SOFTWARE FRAMEWORK FOR PROTECTION RELAY TESTING

By

Jason Caleb Johnston

A Thesis
Submitted to the Faculty of the
University of Tennessee at Chattanooga
in Partial Fulfillment of the Requirements
for the Degree of Master of Science
in Electrical Engineering

University of Tennessee at Chattanooga
Chattanooga, Tennessee

December 2011

ABSTRACT

A software application is presented in this thesis that was designed to provide the framework for comprehensive automated testing of power system protection relays. This application was implemented using a model protection system that was designed to simulate actual relaying operations performed in the electric power system. The implemented application provided automated testing functionality that performed phase instantaneous over-current protection and phase distance protection testing. The modularity of the implemented software application provides the framework for the automated testing of a relay's complete protection abilities. Automated testing using this application was performed by driving the outputs of an Omicron CMC 256 relay test set to provide input signals to a GE Multilin D90-Plus Distance Protection System; though the framework is applicable to a wide variety of protective relays.

ACKNOWLEDGEMENTS

The author desires to express his gratitude for the people who helped make this thesis possible. Sincere appreciation is expressed to Dr. Stephen Craven for helping to make submittal of this thesis possible under the time restraints required to complete the degree at hand in the desired time frame. Sincere gratitude is also expressed to Pat Caldwell for bestowing knowledge of the GE Multilin D90-Plus and ABB circuit breaker simulator.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS iv

LIST OF FIGURES vii

LIST OF ABREVIATIONS ix

LIST OF SYMBOLS x

I. INTRODUCTION 1

The Electric Power System 2
Protective Relaying 3
Protective Relaying Architecture 6
Quantifying Protective Relay Performance 9
Protective Relay Configuration and Testing 10

II. PRIOR ART 14

III. SOFTWARE FRAMEWORK IMPLEMENTATION 19

Protection Equipment 19
Protection System Model 22
Automated Protection System Testing 27
Phase Instantaneous Over-current Protection 33
Phase Distance Protection 34

IV. EXECUTION RESULTS 37

Phase Instantaneous Over-current Protection Testing Results 37
Phase Distance Protection Testing Results 40

V. CONCLUSION 44

REFERENCES CITED 48

APPENDIX

A. FORWARD DECLARATIONS AND STRUCTS 50

B. FUNCTIONALITY FOR INTERFACEING WITH OMICRON CMC 256 53

C. WRAPPER FUNCTIONS THAT CONTAIN EXECUTABLE COMMANDS 56

D. FUNCTIONS THAT CREATE AND EXECUTE LINKED LIST OF STATES 61

E. MAIN METHOD TO SET UP TESTS AND ISSUE COMMANDS TO BEGIN EXECUTION 67

VITA 73

LIST OF FIGURES

1 Fault waveform for A phase current 4

2 Single phase protection relay control circuit 7

3 Digital protective relaying system 9

4 Relay testing demonstration 12

5 Omicron state configuration 15

6 Omicron distance testing GUI 16

7 Protection testing model 23

8 GE Multilin D90-Plus Contacts 24

9 Schematic of simulated protection system 26

10 Code used to obtain control of CMC 256 28

11 Function used to issue commands to relay test set 29

12 Define and output an analog voltage quantity 30

13 Define and output an analog current quantity 30

14 Automated relay testing flowchart 32

15 Create linked list of states for Phase IOC testing 33

16 Execute phase IOC testing 34

17 Mho characteristics for phase distance protection 35

18 Phase instantaneous over-current protection testing 39

19 Default relay settings for mho characteristic circle 40

20 Impedance coordinates that define fault locations 42

21 Mho circle defined by relay operation 42

LIST OF ABBREVIATIONS

AC, Alternating current

DC, Direct current

CT, Current transformer

VT, Voltage transformer

ADC, Analog-to-digital converter

ROM, Read only memory

PROM, Programmable read only memory

ms, Millisecond

s, Second

RMS, Root mean squared

GUI, Graphical user interface

API, Application programming interface

IOC, Instantaneous over-current

LIST OF SYMBOLS

Hz	Hertz
Vdc	Volts direct current
Vrms	Volts root mean squared
V	Volts
A	Ampheres
Ω	Ohms

CHAPTER I

INTRODUCTION

The electric power system entails a vast infrastructure that provides the delivery of power to the end user. This system provides vital resources that have become accepted as necessities for the functioning of modern society. Due to the strong emphasis placed on the need for readily available power, outages in the electric power system are intolerable. To provide reliability and mitigation of the effects of failure, protection systems are integrated into the electric power system.

Protection systems provide methods for preventing power system failures and reducing the damage incurred in the event of a failure. However, protection mechanisms are only utilized in the event of a power system disturbance. Since the design of the power system stresses strong system reliability, it is rare that corrective action from these protection mechanisms is invoked. Furthermore, since these devices protect critical system assets, it is imperative that they operate correctly and in a timely manner when corrective action is required. To ensure reliable operation, routine testing is performed on protection system devices.

The bulk of the logic operations required to trigger a response in a protection system are performed by protection relays. Testing of these relaying devices is performed using relay test sets that simulate actual signal quantities that would be seen by a relay's inputs in a typical energized power system. However, these relay test sets do not come packaged with automated testing software, but rather with software components that are designed to provide the greatest ease of use to the end user, typically a power engineer or field technician. Thus, these software packages typically provide a graphical interface to the user that is used to define individual test cases. This method of relay testing may provide the simplest method of performing testing, but lacks the ability to perform comprehensive relay testing.

The Electric Power System

The electric power delivery system is an intricate arrangement of components that provides the functionality required for power generation, transmission, distribution, and operation. The components required for this functionality are as follows [1]:

- Generation
    - Generators
- Transmission
    - Media structures and power lines
    - Transformers
    - Protection – circuit breakers, relays, sensors
- Distribution
    - Media – structures, power lines, cabling
    - Distribution transformers
    - Protection – circuit breakers and relays
- Operation – monitoring and management

The capital investment in this system amounts to an enormous sum, as the generation alone involves large scale site operations dependent upon one or multiple power generators that require a substantial energy source, often nuclear or fossil fuels, to complete the power generation process. The transmission of power utilizes extensive spans of high voltage power lines supported by large steel structures. High voltage transformers are required for the transmissions of power from the generation source to local distributors, where high voltage transformers are also required to step down the voltage to a level that can be distributed to the end consumer. These components all require protection equipment to protect the high dollar investments that make up the electric power delivery system. Also, this complex system requires constant maintenance and monitoring that effectively comprises the work force in the power industry [1].

The primary operating conditions of the electric power system consist of normal operation, prevention of electrical failure, and mitigation of the effects of electrical failure. Normal operation infers the absence of equipment failure, personnel mistakes, or natural disaster. The design of the electric power system is dictated primarily by the need to meet normal operating requirements; which is comprised of supplying sufficient power to meet the existing load plus a certain additional amount of the anticipated future load. However, additional provisions must also be included in the design to minimize damage to equipment and time of service interruptions when failures occur in the system [2].

Protective Relaying

Disruptions in a power system can include numerous occurrences such as loss of power, voltage sags, and over voltages; these events can stem from consequences of natural events, physical accidents, equipment failure, and human error [3]. A type of electrical failure with the utmost severity in a power system is created by a short circuit; generally called a "fault." A short circuit is characterized by a sudden and significant increase in current, or a transient current; however fault currents depend on the impedance in the system between the fault location and the generation sources, as well as the pre-fault current flowing in the system and the voltage waveform's amplitude at the instance that the fault occurs [4]. System faults generally include significant changes in system quantities; some of these quantities include: current, voltage, phase angle, power factor, impedance, frequency, and temperature [3]. With the exception of temperature, all of these quantities can be derived from voltage and current measurements.

Furthermore, faults can occur concurrently on all three phases or individually as a phase to ground fault. An example of a phase to ground fault is shown in Figure 1. In this figure the A phase current is subject to a significant increase in amplitude while the other phase currents maintain a constant magnitude. All values of current in the system can be seen to subsequently return to zero when the fault is removed from the system. Moreover, small disturbances can occur regularly in a power system that are not severe enough to pursue corrective action. A possible way to distinguish power system disturbances is as follows [5]:

- Load disturbances:
  - Fluctuations in system quantities caused by varying loads
- Event disturbances:
  - Faults occurring on a transmission line attributed to equipment malfunction
    - Generally caused by lightening or other natural events
  - Residual disturbances from protective relay actions following severe system disruption
  - System fluctuations caused by generators going offline due to malfunction

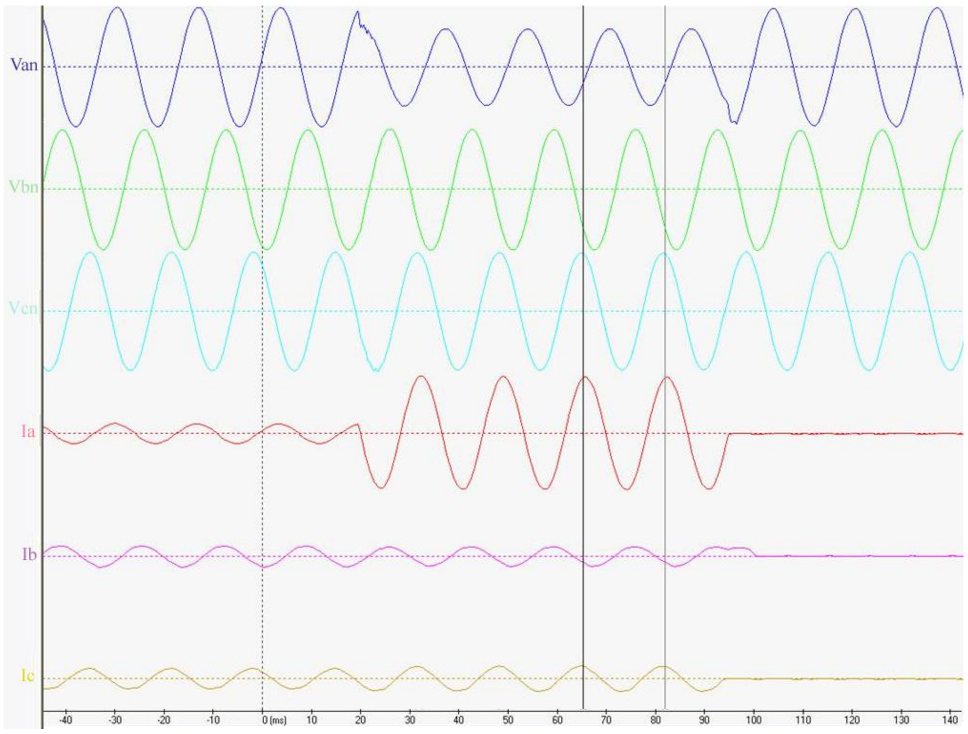


Figure 1 Fault waveform for A phase current

Continual operation of the electric power system in the presence of a fault presents functional hazards. Once a fault occurs in a system, the faulty element needs to be isolated from the rest of the system to minimize the effect of the event on the rest of the system. Means of promptly removing a faulty element from the system include: protective relaying, circuit breakers, and fuses. The aforementioned protective equipment is used concurrently to effectively minimize the effects of a fault [2]. Furthermore, the system cannot be returned to normal operation until the faulty section of the system is de-energized, allowing the abnormal operating condition to clear itself; although the use of alternate circuits, reserve generators, and automatic restoration equipment can mitigate the damage and loss of service caused by the faulty element [5].

Relay controlled circuit breakers are included in the system design in a manner that allows each major component, including but not limited to: generators, transformers, and transmission lines, to be completely disconnected from the system. Circuit breakers are able to withstand the greatest short-circuit current that may flow through them and then interrupt this current. Fuses are also over-current protection devices that allow the disconnecting of a circuit due to causing the pathway to be severed by the passage of over-current through the device. [3]

The primary function of protective relaying is to ensure the immediate removal of any element of a power system that experiences a short circuit, or begins to operate in any abnormal manner that could potentially damage or interfere with the operation of the rest of the system. This task is aided by circuit breakers that are able to effectively disconnect the faulty component when instructed to do so by the relaying equipment [2].

Additional functionality provided by protective relaying includes determining the distance from the protective relay to a fault that has occurred in the system; known as line distance protection [4]. This functionality utilizes phasor comparisons between voltage and current in the system and knowledge of the line impedance to indicate the distance to the fault. Furthermore, the ability to calculate the distance to a fault introduces qualitative measurements of the effectiveness of fault prevention and mitigation services in the system, as well as providing information about the effectiveness of the protective relaying itself [6].

Protective Relaying Architecture

The protective relay system measures system quantities from the power system, such as AC voltages and currents. These quantities have been stepped down by current transformers (CTs) and voltage transformers (VTs) from large magnitudes to smaller, more feasible values, to be read by electric monitoring equipment. Protection equipment in the electric power system continuously measures certain system quantities and determines if these values are within a specified threshold range that is computed and programmed into protection equipment by a protection engineer. If a measured quantity is outside of a specified threshold then a decision must be made by the protection equipment on how to properly handle the abnormal operating conditions. Typical action taken to protect the system generally consists of instructing circuit breakers to open to isolate a section of the system [5].

Protective relays have separate circuits for measuring AC quantities and for issuing protective actions to be performed by other protection equipment. The relay controls a circuit breaker by completing a DC circuit that consists of a battery connected to both the relay and trip coils in the circuit breaker, as shown in Figure 2. During normal operation the circuit breaker is closed which permits load currents to flow in the system and the relay contacts in the DC circuit are open preventing current flow into the breaker's trip circuit. When the relay detects a fault, it closes its contacts in the DC circuit allowing current to flow to the circuit breaker's trip coils.

When a relay permits current to flow to the breaker's trip coils, the circuit breaker opens, preventing load currents from flowing; thus de-energizing the connected power system and removing the fault from the system. When the circuit breaker operates, the current in the DC circuit is also interrupted, allowing the relay to remove itself from the circuit; thus making the relay ready to operate again for the next fault [7].

