 7 Phase during State 1
    dis '300'       % Duration of State 1
    dis '60.000000' % Base Freq of Src 1 of the Master - S1
    dis '60.000000' % Base Frequency of System in State 1
    dis 'Fault'     % State 2 Name
    dis 'TIME'      % State 2 Type
)

% This writes out the results of one fault.
ma(write_fault_data,
    dis ABS(VA_1) % Source 1 Amplitude during State 1
    dis ARG(VA_1) % Source 1 Phase during State 1
    dis ABS(VB_1) % Source 2 Amplitude during State 1
    dis ARG(VB_1) % Source 2 Phase during State 1
    dis ABS(VC_1) % Source 3 Amplitude during State 1
    dis ARG(VC_1) % Source 3 Phase during State 1
    dis ABS(IA_1) % Source 4 Amplitude during State 1
    dis ARG(IA_1) % Source 4 Phase during State 1
    dis ABS(IB_1) % Source 5 Amplitude during State 1
    dis ARG(IB_1) % Source 5 Phase during State 1
    dis ABS(IC_1) % Source 6 Amplitude during State 1
    dis ARG(IC_1) % Source 6 Phase during State 1
)


 % This code applies a B-phase to ground fault on the temporary
 % bus.  It calls the save_fault_quantities function and saves
 % the fault quantities at the terminal of interest for this fault


 af b_g newbus1 x
 save_fault_quantities(first_bus,second_bus,startckt,ct_first_bus)
    dis 'B-G',first_fault_location,' pu from',first_bus,'
 to',remotebus,'.'
 insert_prefault
 write_fault_data
     dis '1.00000'       %Source 7 polarizing current magnitude default
     dis ARG(IB_1)  % polarizing takes on angle of faulted phase
     dis '10'        % Duration of State 2
 insert_postfault


 % This code function determines what particular line segment the
 % desired fault location is in.  That is, when the user specifies
 % for a fault to be located 75% between bus A and bus B, if there
 % are several different line segments and buses between A & B the
 % code will determine what segment and where on that segment to
```

```
% place the fault bus

ma(locate_fault,


clear_change


IF(tapped < 1.5) THEN


    save 0 as g_t

    save 0 as totalZ

    save 0 as g_z

    Save 0 as Zsum

    save false as g_fp


    dopath_lines ($1 $2 $3 $5,
       save TRUE as g_fp
       save  zkmp #k #m #c as g_z
       save bus_name #k as from_bname
       save bus_name #m as to_bname
       save g_t + g_z as g_t
       )

% figuring which segment fault_location percent is in.
     save g_t * $4 as reach
     save 0 as totalZ
     Save 0 as Zsum

     dopath_lines ($1 $2 $3 $5,
       save TRUE as g_fp
       save  zkmp #k #m #c as g_z
       save bus_name #k as from_bname
       save bus_name #m as to_bname
       save totalZ + g_z as totalZ

       IF (totalZ > reach) THEN
             save #k as mid_near_bus
             save #m as mid_far_bus
             save #c as fault_circuit_number
             save (reach-Zsum)/g_z as fault_pu_distance
             save ABS(real(fault_pu_distance)) as mid_pu_dist
             BREAK
       ELSE
            save Zsum + g_z as Zsum
       ENDIF
       )
```

```
% This final example of CUPL code writes the magnitude and angle
% of various phase-variable quantities after a fault

ma(write_fault_data,
    dis ABS(VA_1) % Source 1 Amplitude during State 2
    dis ARG(VA_1) % Source 1 Phase during State 2
    dis ABS(VB_1) % Source 2 Amplitude during State 2
    dis ARG(VB_1) % Source 2 Phase during State 2
    dis ABS(VC_1) % Source 3 Amplitude during State 2
    dis ARG(VC_1) % Source 3 Phase during State 2
    dis ABS(IA_1) % Source 4 Amplitude during State 2
    dis ARG(IA_1) % Source 4 Phase during State 2
    dis ABS(IB_1) % Source 5 Amplitude during State 2
    dis ARG(IB_1) % Source 5 Phase during State 2
    dis ABS(IC_1) % Source 6 Amplitude during State 2
    dis ARG(IC_1) % Source 6 Phase during State 2
)
return
```

Author

Russell W. Patterson: Mr. Patterson received a BS in Electrical Engineering in 1991 from Mississippi State University where he is currently completing thesis work to earn his MS in Electrical Engineering. Mr. Patterson is presently a System Protection Engineer for the Tennessee Valley Authority (TVA) in Chattanooga, Tennessee. Prior to his current position, Mr. Patterson worked as a Field Engineer for TVA in West Point, Mississippi. Mr. Patterson is a member of the IEEE Power Engineering Society. Mr. Patterson can be reached via email at rwpatterson@tva.gov (work) or rwpatter@bellsouth.net (home).