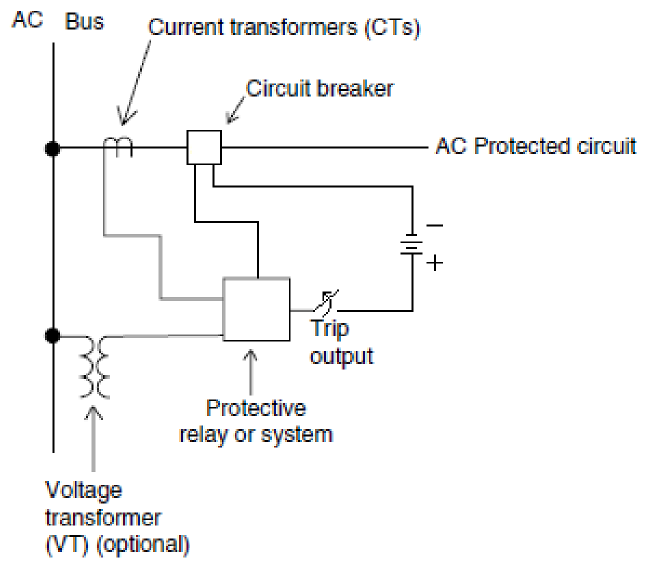

AC Bus
Current transformers (CTs)
Circuit breaker
AC Protected circuit
−
+
Trip output
Protective relay or system
Voltage transformer (VT) (optional)

Figure 2 Single phase protection relay control circuit [3]

Furthermore, since phase-to-ground faults can occur on any of the individual phases in a power system, a protective relay system, comprised of relaying equipment for monitoring each individual phase, is required to ensure total system protection. Additionally, a relay monitoring each individual phase allows for detection of phase to phase faults and other multiple phase fault occurrences.

The digital protective relaying system shown in Figure 3 demonstrates the manner in which input samples are obtained and processed by relays. This procedure begins with the measurement of analog quantities in the electric power system, performed through digital sampling. Analog quantities, such as voltages and currents, are measured from VTs and CTs to provide insulation from the large magnitudes seen in the power system [8]. The input signals used, which can vary depending on protection and control functions being implemented, are specified by protection engineers to be sampled by the relaying equipment. The sampled values are instantaneous quantities from the measured waveform that are converted to a digital form, generally an 8 or 16 bit word, by an analog-to-digital converter (ADC) [9]. It can be seen in Figure 3 that the current and voltage quantities are first processed by a digital filter and other types of signal conditioning techniques to remove noise that is not pertinent to the relay decision making [8]. The sampling clock, seen as input to the A/D (ADC) module in Figure 3, determines the rate that samples are collected. A sampling clock can have frequencies between 240 and 2000 Hertz [8], depending upon the time constraints imposed by the need to conduct operational decisions [9]. The digital data is then used by a microprocessor to produce phasor representations of the measured quantities, which subsequently are processed by algorithms stored in the relay's ROMs or PROM that perform the functionality required for making relaying decisions [9]. Additionally, protection contacts on the relay supply digital input to the microprocessor, including information such as the state of the attached circuit breaker or other pertinent logic operands, to be used by relaying algorithms to determine relay operation. Furthermore, the relay's protection contacts can provide digital outputs to other devices in the protection system such as breakers or other monitoring devices.

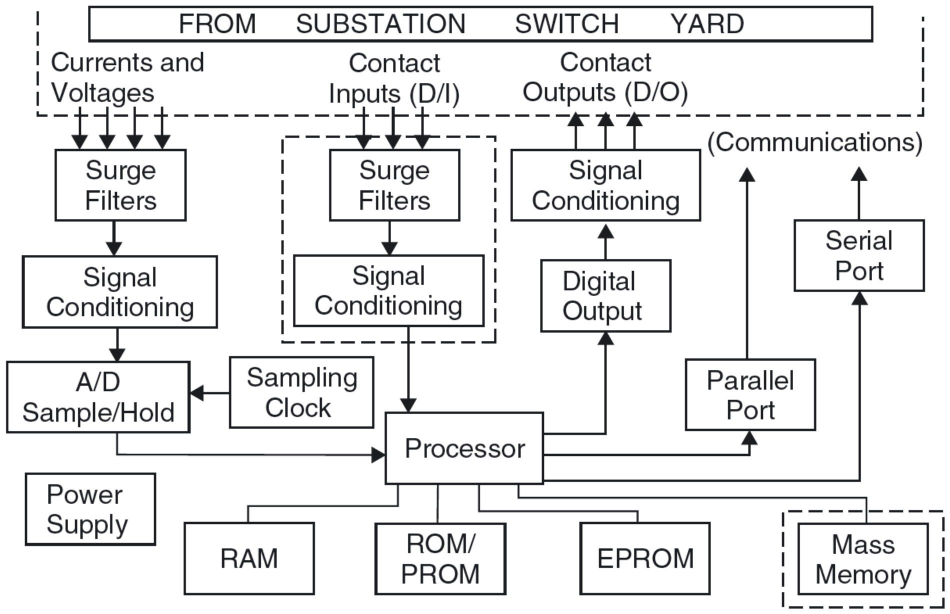


Figure 3 Digital protective relaying system [10]

Quantifying Protective Relaying Performance

The reliability of protective relaying equipment is dependent upon two aspects: dependability and security. According to the ANSI/IEEE standardized device function numbers (IEEE C 37.2), the dependability of a relay is, “the degree of certainty that a relay or relay system will operate correctly;” and the security of a relay, “relates to the degree of certainty that a relay or relay system will not operate incorrectly.” Therefore, the reliability of a relay is a function of its ability to operate correctly when required, and avoid unnecessary operation for small disturbances and varying load [3].

Dependability of a relay can be tested simply by ensuring that a relay will perform its duties as expected when the operating thresholds are exceeded. However, security is more difficult to determine due to the inherent fluctuations of a nearly infinite range of transient currents; thus it can be difficult to provide a complete and thorough testing of a device’s operation when subjected to these conditions [3].

Each protection relay is responsible for monitoring a specific region of the power system. Occasionally they may properly operate in response to a fault outside of this specified region, in which case it is providing backup protection to other protection systems outside of its primary operating area, known as a backup zone. Selectivity of a relay is the process of configuring and implementing a relay for its specified region in such a way that it operates as fast as possible for events inside its specified zone, and in addition providing delayed operation in its specified backup zone. The delay of operation for events in its backup zone is a configured relay setting that provides coordination with relays outside of its primary zone to enable a second line of defense should a relay in a nearby zone fail to clear a fault [3].

Speed of operation of a relay is highly important, however, instantaneous operation can increase the likelihood of operating incorrectly. The term instantaneous operation indicates that no delay period is specified in the relay's operation settings before performing corrective action. These operations are generally performed within 50 ms of detection of a fault, or less. Time coordinated relays operate typically on the order of 0.2 s to 1.5 s [3]. Thus, one should expect relay operation triggered by an event in its primary zone to occur within the first few cycles subsequent to measuring fault conditions; with a cycle length for a 60 Hz relay being defined as 16.67 ms.

Protective Relay Configuration and Testing

Implementations of protective relay systems require the following procedures to ensure proper operation [9]:

- Relay bench testing
- Programming of specified settings
- Verification of proper wiring
- Functional tests of the entire protection system

- Periodic relay testing once the device is in service

Functional tests performed on a relay before connecting to an energized load are performed to verify that the relay operates properly, that it maintains its role in the overall protection scheme of the systems design, and that quantitative measurements are correct. However, since relay operation is largely controlled by software algorithms, functional testing may merely consist of ensuring that a relay operates per design specifications. Furthermore, since software is applied uniformly across individual relay models, it is common practice to perform required testing only once on a specific model to verify that the software performs properly and the hardware operates in accordance with design specifications. Thus, successive implementations may refrain from bench testing assuming that manufacturer's testing is sufficient to properly verify hardware operation [9].

Protective relaying operation is largely dependent upon settings which are electronically programmed into the relaying device by protection engineers; therefore testing of the programmed settings is especially important to detect human error. Relay settings are required that identify phase rotation, CT and VT ratios, and to enable or disable various functionalities of the relay. In addition, electronic inputs to the relaying equipment will also define configuration and logic that determines relay behavior. These configuration settings are tested by applying sample system quantities and verifying that the relay operates in accordance with that of the configuration parameters. These sample quantities are supplied to the relay using a relay test set, like the Omicron CMC 256 pictured in Figure 4. This test set provides the capability of simulating secondary values seen by a relay monitoring an energized load. Various types of functionality testing can be performed on the relay pictured in Figure 4 using this relay test set, which can simulate a wide range of operating conditions that would be present in an energized power system.

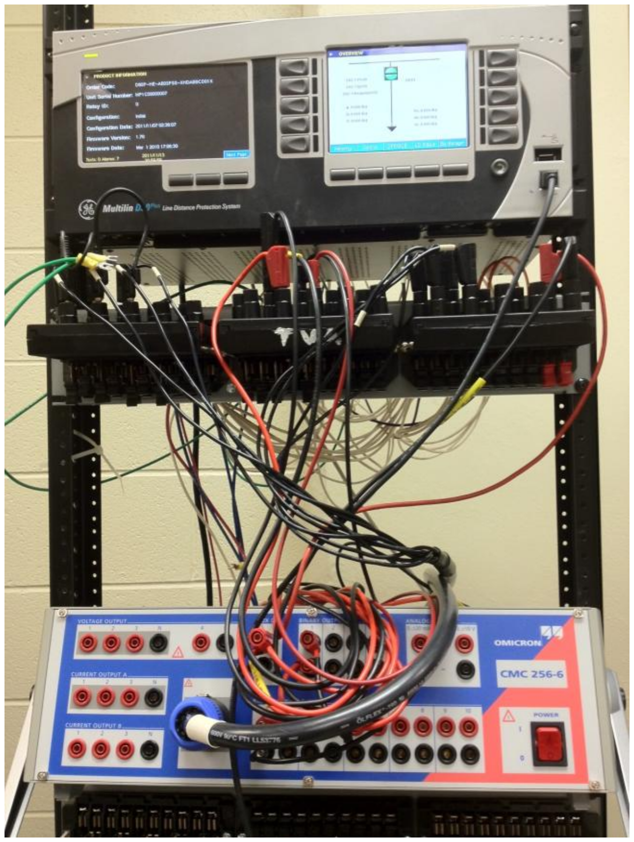


Figure 4 Relay testing demonstration

Once a relay is put into operation in a power system, periodic testing is employed to ensure that protection system reliability is maintained, and to reduce the in-service time of failed yet undetected equipment. Scheduled maintenance of protective relays ensures that any failed components are replaced or repaired and provides one method for computing the probability of mean time between failures. Furthermore, to optimize system availability it is necessary to determine the optimal duration between system inspections. This can be determined by computing the average availability time of the protection system, taking into account that the system is unavailable for protection during testing [11]. Moreover, various changes in system elements also incite the need for field testing. Firmware upgrades are occasionally supplied by the manufacturer to remedy software issues that may be discovered after deployment, or simply to enhance a device's operational features. When the firmware on a protection element is updated, all such devices on that system should be updated in conjunction with that element [9]. In addition, thorough testing should be performed on all updated devices to ensure that all software updates have been applied properly and that the new software does not contain any issues that could provoke incorrect device operation.

CHAPTER II

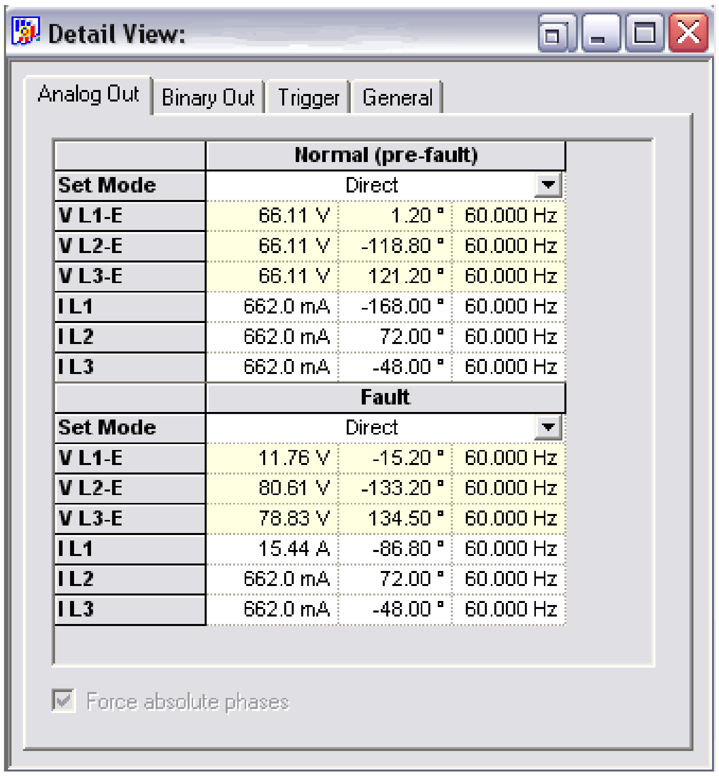
PRIOR ART

Various software packages and relay test sets exist for conducting relay testing to ensure correct relay settings and operation. Relay test sets are packaged with software interfaces that enable the user to perform this testing. Additionally, several third party software vendors provide software packages aimed at minimizing the time required to perform relay testing.

The Omicron CMC 256 relay test set provides the user with the ability to test relaying equipment through simulation of operating conditions that occur in the electric power system [12]. These operating conditions are individually configurable by the user by interfacing with a software GUI provided by Omicron with the CMC 256 test set. This software interface permits the configuration of individual test cases that must be executed serially. These test cases consist of a collection of individually configured states that execute sequentially. Each state is defined by three phase voltage and current pairs that are produced by the CMC 256 for a specified interval of time. Furthermore, multiple test cases can be combined to create user generated test plans [12].

The ability to execute a sequence of states allows the user to simulate one or several over-current events that may occur in a power system. This testing is accomplished by first creating a pre-fault state to simulate a power system operating under normal conditions, which provides normal voltage and current values to a relay for a specified length of time. A screen shot of this state designed using Omicron's Test Universe [13] GUI is labeled "Normal (pre-fault)" in Figure

5. Next the user designs a fault condition that will potentially provoke action from the relay's protection outputs; this state is labeled "Fault" in Figure 5. In this "Fault" state it can be seen that the current on an individual phase increases to a significantly higher value while the current on the other two phases remains constant; thus modeling a phase to ground fault. This condition is applied for a specified length of time or until the relay attempts to open a circuit breaker, in which case a post fault condition configured by the user is supplied to the relay by the test set.

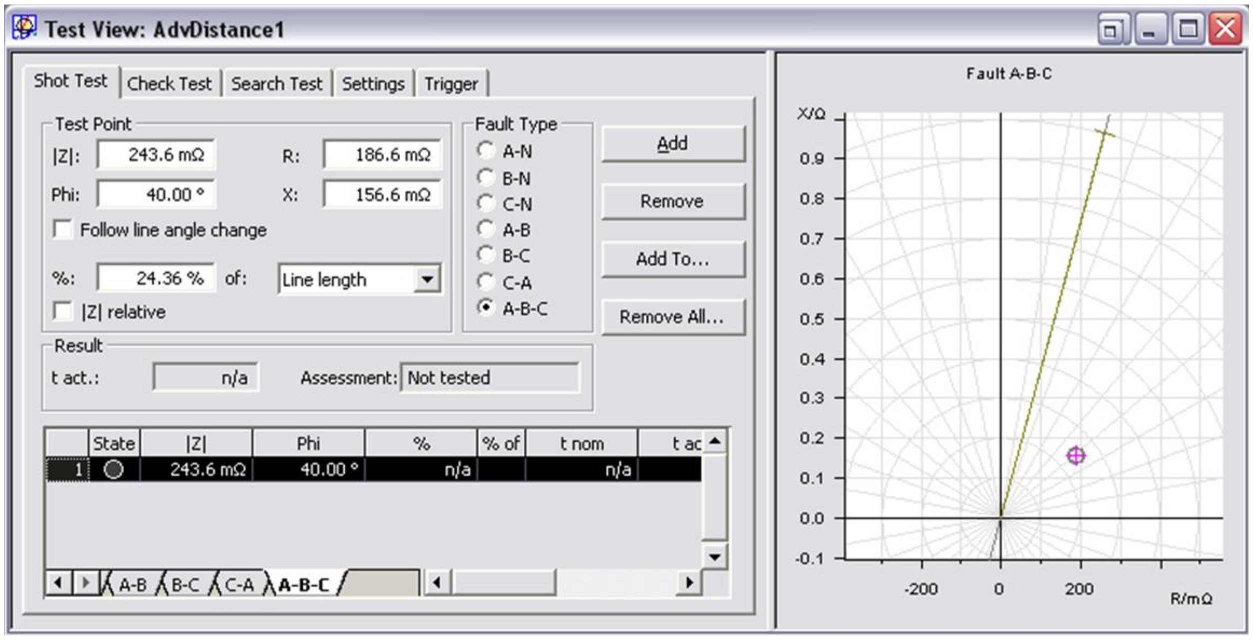


	Normal (pre-fault)		
Set Mode	Direct		
V L1-E	66.11 V	1.20 °	60.000 Hz
V L2-E	66.11 V	-118.80 °	60.000 Hz
V L3-E	66.11 V	121.20 °	60.000 Hz
I L1	662.0 mA	-168.00 °	60.000 Hz
I L2	662.0 mA	72.00 °	60.000 Hz
I L3	662.0 mA	-48.00 °	60.000 Hz
	Fault		
Set Mode	Direct		
V L1-E	11.76 V	-15.20 °	60.000 Hz
V L2-E	80.61 V	-133.20 °	60.000 Hz
V L3-E	78.83 V	134.50 °	60.000 Hz
I L1	15.44 A	-86.80 °	60.000 Hz
I L2	662.0 mA	72.00 °	60.000 Hz
I L3	662.0 mA	-48.00 °	60.000 Hz

Figure 5 Omicron state configuration [14]

This testing functionality is capable of testing a single fault condition for each pre-fault and fault pair of defined states. However, since there are limitless possibilities of fault occurrences in a power system, there is no feasible way to test a multitude of these cases with any degree of granularity due to the cumbersome approach of creating these individually defined states using a GUI.

Distance protection testing schemes provided to the user through the Test Universe GUI are subject to a similar shortcoming. Phase distance protection settings are programmed into the relay by designating impedance characteristics that constitute abnormal functionality in the system within a specified distance from the relaying equipment. The software interface used to control the CMC 256 test set allows a user to pick one or several coordinates in a plot of resistance versus reactance, as shown by the point chosen in Figure 6, and outputs the corresponding analog quantities that would allow the relay to see these impedance values in the system once a "run test" command is issued after the desired coordinates are chosen. This cumbersome method of distance testing also prevents the possibility of performing a thorough analysis of the systems operation for a wide range of impedance quantities.


Test View: AdvDistance1

Test Point
|Z|: 243.6 mΩ   R: 186.6 mΩ
Phi: 40.00 °   X: 156.6 mΩ
☐ Follow line angle change
%: 24.36 %   of: Line length
☐ |Z| relative

Fault Type: A-N B-N C-N A-B B-C C-A [✓ A-B-C]

Result
t act.: n/a   Assessment: Not tested

	State	|Z|	Phi	%	% of	t nom	t ac
1	○	243.6 mΩ	40.00 °	n/a		n/a	

Figure 6   Omicron distance testing GUI [14]

Relay test sets produced by Doble Engineering provide software interfaces, similar to that provided by Omicron, packaged with their test sets. Doble's software GUI provides testing templates that are predefined test plans able to perform common testing procedures, such as analysis of over-current and impedance protection, with user defined values. Using this GUI, multiple testing templates can be chosen by the user to execute successively; with details of the testing plans and their respective results capable of being recorded into a database to establish a detailed test history [15].

There exist third party software vendors that provide automated relay testing applications. These applications provide easier means of performing the basic testing functionality that is supplied by relay test sets such as the CMC 256. Thus, testing procedures generated by these third party applications are implemented to determine reliability, accuracy limits, levels of deterioration of components, correctness in relay wiring connections, properly programmed settings, and if a fatal failure exists in the relay under test [16]. Third party vendor applications are designed to increase the speed of the initial set up required for testing, and to minimize user interaction. The goal accomplished by these software packages is testing reliability that reduces system downtime required to perform testing and reduced skills required for performing relay tests [16].

Enoserv, a software development firm for power applications, provides an automated relay testing application designed to decrease labor time required for relay testing; claiming that the power industry is struggling to balance preventative maintenance with other projects due to a limited supply of man-hours [17]. Thus, this software is only designed to perform relay tests that are typically performed through manual testing schemes created with a relay test set's GUI; however Enoserv provides software capable of interfacing with relay test sets produced by several manufacturers, including Omicron and Doble. The provided software is capable of performing the same relay testing schemes regardless of the test set manufacturer, with the ability to compile a database of the tests performed and their respective results. Furthermore, additional tests can be created by software users through a menu-driven method [17].

Siemens Corporation has also designed an automated relay testing platform. The goal of this software platform was to dynamically design relay tests dependent upon system specific behavior. This software leverages a software generation tool for controlling the test case generation process. The relay testing platform designed by Siemens Corporation utilized a relay test set that was managed by a control program capable of waveform production and event identification, as well as data collection and result comparison. This testing scheme was conducted on a motor over-current protection device [18], which performs similar operations to the aforementioned relaying devices. Test generation provided by this software platform effectively provoked operation of a relay for each of its protective functions, but did not perform a rigorous testing of the relays operation for every scenario that could occur.

Software interfaces provided by relay test set manufacturers are designed to promote usability and visually appealing interfaces. In addition, testing schemes provided through manufacturer's software may commonly be provided in the form of pre-structured templates, which do not provide the testing variability required to perform comprehensive analysis of each protection function provided by a relay. Furthermore, third party software applications can perform automation of the testing schemes provided in manufacturer's software; however, this software fails to implement a more complete and comprehensive testing procedure than that which is typically performed, it merely removes the time required to set up and perform typical testing procedures.

CHAPTER III

SOFTWARE FRAMEWORK IMPLEMENTATION

Digital protection relay functionality is provided through the interfacing of a microprocessor, an AC signal data acquisition system, memory components containing relaying algorithms, contact inputs to control the relay, and contact outputs to control other components in the protection system [19]. These components were all concurrently placed under testing to perform a thorough analysis of the relay's operational characteristics. Since the relay's operational characteristics are defined by the algorithms and settings programmed into the relay's memory modules, this functionality was carefully studied and automated testing procedures were designed to determine the accuracy and reliability of relaying operations. Testing procedures were performed through the manipulation of inputs into relaying system and the measurements of its respective outputs were compared to the expected results stated in manufacturer specifications.

Protection Equipment

An electric power system simulation was created that provided inputs to a model protection system, comparable to secondary quantities in a power system that would be obtained from current and voltage transformers. The modeled protection system consisted of a line distance protection relay that was connected to a circuit breaker simulator. The power system simulation was provided by a relay test apparatus, which was capable of simulating three phase secondary voltage and current quantities.

The protective relaying device under test was a General Electric Multilin D90-Plus Line Distance Protection System. This microprocessor based device exhibits a modular design with each individual module sharing a common bus. Thus, it is a scalable platform that consists of protection, control, automation, monitoring, communication, digital fault recorder, and security functionality. As its name suggests, the core feature of this protective relaying device pertains to sub-cycle distance functionality, which provides a high degree of sensitivity and selectivity for all types of faults. The distance function provides five zones of phase and ground distance protection, thus providing flexibility in monitoring primary zones of protection or coordinating with other relaying devices to provide backup protection for nearby protection zones. The D90-Plus also provides over-current protection functionality for phase, neutral, and ground faults, which can run in parallel with distance protection or can be programmed to provide primary protection [20]. The functionality of the D90-Plus is similar to the latest offerings from other major vendors, including ABB and SEL.

Analog input signals to the D90-Plus are sampled at 128 samples per power system cycle; which occur every 16.67 ms for the 60 Hz power system under test. Current and voltage signals are filtered before sampling is performed. DC components are removed from current signals through digital filtering along with removing noise from both the voltage and current signals [20].

The modeled protection system utilized an ABB circuit breaker simulator test accessory to complete the relay’s trip circuit. The circuit breaker simulator was powered by 110 Vdc in a similar manner to that of a station battery providing power to its protection equipment. This device accepted inputs from the D90-Plus and simulated the operations that would be performed when a breaker’s trip coils or close coils were energized. Outputs from the breaker simulator were used to notify the relay of the breaker’s operation and current state.

Operation of the breaker simulator can be controlled solely by relay outputs or by user interaction through buttons on the breaker which controlled its state. The ability to manually control the state of the breaker provides full control over the breaker’s simulation and allows for the modeling of situations such as those that require manual operation of a breaker for de-energizing a line or system. The breaker simulator also includes functionality that would prevent it from opening when instructed to do so by a relay or other protection logic to simulate the effects of a breaker failure.

The relay test apparatus used to simulate an electric power system consisted of an Omicron CMC 256 relay test set, capable of producing three phase voltage and current values up to 300 V and 25 A respectively. The CMC 256 provides the ability to individually configure AC current and voltage signals to model three phase quantities supplied to a relay from instrument transformers.   These signals are essentially analogous to that of secondary voltage and current values seen in a three phase power system. Furthermore, an auxiliary DC supply is also provided, capable of modeling a substation’s DC power source. Moreover, this relay test set provides functionality capable of measuring AC voltages up to 600 Vrms [21]. This capability allows for a secondary measurement source should the relay metering quantities presented in primary values not suffice or to explicitly read secondary voltages to verify that VT ratios have been properly programmed into the relay. The CMC 256 is comprised of ten of these analog inputs, which can also be used as binary inputs. This functionality allows for measuring when a relay’s trip or close circuit is energized to verify when a breaker should operate. Finally, four binary outputs are also supplied by the CMC 256 which are essentially switches that close to simulate a high binary output. Additionally, wetting of these outputs, achieved by applying a voltage to a binary output, can be performed when used in conjunction with the auxiliary DC voltage supply.

Protection System Model

A simulation of a power protection system was designed using the aforementioned equipment to provide a working model of the actual operation of a protection system when applied to a three phase power system. The relay test set was capable of providing voltage and current quantities to the relay that were analogous to quantities that would be provided to a relay from current and voltage transformers. The relay was connected to the test set in a fashion that allowed it to receive voltage and current quantities and make decisions about whether corrective action was needed based upon the values received at its inputs. The relay provided outputs as well as accepted inputs from a circuit breaker simulator that allowed the modeling of the operational signals that would be produced in the event of a fault and the response the relay would receive from protection equipment under these circumstances. The relay's connections to the circuit breaker simulator enabled the real time simulation of protective action that would be taken be taken in the event of a fault; in addition it provided means of testing the relay's ability to monitor a circuit breaker's state and operation.

A schematic of the complete system model is contained in Figure 7. This schematic demonstrates the connections provided from the CMC 256 test set as analog signal inputs to the D90-Plus relay, as well as the connections between the relay and the test set used for logic processing. The schematic also includes the wiring of the circuit breaker that enabled its operation as well as the communication of its operating state to the relay.

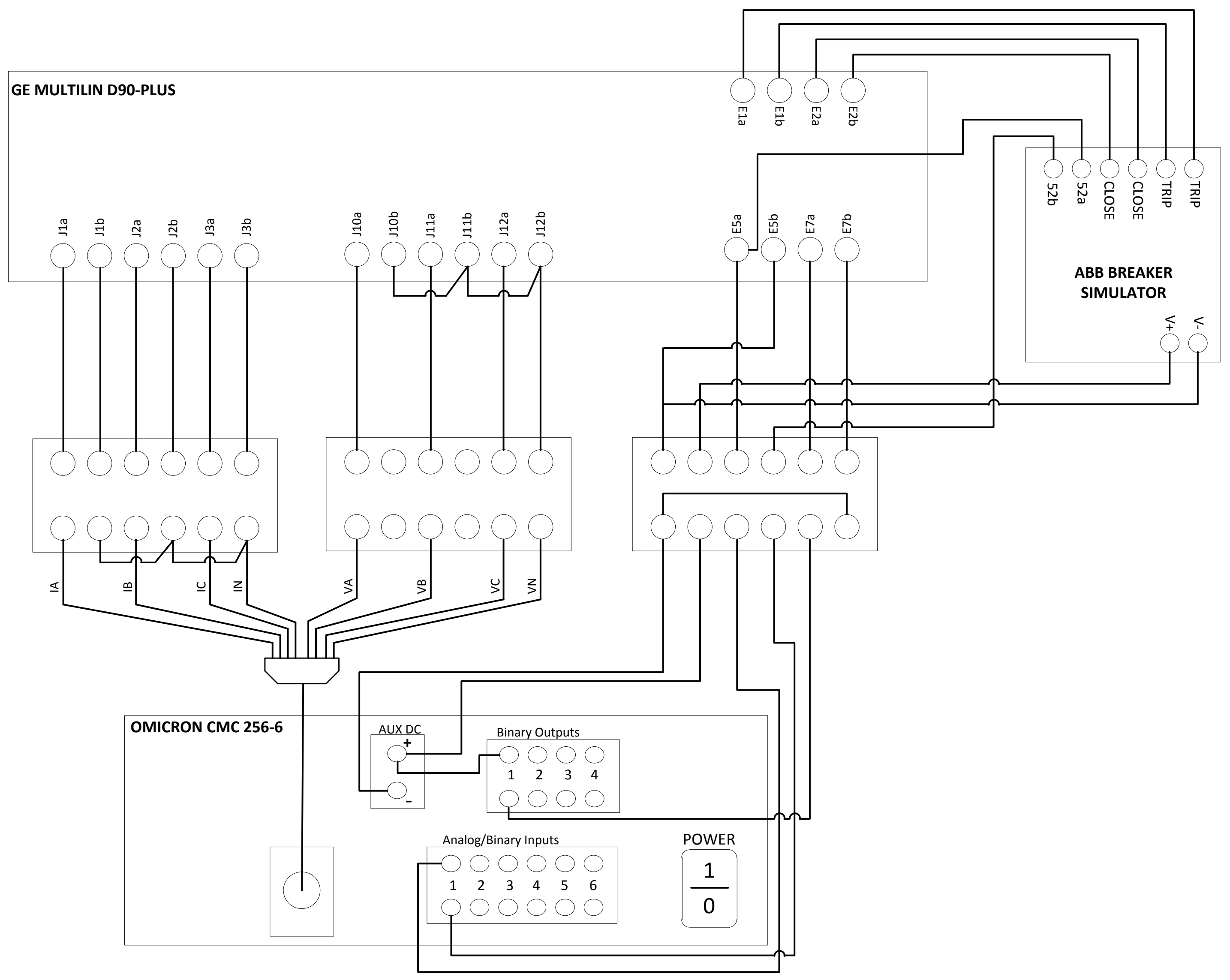


Figure 7 Protection testing model

It can be seen in Figure 7 that the analog signal quantities produced by the CMC 256 were output on one port from the test set and fanned out into eight connections to be provided to the D90-Plus. The wiring of these signals to the relay's input connections corresponds to the actual wiring connections that would be provided from instrument transformers as shown in Figure 8. It can be seen in Figure 8 that the current transformers in a three phase power system are directly connected to the inputs J1, J2, and J3 respectively, with each input to the relay also tied to the neutral line. Figure 8 also demonstrates the wiring connections provided from voltage transformers in a three phase power system to the J10, J11, and J12 contact inputs of the D90-Plus. In the same manner as the contact inputs for the current, each voltage contact input is also tied to the neutral line.

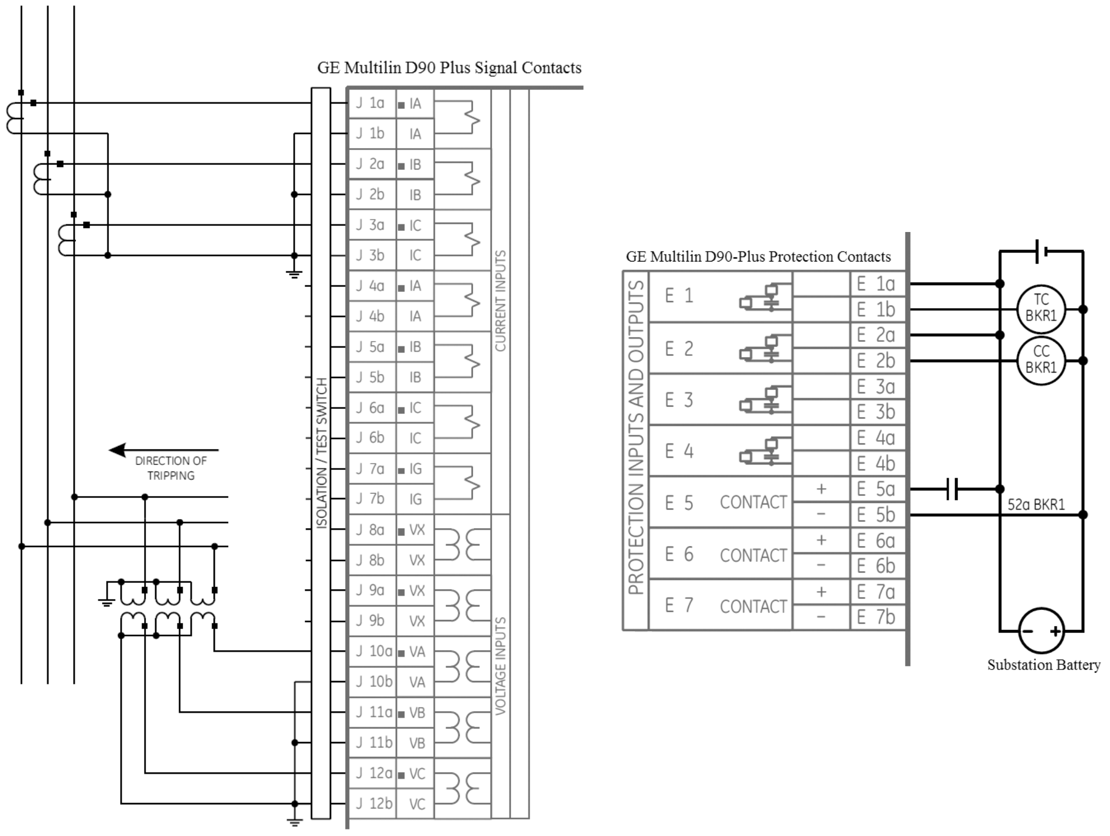


Figure 8 GE Multilin D90-Plus Contacts [19]

The CMC 256 was wired into the model protection system in a manner that allowed it to read the breaker's status. This wiring configuration, shown in Figure 7, consisted of connections between the test set's binary inputs and the relay's protection inputs that read the status of the breaker. The circuit used by the CMC 256 to read the breaker's status was completed by connecting the unit to the breaker's 52b output. However, this connection could have been bypassed by establishing a connection with the auxiliary DC source and the relay's 52b protection input; wiring would typically be connected in this manner in a protection system, as demonstrated in Figure 9. However, it was desirable to read the breaker's status directly to verify the readings obtained by the relay.

The D90-Plus provided configurable logic sources that were capable of controlling the relay's protection outputs. One of these logic sources was routed to the relay's E7 contact input and was employed by the CMC 256 to enable user control over the breaker's close circuit. This control allowed the CMC 256 to energize the relay's close coil by wetting a binary output, using the auxiliary DC source from the test set, and applying it to the relay's E7a protection contact input, as shown by the connections in Figure 7.

The protection system model was designed to utilize the exact contacts on the D90-Plus that would receive and transmit signals during normal operation in an energized power system. This model was designed to simulate to the system shown in Figure 9. In this system the D90-Plus is connected directly to instrument transformers from a three phase power system, providing secondary voltage and current values to the relay. These analog values are sampled by the D90-Plus and processed by relay logic. The circuit breakers in this system, shown on the transmission bus in Figure 9, are connected to the relay's protection contacts in the manner displayed in the breaker configuration, also contained in Figure 9. This breaker configuration consists of the trip circuit and close circuit connected to the relay's protection outputs, and the breaker's 52a state signaling output connected to the relay's protection input contact. In the system shown in Figure 9, a DC voltage of 110 V to 125 V is supplied to the circuit breaker from a power source that would be provided at the equipment's location.

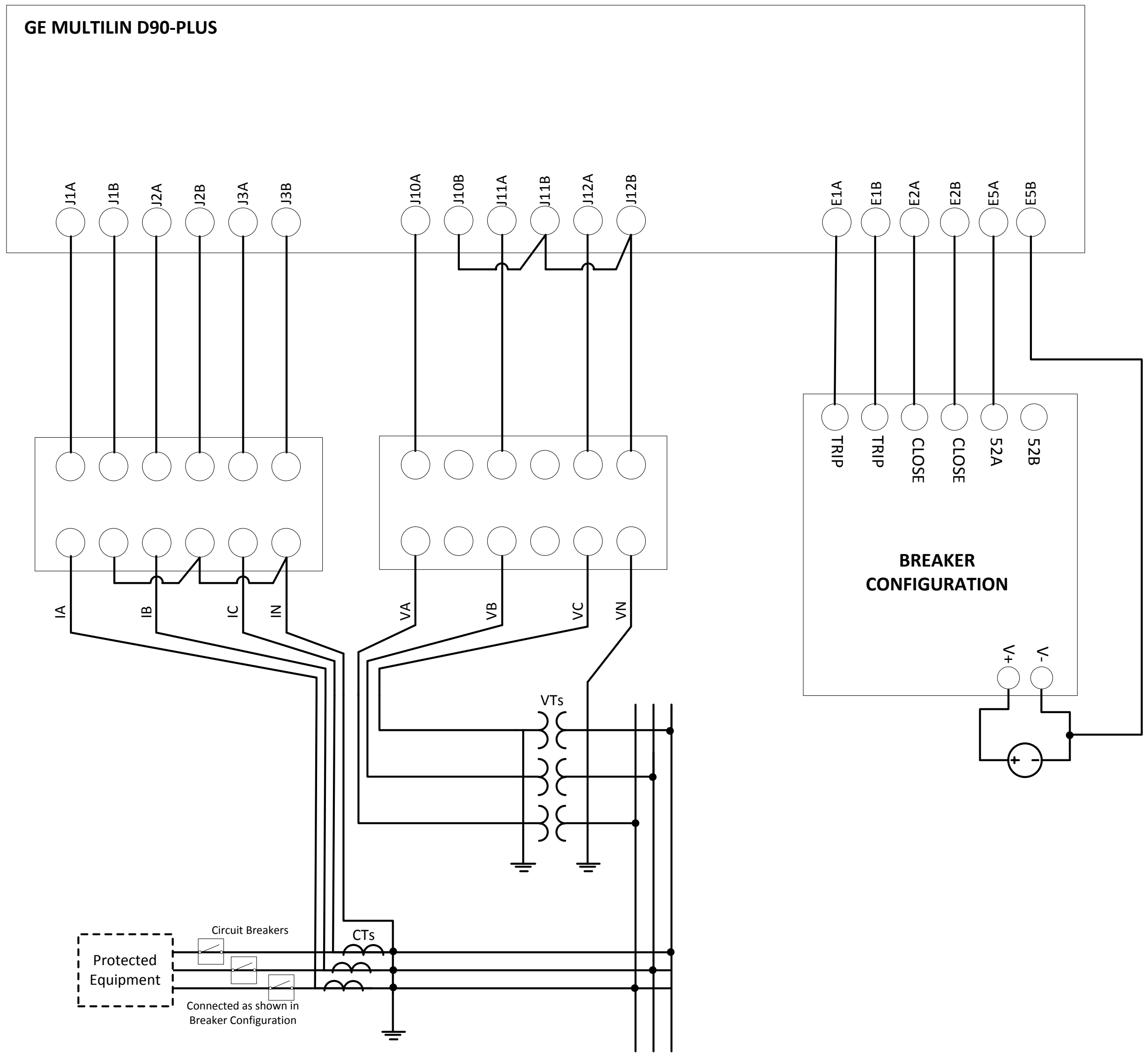


Figure 9 Schematic of simulated protection system

The protection system model contained in Figure 7 was able to successfully simulate the operation of the protection system shown in Figure 9. The CMC 256 provided the signal quantities that would be produced by the instrument transformers shown in Figure 9, and these values were sampled and processed using the same relay configuration detailed in Figure 9. The wiring connections between the relay and the circuit breaker were the same in the simulation model as they are in the modeled system with the addition of a second measurement across the 52a contact of the breaker to ensure that the relay determines the correct state of the breaker. The only additional functionality seen in the simulation model was the additional configuration of logic inputs to the relay that enabled a manual breaker close signal to be generated when the relay received a high binary signal on these inputs. This functionality was included in the protection system model to allow for automated tests to be continuously performed while utilizing the same breaker.

Automated Protection System Testing

Control of the CMC 256 through a GUI may suit a vast majority of user's needs for controlling the relay test set to produce protection system testing, however Omicron also provides a software library that can be used by several programming languages to control a CMC 256. This software library provides a list of commands capable of performing the vast majority of operations that can be carried out using the CMC's GUI. The availability of a software library for controlling a CMC 256 test set enables the use of additional logic capabilities to perform exhaustive relay testing and detailed operational analysis.

The software library provided by Omicron for interfacing with the CMC 256 test set, referred to as the CM Engine, was developed in Visual C++. Using the CM Engine, a C++ application was designed to perform extensive testing of the protection system model contained in Figure 7. The aforementioned software application interfaced with the CM Engine by creating functionality wrappers in C++ that encapsulated the Visual C++ methods required to obtain control of the CMC 256 test set and issue commands for execution. The CMEngineWapper.cpp file, located in Appendix A, contains the methods used to encapsulate this functionality.

An example of how the methods defined in CmEngineWrapper.cpp were used to obtain control of the CMC 256 is shown in Figure 10. The "CM" data structure, defined in wrapper.h and contained in Appendix A,was passed as an input parameter to methods that were used to obtain control of the CMC 256 for execution. This data structure contained a variable, "Engine," that held an instance of a CM Engine object and a "deviceID" variable to uniquely identify the desired CMC 256 to be placed under control. The "Engine" variable from the CM structure was first passed to the engine_object function to create an instance of a CM Engine object. Next, the variables from the CM structure were passed to a lock_CMC function, which assigns a unique identifying number to the "deviceID" variable if the CMC can be obtained for execution. If the CMC is not available then a return value of "-1" from this function will indicate that it is not accessible. Once the CMC is locked for execution, all commands that are executed using the CM Engine application programming interface (API) are issued using the "Exec" function defined in CmEngineWrapper.cpp, which is shown in Figure 11.

```
CM *cmc= (CM *) malloc (sizeof(CM));
//create Engine instance before passing Engine variable to lock the CMC
engine_object(&cmc->Engine);
//modifys the device ID used to reference the CMC, returns -1 if function fails
fail=lock_CMC(cmc->Engine, &cmc->deviceId);
//check to see if CMC was locked for sending commands before continuing
if(fail<0)
       exit(0);
```

Figure 10   Code used to obtain control of CMC 256

```
// Send command to CMC and read return value
LPTSTR Exec(LPCTSTR szCmd, CM *CMC)
{
	try {
		return (CMC->Engine)->Exec(CMC->deviceId, szCmd);
	} catch(_com_error ex) {
		_tprintf(_T("Failed to execute engine command:%s\n%s"),
			(LPTSTR)szCmd,(LPTSTR) ex.Description());
	}

	return _T("");
}
```

Figure 11	Function used to issue commands to relay test set

The "Exec" function shown in Figure 11 is used for every command issued to the CMC 256. This function utilizes an executable instance of the CM Engine object that is stored in the CM structure's "Engine" variable. This executable instance, referenced through a method also named "Exec," accepts parameters consisting of a unique identifier specifying a CMC and a formatted string that contains a command, provided in the CM Engine API, that defines an operation to be performed. The method prints an error message to the screen if the command is not contained in the CM Engine API; otherwise an acknowledgement of a completed command received from the CMC test set is used as a return value from the function.

Operations performed by the CMC 256 are issued through formatted commands provided by the CM Engine API, and occasionally require multiple commands to be issued to the test set to initialize operational parameters and define functional characteristics before the operations can be performed. In an effort to produce a modular software application, basic functionality that is commonly performed by the CMC 256 test set was encapsulated into software methods so that the functionality could be initiated from a single function statement. This encapsulated functionality can be found in the wrapperFunctions.cpp file located in Appendix C. An example of methods that define voltage and current outputs are shown in Figure 12 and 13 respectively.

In each method an API command must be formatted appropriately to configure an analog output followed by a subsequent formatted command to turn on the analog output. Each of these device commands are executed using the “Exec” function described in Figure 11. Other functionality included in the wrapperFunctions.cpp file consists of controlling binary logic outputs of the CMC test set as well as reading binary inputs to the test set that were used to determine the status of the circuit breaker simulator.

```
//sets an individual voltage output
void Vphase(int output, double angle, double amplitude, CM *CMC)
{
       double frequency=60;
       CString phase,turnOn;
       phase.Format(_T("out:ana:v(1:%d):a(%f);f(%f);p(%f)"),output,amplitude,
              frequency,angle);
       turnOn.Format(_T("out:ana:v(1:%d):on"),output);
       Exec(phase,CMC);
       Exec(turnOn,CMC);
}
```

Figure 12 Define and output an analog voltage quantity

```
//configure an individual phase current amplitude
void Iphase(int output,double angle, double amplitude, CM *CMC)
{
       double frequency=60;
       CString phase,turnOn;
       phase.Format(_T("out:ana:i(1:%d):a(%f);f(%f);p(%f)"),output,amplitude,
              frequency,angle);
       turnOn.Format(_T("out:ana:i(1:%d):on"),output);
       Exec(phase, CMC);
       Exec(turnOn, CMC);
}
```

Figure 13 Define and output an analog current quantity

The majority of the protection system testing functionality designed using the CM Engine library is contained in the faultStates.cpp file, which is located in Appendix D. The faultStates.cpp file utilizes the “state” data structure defined in the wrapper.h header file to create a linked list of states that are capable of imitating the states that are individually created using the

CMC 256 GUI interface. This data structure contains variables to hold all output quantities used for formatting CM Engine API commands as well as a structure pointer to allow the implementation of a linked list. This linked list of states is capable of growing dynamically to model a large variety of scenarios capable of occurring in the electric power system. Methods for defining these states included in the faultStates.cpp file provide a degree of variability in the amount of required input parameters used to configure states. Individual states can be configured with equal magnitudes for all three phases and equal separation between phase angles, or quantities for each phase can be individually configured. Each state is added to the end of the linked list when its quantities are configured. The variability in the control of state configuration allows for individual states to be configured dynamically with the capability of performing general to more specific testing.

The faultStates.cpp file contains a function called "exec_states" that executes the states contained in the linked list. This function executes each state in the linked list by outputting current and voltage quantities for a specified length of time or until the CMC 256 accepts a binary input acknowledging that the breaker simulator operated. This function retains the quantities provided to the relay that incited protective relaying action and this data is formatted and output to a file to tabulate the testing results. This function is capable of testing a relay's phase instantaneous over-current (IOC) protection scheme and distance protection scheme. The execution logic of the relay testing software application is detailed in software flowchart shown in Figure 14.

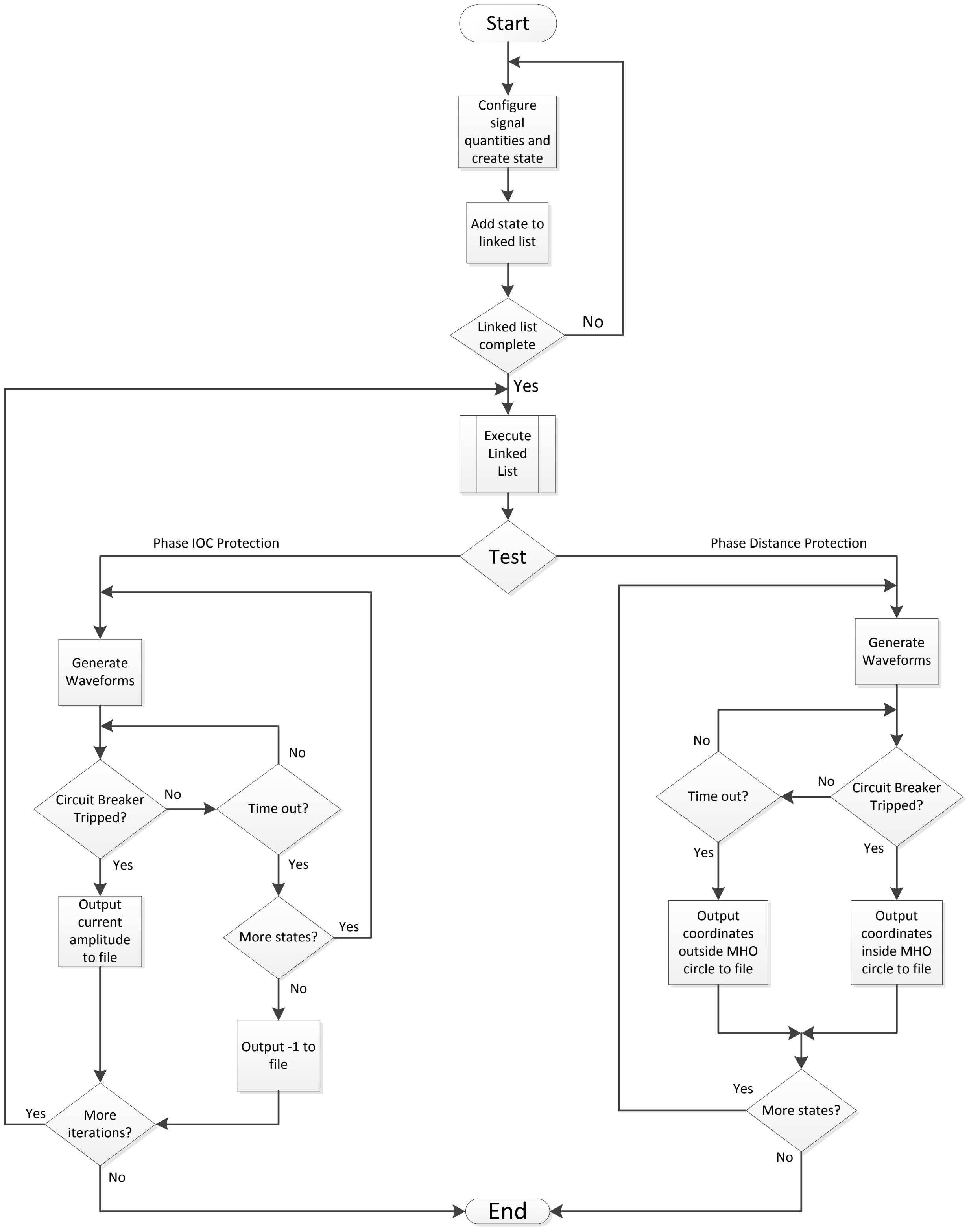


Figure 14 Automated relay testing flowchart

Phase Instantaneous Over-current Protection

The phase IOC protection functionality provided by the D90-Plus features a set of comparators that accept current phasors as inputs and determines if the signal quantities are outside the normal operating range, specified by programmed relay settings. This over-current protection scheme analyzes the current signals from all three phases and energizes the relay's trip circuit if the current magnitude on any of the phases exceeds a specified limit.

The logical design of a comprehensive testing of the D90-Plus' over-current protection scheme is detailed in Figure 14. This flowchart demonstrates the methodical approach to analyzing the reliability and precision of the D90-Plus' over-current protection features. The automated testing scheme features a dynamically allocated list of states that iteratively applies a current signal of increasing magnitude with a fine grained step size to the D90-Plus' analog signal inputs. The code used to create this linked list of states is shown in Figure 15. The list of states is then executed by looping over the "exec_states" command shown in Figure 16. When the states are executed, the current input quantities applied to the relay's signal inputs continue to increase in magnitude until the relay's trip coils are energized or until the theoretical value of trip current is surpassed. This test was performed a multitude of times to verify the reliability of the D90-Plus and to analyze the precision at which it operates.

```
//Phase IOC testing
       //set up states with 3 phase I & V to execute at speed of 100ms
       for(i=.9; i<1.1;i+=.001)
       {
              for(j=0;j<3;j++)
              {
                     I[j]=i;
                     V[j]=1.5;
              }
              numStates=addState(&IOCinit_State,V,I,vAngle,iAngle,100);
       }
```

Figure 15 Create linked list of states for Phase IOC testing

```
        for(j=0;j<1000;j++)
        {
               Istep=exec_states(IOCinit_State,0,cmc);
               fprintf(csvIOC, "%f\n", *Istep);  //records value of A phase current
since current is equal across all phases
               Sleep(300); //allow mechanical operation of Omicron's CMC to reset
        }
```

Figure 16 Execute Phase IOC testing

Relay settings that are configured by the end user define the operational characteristics of the device. These relay settings allow for variations in relay operation ranging from the level of signal amplitude that indicate faulty system operations to the type of event that will provoke corrective action from the relay. Due to the nature of variability in device operation depending upon user defined settings, it is desirable to verify these settings have been correctly programmed through complete and rigorous device testing.

The settings used for the testing of the D90-Plus primarily consisted of default settings programmed by the manufacturer. These settings must be manually configured and produce the only source of automation throttling in the testing design. Therefore, the majority of the D90-Plus' settings were left in their default state to produce automated testing concepts, however the developed software platform could perform detailed analysis regardless of any combination of settings programmed into the relay.

Phase Distance Protection

Phase distance protection is provided by the D90-Plus using phasor comparators that analyze the ratio of voltage at the relaying point to the measured current. Since the impedance of a transmission line is proportional to its length, the apparent impedance that is observed through this ratio is compared with information about the line impedance to determine the distance from the relay to an event occurrence [22]. Impedance characteristics in a system are modeled using a mho diagram which consists of a plot of the resistance versus reactance of a transmission line. Relay settings for phase distance protection are defined using mho characteristics to model the impedance of a system that is seen by the relay during the occurrence of a fault state.

The mho characteristic circle encompasses all values of measured transmission line impedance that are considered fault conditions; thus any measured impedance that falls inside the circle should initiate a breaker trip command. The mho characteristic circle is shown in Figure 17 in the forward direction orientation. The configurable relay settings that govern its distance protection characteristics include direction, RCA , and reach. The RCA setting specifies the characteristic angle of the distance protection, which is offset by 180° should the reverse direction setting be chosen. The reach of the mho circle is shown in Figure 17, which is entered in secondary ohms and configured at an offset angle specified by the RCA setting.

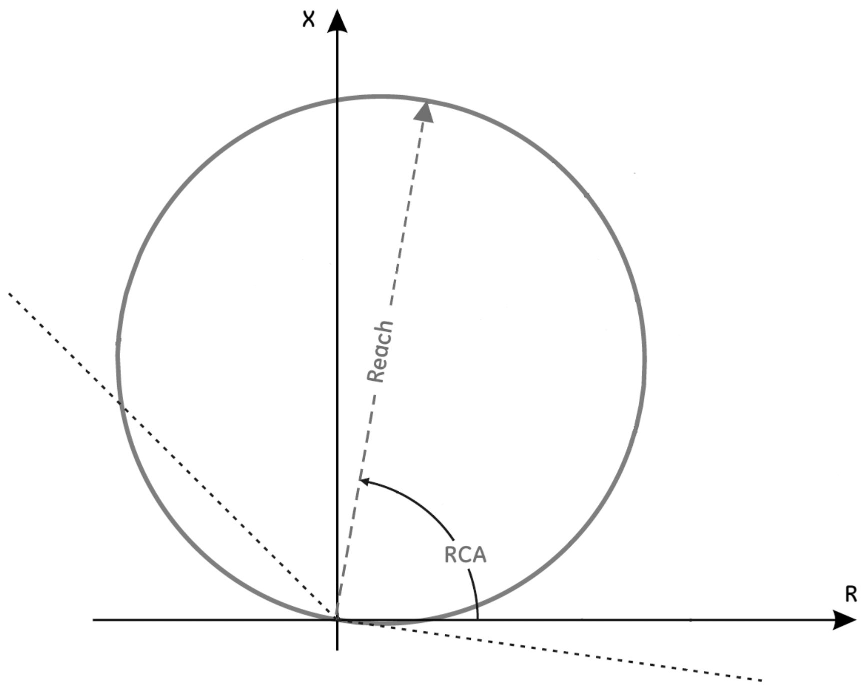


Figure 17 Mho characteristics for phase distance protection [20]

The logical steps executed to perform testing for the D90-Plus' distance protection functionality are detailed in Figure 14. Executable states were defined in a similar fashion to that performed for the testing of phase IOC. These states were then passed to the same function that executed the states in the over-current test, however with a different flag to notify the function to format execution results in a different fashion and output them to a separate file.

The execution of the defined states was also performed in a similar fashion to the over-current testing, however, if a breaker failed to trip for an individual state, the signal quantities provided in the state were flagged as quantities that did not provoke protective action from the relay and were formatted appropriately for output to file. The phasor signal quantities produced by the CMC 256 test set were converted to rectangular coordinates before being output to file for plotting purposes. Additionally, if a state produced signal quantities that energized the trip circuit then these phasors were also converted to rectangular coordinates and flagged as values that fall within the defined mho characteristics, and output to file.

CHAPTER IV

EXECUTION RESULTS

This chapter leverages the developed software framework for exhaustive over-current and line distance protection testing. The implemented protection testing schemes were performed using the software platform that also formatted their respective results in a desirable manner. The results produced by these implemented testing schemes are discussed in this chapter.

The ability to control the CMC 256 relay test set using commands issued by a user defined program allowed for the flexibility of designing test cases to rigorously test relay operation and the ability to perform verification of relay logic for an entire protection scheme. This flexibility was extended to the user through Omicron’s CM Engine software library, and this programming library was used to lay the framework for a relay testing application that could perform a comprehensive verification of a relay’s protection logic. The presently implemented testing procedures include a rigorous analysis of the D90-Plus’ over-current protection precision, and a comprehensive test of its distance protection logic for the relay’s default settings.

Phase Instantaneous Over-current Protection Testing Results

The precision of the D90-Plus’ over-current protection logic was tested using the model protection system in Figure 7 to simulate the relays protective operation in the event of a current magnitude applied to its inputs that exceeds the rated current amplitude. This simulation was performed by modeling a transmission system consisting of purely resistive transmission lines. This was accomplished by creating “in phase” voltage and current signals with each of the three phases evenly separated by 120°, at a frequency of 60 Hz. In phase quantities were chosen to model an ideal system where line reactance was negligible.

The primary settings that govern the relay's over-current protection scheme include: phase CT ratio, phase CT secondary, and pickup. The CT ratio and CT secondary settings are required to calculate the primary current values in the system given the secondary input values supplied to the relay. The pickup setting is used to specify the phase IOC pickup level in per unit values [19]. The per unit values specify the percentage of the CT primary value that will designate a trip state. The CT primary value is calculated using the following equation [20]:

$$\text{CT Primary} = \text{Phase CT Ratio} \times \text{Phase CT Secondary} \qquad (4.1)$$

The default settings of the D90-Plus designated a CT ratio of one and CT secondary of 5 A. Since the CT ratio was equal to one, the secondary current signals provided to the D90-Plus were of equal magnitude to that of the primary current signals. The phase over-current testing was performed using a pickup value of 0.2; therefore one could accept the theoretical trip value for phase over-current protection to be equal to 1 A for these relay configuration settings.

The test was executed by first iteratively defining states of equal voltage magnitude with a 1 mA increase in current amplitude between the values of ± 10 % of the theoretical fault current magnitude. For these testing purposes the theoretical fault current was designed to be 1 A and thus the states ranged in a current amplitude of 900 mA to 1.1 A. Signal quantities from each state were iteratively applied to the relay's analog inputs for 100 ms, or until the circuit breaker was opened by the relay's protection logic; at which point the fault current was cleared from the system and the breaker was closed. Individual signals were applied for 100ms since the operation time for over-current detection was rated at less than 16 ms, and ample execution time was desirable to determine the exact value at which the relay would recognize over-current.

When the relay recognized a fault and energized the trip circuit, the current amplitude at which the relay recognized a fault was recorded and the state execution was restarted; beginning again at an initial value of 900 mA. This process was repeated for 1000 trials to determine the precision at which the relay recognized current amplitude that exceeded its rated value for normal operating conditions. The value of trip current discovered in each test is plotted in Figure 18.

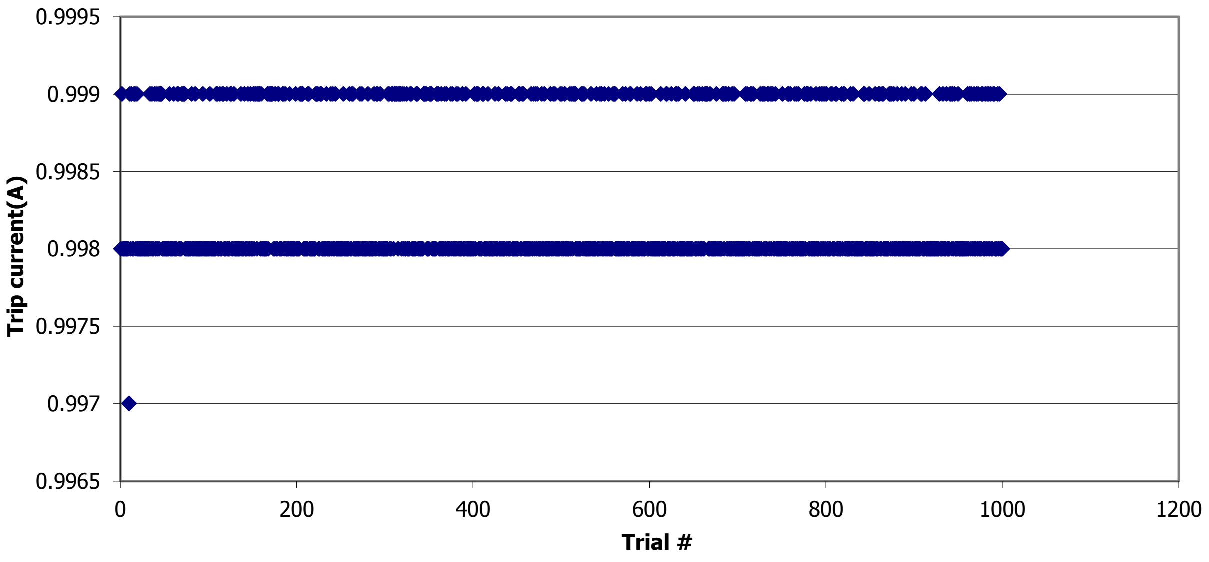


Figure 18 Phase instantaneous over-current protection testing

The maximum current amplitude allowed by the D90-Plus before energizing the breaker's trip coils was discovered to be 999 mA. Out of 1000 trials this amplitude was reached 291 times before the trip circuit was energized. The furthest from the rated trip current value that the relay issued protective action occurred in the 10th trial, where the trip circuit was energized when an amplitude of 997 mA was applied to the relay's signal inputs.

The D90-Plus exceeded the rated accuracy for over-current protection in all of the trials performed. The D90-Plus' over-current protection is specified to have an accuracy of ± 0.5 % of the signal reading or ± 1 % of the rated fault current; with the greater of the two being defined as the minimal required accuracy. Therefore, for the settings used in this test the minimal accuracy rating was required to be ± 10 mA, but the D90-Plus exceeded even the rated accuracy of ± 0.5% of the signal reading; which for this test was between the range of 4.985 and 4.995 mA. For 1000 trials the relay never allowed the fault current to exceed the rated limit and operated within at least 0.3 % of the rated value.

Phase Distance Protection Testing Results

The model protection system contained in Figure 7 was used to simulate event occurrences in a power system that would elicit relaying operations to provide phase distance protection. A comprehensive testing scheme was designed to determine the D90-Plus' reaction to input signals that produced impedance quantities that correlated with mho circle characteristics defined by default relay settings, as shown in Figure 19. These settings defined a forward direction mho circle with an RCA of 85°, at a reach of 1.7 Ω.

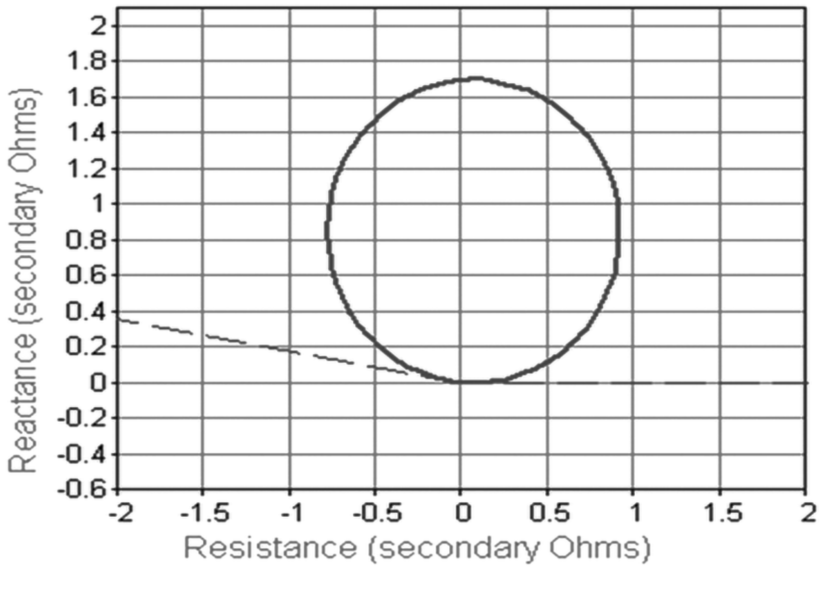


Figure 19 Default relay settings for mho characteristic circle

The comprehensive distance testing was performed by producing impedance phasor quantities to perform sweeps across polar curves throughout the mho circle shown in Figure 19. The granularity of iterations performed consisted of a radial distance varying by 0.05 Ω and polar angles increasing by 1°. Since the mho circle defined by default relay settings is forward directional and only exists in the first two quadrants, the functionality to define states in the third and fourth quadrant was removed from the software testing program through the use of program comments. The range of coordinates tested created a semi-circle in the first two quadrants with a radius of 2, centered about the origin.

The test states for phase direction testing were designed keeping current amplitude constant at a value of 1.324 A, and voltage phase angles constant at a value of 0°, 120°, and -120° for A, B, and C phase respectively. Voltage signals were varied in magnitude from 0 to 2.6V, and for each variation in magnitude 360 states were created that allowed current phase angles to be evenly separated by 120° between each phase and incremented by 1°. Signal quantities defined for each state were input to the relay for a duration of 100 ms, or until the relay's trip circuit was energized.

The comprehensive testing used to determine each point that is defined by mho circle characteristics was plotted in Figure 20. Each point in Figure 20 was determined by whether the relay issued a trip command to the breaker when the impedance value of that state was present in the system or if the breaker stayed closed for the 100 ms duration. The plot in Figure 20 closely resembles the mho characteristics defined in Figure 19 with the addition of a few states that tripped the breaker that are outside of the mho characteristic circle. These points are undetectable in Figure 20 due to the close concentration of plotted points; however Figure 21 displays solely the points that incited action from the relay due to the occurrence of a distance fault. There are also discrepancies between the observed mho characteristic circle and relay defined characteristics occurring near the origin due to voltage signal levels below threshold.

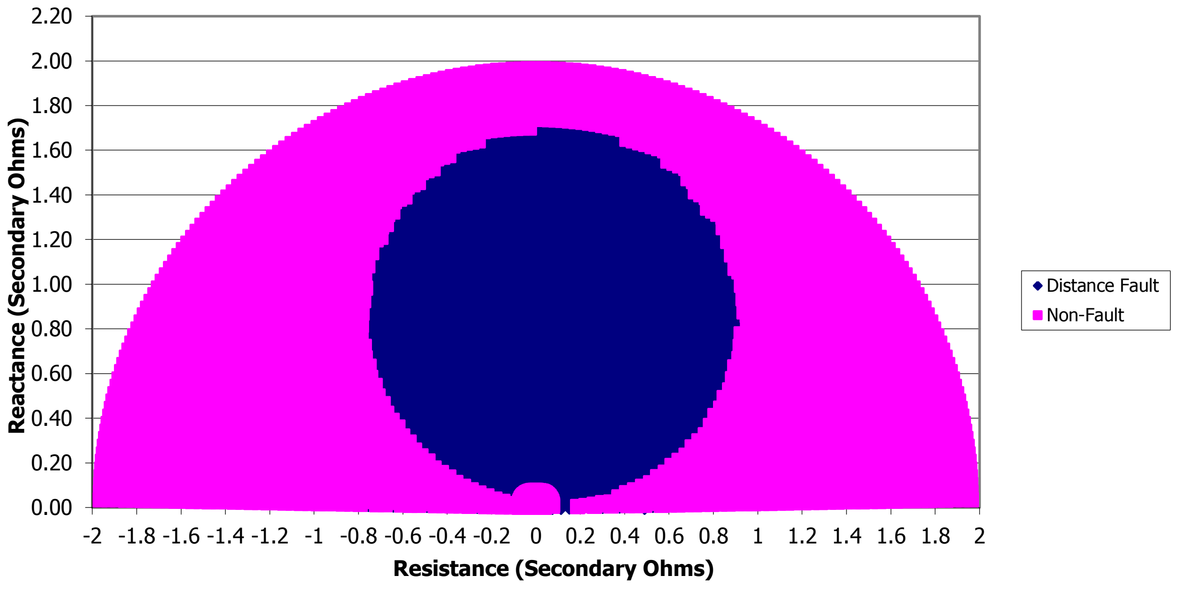


Figure 20 Impedance coordinates that define fault locations

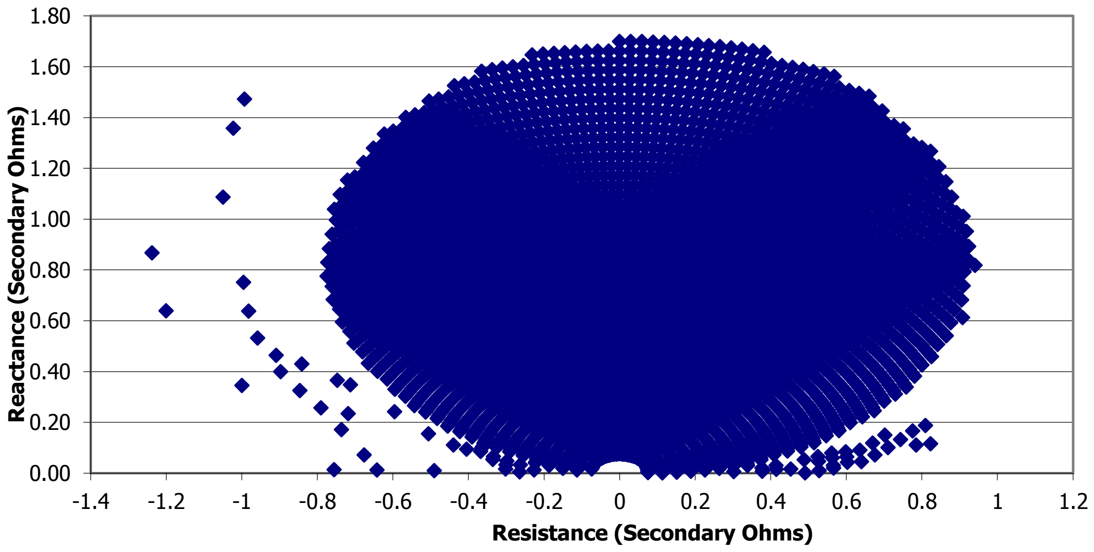


Figure 21 Mho circle defined by relay operation

The plotted mho characteristic circle is observed to have a reach of 1.7 Ω, and the orientation of the characteristic circle is tilted at an 85° angle. These observations correlate with the mho circle characteristics defined by the D90-Plus' default distance protection settings. Figure 20 clearly shows that around the origin there were several polar curves that did not exhibit fault characteristics in the system, although the relay's defined mho circle characteristics exist in the first two quadrants about the origin. These occurrences are due to voltage levels applied in the system approaching or falling below the minimum voltage threshold that can be read by the relay; the minimum voltage threshold is rated at a value of 100 mV.

Figure 21 displays several points that exist outside of the characteristic mho circle, however these states incited protective relaying action when these quantities were present in the system. These extraneous points are within a nearby region of the mho characteristic circle; however they are clearly outside of its defined region. These extraneous faults may indicate an issue with the relay settings; however, as this thesis focuses on the software framework, determining the source of these extraneous faults is beyond its scope.

The phase IOC protection provided by the D90-Plus operated correctly in every trial performed; exceeding the minimum accuracy level in each case. In these IOC tests the minimum accuracy level according to manufacturer specifications was rated to be ± 1 % of 1 A; however in each test the relay issued corrective action before 1 A was supplied to its contacts, within an accuracy level of 0.3 %. Furthermore, the D90-Plus exhibited well defined operational characteristics for its phase distance protection functionality; however, during a comprehensive test across its entire protection range, it operated outside of its specified fault region on approximately 0.5 % of the tests and did not operate inside its specified region at numerous points when voltage supplied to its inputs was below the rated input threshold. The lack of protective action performed inside the fault region was as expected when the input voltage threshold was not met; however, the source of the extraneous fault sources was not determined.

CHAPTER V

CONCLUSION

Protection relaying systems provide a significant source of safety and security in the electric power system. Protection systems introduce additional expenditure in a system's design and are only needed in the rare occurrence of system disturbances. However, the failure mitigation services provided by these protection systems are vital to a power system's fault recovery schemes.

Relay test sets, like the Omicron CMC 256 are commonly used to verify the reliability of protection relay functionality provided by an electric power protection system. These relay test sets generally include application software provided to the user in a GUI form to facilitate testing of protection relays. Relay test set software is fully capable of simulating fault occurrences in a power system; however, they are often unable to easily perform comprehensive relay testing due to the immense range of system disturbances that may occur. This issue was successfully addressed by a software platform that utilized commands provided by the CM Engine software library to control a CMC 256 relay test set. This software library bestowed a complete range of relay testing functionality to the programmer, thus enabling additional logic and automation features to be included in protection relay testing schemes.

Automation of protection relay testing was developed using C++ to perform comprehensive testing on prominent relaying features. The software platform provided modular functions that remove the overhead required for controlling and issuing commands to the CMC

256; thus enabling new test plans to be easily implemented. Test plans for phase IOC and phase distance protection were implemented that successfully performed thorough analysis of these protection features supplied by a GE Multilin D90-Plus protection relay.

A phase IOC protection testing scheme was created using a model protection system and simulating granularly increasing current amplitude until the relay's trip circuit was energized. This test plan was executed iteratively for 1000 trials to rigorously test the reliability and precision of the D90-Plus' over-current protection ability. For all 1000 trials the D90-Plus operated properly, performing with significantly greater precision than the rated accuracy limit in every trial.

The D90-Plus' phase distance protection functionality was also rigorously tested through an automated testing scheme. This test design consisted of analyzing a complete range of impedance phasors that would provoke protective action from the D90-Plus. This range was defined by the relay's default settings which provided characteristics for a mho circle diagram. Impedance phasors in this range were tested using polar coordinates with a radial distance granularity of 0.05 Ω and an angle granularity of 1^{o}. Testing sweeps were performed across polar curves up to a radius of 2 Ω which was well beyond the entire region defined by default mho characteristics. This test design discovered that the operation of the D90-Plus was well defined throughout the mho characteristic region, however there were several extraneous operations that occurred in nearby outlying regions. These extraneous operations were more prominent in the negative resistance quadrant and occurred across a 45^{o} range and throughout a large range of radial distance from the origin. In the positive resistance quadrant the extraneous operations occurred entirely within the first 20^{o} from the origin, across a smaller range of radial distance from the origin.

These rigorous relay testing schemes provided automated testing functionality that verified relay reliability and accuracy. These test plans consisted of a comprehensive list of scenarios that may exist in a power system and be seen by a relaying device. These testing schemes were designed by implementing a software application framework for performing automated comprehensive testing of all relaying functionality. The implemented application performed comprehensive testing of a relay's over-current and distance protection. However, the designed software application provides a degree of modularity that enables further additions of testing schemes utilizing the same underlying logic functionality.

These implemented testing procedures provide a more comprehensive testing scheme than that currently implemented by third party software vendors for relay test sets. The aforementioned testing procedures reduce the time incurred when performing relay testing as well as providing a simpler method for executing these tests; thus reducing skilled input required from testing personnel. Moreover, these implemented procedures perform a more thorough analysis of the relay's protection abilities than that tested by third party applications or by Omicron's supplied testing software.

Automation of relay testing scenarios was enabled through the use of a model protection system, and driven through outputs of a CMC 256 relay test set controlled through interfacing the designed software platform with the CM Engine software library provided by the test set's manufacturer. Automated testing procedures were only implemented that tested primary relaying functionality, however the software developed for these automated testing procedures produced the framework for a comprehensive protection relay testing design. This software provides modular functionality that can be used to build a relay test plan that thoroughly assesses a relay's entire protection ability.

The designed software platform provides the framework for future work. In addition to the implementation of supplementary relay testing schemes, future work could include the design of a database that stores successive testing information as well as test results so that changes in a protection system’s functional ability could be tracked. Furthermore, the incorporation of IEC 61850 into the software framework would coordinate relay testing schemes with the current substation communication standards as well as promote testing interoperability between relaying equipment from different vendors.

REFERENCES CITED

[1] R. Yaqub, “Introduction to the Smart Grid,” presented at The University of Tennessee at Chattanooga, Chattanooga, TN, Jan. 2011.

[2] C. R. Mason, “Philosophy of Protective Relaying,” in *The Art & Science of Protective Relaying,* New York: Wiley, 1956, ch. 1, pp. 1-9.

[3] J. L. Blackburn, “Introduction and General Philosophies,” in *Protective Relaying Principles and Applications,* 3rd ed. Boca Raton, FL: CRC Press, 2007, ch. 1, pp. 31-34, 48-56.

[4] P. M. Anderson, “System Characteristics,” in *Power System Protection,* New York: Wiley, 1999, ch. 5, sec. 5.1, pp. 148.

[5] P. M. Anderson, “Introduction,” in *Power System Protection,* New York: Wiley, 1999, ch. 1, sec. 1.2-1.5, pp. 4-11.

[6] W. A. Elmore, “Protection Against Transients and Surges,” in *Protective Relaying Theory and Applications,* New York: CRC Press, 2004, chap. 7, sec. 3.2, pp.109

[7] P. M. Anderson, “Protection Measurements and Controls,” in *Power System Protection,* New York: Wiley, 1999, ch. 2, sec. 2.1, pp. 17-19.

[8] P. M. Anderson, “Relay Characteristics,” in *Power System Protection,* New York: Wiley, 1999, ch. 3, sec. 3.3, pp.74-77.

[9] J. L. Blackburn, “Microprocessor Applications and Substation Automation,” in *Protective Relaying Principles and Applications,* 3rd ed. Boca Raton, FL: CRC Press, 2007, ch. 15, pp. 579-80, 589-90, 597-600.

[10] A. G. Phadke and J. S. Thorp, “Introduction to computer relaying,” in *Computer Relaying for Power Systems, 2md ed. West Sussex, England: Wiley, 2009, ch. 1,sec. 1.3, pp. 7.*

[11] P. M. Anderson, “Markov Modeling of Protective Systems,” in *Power System Protection,* New York: Wiley, 1999, ch. 28, pp.1208-10, 1226-28.

[12] *CMC 256 Plus – The Reference*, Omicron, Klaus, Austria, 2011

[13] *Software Overview*, Omicron, Klaus, Austria, 2011

[14] Omicron Test Universe version 2.30. Klaus, Austria:Omicron, 2009

[15] *Doble Protection Testing Software Suite*, Doble Engineering, Watertown, MA, 2008

[16] P. Revuru, (1997, Feb). *Protective Relay Maintenance* [Online]. Available: http://www.electricity-today.com/et/Feb97/febrelay.htm

[17] Enoserv, “Enoserve RTS” [Online]. Available: http://enoserv.com/ENOSERV-RTS-relay-testing-software.html

[18] N.I. Santoso and J.Y. Avins, “Real-time software testing for microprocessor based protective relays,” *Power Delivery*, vol. 9, no. 3, pp. 1359-1367, July, 1994.

[19] J. J. Kumm, M. S. Weber, E. O. Schweitzer, III, and D. Hou, “Philosophies for Testing Protective Relays,” proceedings of the 48th Annual Georgia Tech Protective Relaying Conference, Atlanta, GA, May 1994.

[20] *D90-Plus Line Distance Protection System Instruction Manual*, GE Multilin, Ontario, Canada, 2010.

[21] *CMC 256plus: 6 Phase Current + 4 Phase Voltage Test Set and Universal Calibrator Technical Data*, Omicron, Klaus, Austria, 2011

[22] Alstom Grid, “Distance Protection,” in *Network Protection & Automation Guide*. Stafford, UK: Alstom Grid, 2011, pp. 172.

APPENDIX A

FORWARD DECLARATIONS AND STRUCTS

```
//wrapper.h
#ifndef wrapper_h  //prevent functions from being defined multiple times on multiple
includes
#define wrapper_h

#include <iostream>
#include <math.h>
#include <atlstr.h>  //include to use Cstring without requiring MFC being turned on
#include <atlbase.h>  //included for VB wrapper functions, this header file automatically
imports windows.h
#include <stdio.h>
#import  "CMEngAL.tlb"

//struct used to hold information about the CMC
struct CM
{
       CMEngine::ICMEngine* Engine;  //instance used to create deviceID
       long deviceId;  //used to reference the CMC
};

struct state
{
       double Vall, Iall, angle, frequency;  //used for the generic settings of I & V
       double Vangle[3], Iangle[3];
       double voltages[3];  //3 entry arrays for the case where individual phases need to
be customized
       double currents[3];
       int time; //time to next state or time-out time in ms
       int binaryAck;  //store the state of the relay, if binary input from CMC high then
relay trip circuit is energized
       struct state *next;  //used to create linked list
};

/////forward declarations for CMEngine wrapper functions////
void engine_object(CMEngine::ICMEngine** g_pEngine);
int lock_CMC(CMEngine::ICMEngine* g_pEngine, long* g_nDevId);
int unlock_CMC(CMEngine::ICMEngine* g_pEngine, long g_nDevId);
LPTSTR Exec(LPCTSTR szCmd, CM *CMC);

/////forward declarations for CMEngine functionality////

       ////Output////
void Vout(double amplitude, CM *CMC);
void Vout(int output, double amplitude, CM *CMC);
void Vout(int output, double amplitude,double frequency, CM *CMC);
void Vphase(int output, double angle, double amplitude, CM *CMC);
void Vphase(double angle, double amplitude, CM *CMC); //create same phase between each
signal
void Vstep(int output, double initialAmp, double finalAmp, CM *CMC);
void Iout(int output, double amplitude, double frequency, CM *CMC);
void Iphase(double angle, double amplitude, CM *CMC);
void Iphase(int output,double angle, double amplitude, CM *CMC);
void Istep(int output, double initialAmp, double finalAmp, CM *CMC);
void Voff(int output, CM *CMC);
void Ioff(int output, CM *CMC);
void allOff(int output, CM *CMC);
              ////Input///
LPTSTR Vrms(int input, double range, int integration_period, CM *CMC);
LPTSTR BinIn(int input, CM *CMC);
int decodeBinary(LPTSTR binResult);
void closeBreaker(CM *CMC);

////forward declarations for States////
int addState(state **init_state, double voltage, double current, double phaseAngle, int
time);
int addState(state **init_state, double Vphases[], double Iphases[], double vAngle[],
double iAngle[], int time);
double* exec_states(state *init_state, int testSelect, CM *CMC);
void print_states(state *init_state);

#endif
```

APPENDIX B

FUNCTIONALITY FOR INTERFACEING WITH OMICRON CMC 256

```
//CMEngineWrapper.cpp
//file creates functions to lock and unlock the CMC for ease of use
#include "wrapper.h"

//create an instance of CMEngine type to instantiate g_pEngine
void engine_object(CMEngine::ICMEngine** g_pEngine)
{
              //Create Engine Object
       HRESULT hr = S_OK;
       hr = CoCreateInstance( __uuidof(CMEngine::CMEngine),
       NULL,
       CLSCTX_INPROC_SERVER,
       __uuidof(CMEngine::ICMEngine),
       reinterpret_cast<void **> (g_pEngine) );

       if (FAILED(hr))
       {
              _tprintf(_T("Couldn't create engine object, possible soln's include:\n-
Change Common Language Runtime support to /clr \n \
                     -check pointers for g_pEngine, ensure g_pEngine is instantiated and
returned to main"));
              getchar();
              exit(0);
       }

}
//returns a value of 1 if the CMC can be locked
int lock_CMC(CMEngine::ICMEngine* g_pEngine, long* g_nDevId)
{
       try
       {
              // Search for CMCs and lock the first CMC for use
              g_pEngine->DevScanForNew(false);
              _bstr_t szResult;    //_bstr_t is a C++ class wrapper for the Visual Basic
string type
              szResult = g_pEngine->DevGetList(CMEngine::lsUnlockedAssociated);

              if (szResult.length() == 0)
              {
                     _tprintf(_T("No CMC available\nPress ENTER to continue ...\n"));
                     getchar();
                     CoUninitialize();
                     return -1;
              }


              //print out version info to show that it is communicating and can be locked
              _tprintf(_T("DevGetList (CMC Device Info): %s\n\n"), (LPTSTR) szResult);
              // Extract ID of the first CMC and lock it
              _stscanf_s((LPTSTR) szResult, _T("%d"), g_nDevId);
              g_pEngine->DevLock(*g_nDevId);  //Lock CMC
              return 1;

       }
       catch(_com_error ex)
       {
```

```
		_tprintf(_T("CMEngineError: %s\nPress ENTER to continue ...\n"), (LPTSTR)
ex.Description());
		getchar();
		exit(0);
	}
}

//returns a value of 1 if the CMC is successfully unlocked
int unlock_CMC(CMEngine::ICMEngine* g_pEngine, long g_nDevId)
{
	try
	{
		g_pEngine->DevUnlock(g_nDevId);
		return 1;
	}
	catch(_com_error ex) {return -1;}
}


// Send command to CMC and read return value
LPTSTR Exec(LPCTSTR szCmd, CM *CMC)
{
	try {
		return (CMC->Engine)->Exec(CMC->deviceId, szCmd);
	} catch(_com_error ex) {
		_tprintf(_T("Failed to execute engine command:%s\n%s"),
(LPTSTR)szCmd,(LPTSTR) ex.Description());
	}

	return _T("");
}
```

APPENDIX C

WRAPPER FUNCTIONS THAT CONTAIN EXECUTABLE COMMANDS

```
//wrapperFunctions.cpp
//functions provided to execute CMC commands for ease of use
#include "wrapper.h"

//Configures a single voltage output with a default phase angle of 0
void Vout(int output,double amplitude,double frequency, CM *CMC)
{
	//convert parameters into LPCTSTR to be sent to the CMC for execution
	CString configure;
	CString turnOn;
	configure.Format(_T("out:ana:v(1:%d):a(%f);f(%f)"),output,amplitude,frequency);
	turnOn.Format(_T("out:ana:v(1:%d):on"),output);
	//send commands to CMC
	Exec(configure, CMC);
	Exec(turnOn, CMC);
}

//sets all Voltage outputs with same amplitude varying by the phase angle specified
void Vphase(double angle, double amplitude, CM *CMC)
{
	double frequency=60;
	CString phase1,phase2,phase3;
	CString turnOn;
	phase1.Format(_T("out:ana:v(1:1):a(%f);f(%f);p(0)"),amplitude,frequency);
	phase2.Format(_T("out:ana:v(1:2):a(%f);f(%f);p(-%f)"),amplitude,frequency,angle);
	phase3.Format(_T("out:ana:v(1:3):a(%f);f(%f);p(%f)"),amplitude,frequency,angle);
	Exec(phase1,CMC);
	Exec(phase2,CMC);
	Exec(phase3,CMC);
	Exec(_T("out:ana:on"),CMC);
}

//sets an individual voltage output
void Vphase(int output, double angle, double amplitude, CM *CMC)
{
	double frequency=60;
	CString phase,turnOn;
	phase.Format(_T("out:ana:v(1:%d):a(%f);f(%f);p(%f)"),output,amplitude,frequency,an
gle);
	turnOn.Format(_T("out:ana:v(1:%d):on"),output);
	Exec(phase,CMC);
	Exec(turnOn,CMC);
}

//steps a voltage output while keeping the waveform's phase
void Vstep(int output, double initialAmp, double finalAmp, CM *CMC)
{
	double frequency=60;
	CString configure_init,configure_final;
	CString turnOn;
	turnOn.Format(_T("out:ana:v(%d):on"),output);
	configure_init.Format(_T("out:ana:v(%d):a(%f);f(%f)"),output,initialAmp,frequency)
;
	configure_final.Format(_T("out:ana:v(%d):a(%f);f(%f)"),output,finalAmp,frequency);
	Exec(configure_init,CMC);
	Exec(turnOn,CMC);
	getchar();  //will be changed to Sleep(ms)
	Exec(_T("out:ana:pmode(diff)"), CMC);  //keep the phase when future on commands 
are applied
	Exec(configure_final,CMC);
	Exec(turnOn,CMC);
}

void Voff(int output, CM *CMC)
{
	//convert parameter into LPCTSTR to be sent to the CMC for execution
	CString turnOff;
	turnOff.Format(_T("out:ana:v(%d):off"),output);
	//send command to CMC
	Exec(turnOff, CMC);
}

//output current on all phases with same parameters
void Iout(int output, double amplitude, double frequency, CM *CMC)
{
	//convert parameters into LPCTSTR to be sent to the CMC for execution
	CString configure;
	CString turnOn;
	configure.Format(_T("out:ana:i(1:%d):a(%f);f(%f)"),output,amplitude,frequency);
	turnOn.Format(_T("out:ana:i(1:%d):on"),output);
	//send commands to CMC
	Exec(configure, CMC);
	Exec(turnOn, CMC);
}

//output current separated by specified phase angle
void Iphase(double angle, double amplitude, CM *CMC)
{
	double frequency=60;
	CString phase1,phase2,phase3;
	phase1.Format(_T("out:ana:i(1:1):a(%f);f(%f);p(0)"),amplitude,frequency);
	phase2.Format(_T("out:ana:i(1:2):a(%f);f(%f);p(-%f)"),amplitude,frequency,angle);
	phase3.Format(_T("out:ana:i(1:3):a(%f);f(%f);p(%f)"),amplitude,frequency,angle);
	//send commands to CMC
	Exec(phase1, CMC);
	Exec(phase2, CMC);
	Exec(phase3, CMC);
	Exec(_T("out:ana:on"), CMC);
}

//configure an individual phase current amplitude
void Iphase(int output,double angle, double amplitude, CM *CMC)
{
	double frequency=60;
	CString phase,turnOn;
	phase.Format(_T("out:ana:i(1:%d):a(%f);f(%f);p(%f)"),output,amplitude,frequency,
angle);
	turnOn.Format(_T("out:ana:i(1:%d):on"),output);
	Exec(phase, CMC);
	Exec(turnOn, CMC);
}

//take an initial current and step it up by a given value
```

```
void Istep(int output, double initialAmp, double finalAmp, CM *CMC)
{
       double frequency=60;
       CString configure_init,configure_final;
       CString turnOn;
       configure_init.Format(_T("out:ana:i(%d):a(%f);f(%f)"),output,initialAmp,frequency)
;
       configure_final.Format(_T("out:ana:i(%d):a(%f);f(%f)"),output,initialAmp,frequency
);
       turnOn.Format(_T("out:ana:i(%d):on"),output);
       Exec(configure_init, CMC);
       Exec(turnOn, CMC);
       Exec(_T("out:ana:pmode(diff)"), CMC);  //keep the phase when future on commands
are applied
       Exec(configure_final, CMC);
}
void Ioff(int output, CM *CMC)
{
       //convert parameter into LPCTSTR to be sent to the CMC for execution
       CString turnOff;
       turnOff.Format(_T("out:ana:i(%d):off"),output);
       //send command to CMC
       Exec(turnOff, CMC);
}

void allOff(int output, CM *CMC)
{
       CString Ioff, Voff;
       Ioff.Format(_T("out:ana:i(%d):off"),output);
       Voff.Format(_T("out:ana:v(%d):off"),output);
       Exec(Ioff,CMC);
       Exec(Voff,CMC);
}

//send signal to E7 contact on relay using wetted binary output to initiate close circuit
void closeBreaker(CM *CMC)
{
       allOff(1, CMC); //clear the fault
       Sleep(100);  //wait 100ms to account for mechanical action then close the breaker
       Exec(_T("out:bin(1):set(0b0001)"),CMC);  //close the breaker
       Sleep(50); //wait for the relay to close then turn off close command
       Exec(_T("out:bin(1):off(1)"),CMC);
}

//configure and read rms voltage
//The following ranges are available for the CMC 256 or newer test set: 100mV, 1V, 10V,
100V and 600V
//integration period specified in seconds, required to wait specified amount of time
before result can be returned
LPTSTR Vrms(int input, double range, int integration_period, CM *CMC)
{
       _bstr_t voltage; //variable to store value of voltage
       //convert parameters into LPCTSTR to be sent to the CMC for execution
       CString configure, rms, getV;
       configure.Format(_T("inp:ana(%d):def(v,%f)"),input,range);
       rms.Format(_T("inp:ana(%d):rms?(%f)"), input, integration_period);
       getV.Format(_T("inp:ana:get?(%d)"),input);
```

```
	//send commands to CMC
	Exec(configure, CMC);
	Exec(rms, CMC);
	//wait integration period before reading measurement, sleep function takes
millisecond arg
	Sleep(integration_period*1000);
	voltage=(LPTSTR)Exec(getV, CMC);
	return voltage;
}

//configure specified binary input as potential free
//returns result in string "<module index>, <binary input>"
LPTSTR BinIn(int input, CM *CMC)
{
	_bstr_t binInput;
	CString bin_config;
	//configure specified input from binary input module 1 to read potential free
input (pf)
	bin_config.Format(_T("inp:bin(1:%d):def(pf)"),input);
	Exec(bin_config,CMC);
	binInput=Exec(_T("inp:bin(1):get?"),CMC);  //read input from binary module 1
	return binInput;
}

int decodeBinary(LPTSTR binResult)
{
	CString	strip=(LPTSTR)binResult;
	wchar_t firstNum;  //hold the value to strip off of the beginning of the string
which may vary depending on the binary input definition
	int atoi;
	firstNum=strip.GetAt(0);  //determine what the input module is from the definition
of Omicron's Binary Inputs
	strip.TrimLeft(firstNum);  //remove leading 1 stating binary input module
	strip.TrimLeft(_T(","));   //remove other excess characters
	strip.TrimRight(_T(";"));
	atoi=_wtoi(strip);  //convert LPTSTR to int
	return atoi;
}
```

APPENDIX D

FUNCTIONS THAT CREATE AND EXECUTE LINKED LIST OF STATES

```
//faultStates.cpp
//functions to create states to be executed by the CMC
#include "wrapper.h"
#include <ctime>
//#include "time.h"
#define _USE_MATH_DEFINES //defined to allow the use of pi constant M_PI
#include <cmath>     //included for trig functions
using namespace std;

//create a state to be executed by the CMC. phase angles are equally spaced with
magnitudes the same across all phases. time is in ms
int addState(state **init_state, double voltage, double current, double phaseAngle, int
time)
{
       int numStates=1; //counter to determine the number of states
       state *currentState, *head;
       currentState=(state*)malloc(sizeof(state));
       head=*init_state;  //copy 1st state into temp state that points to begining of
linked list

       //define state variables
       currentState->Vall=voltage;
       currentState->Iall=current;
       currentState->angle=phaseAngle;
       currentState->frequency=60;
       currentState->time=time;

       if(*init_state==NULL)
       {
              *init_state=currentState;  //current state becomes first state in list
(generally the pre-fault state)
              (*init_state)->next=NULL;
       }
       else
       {
              //traverse list
              while(head->next !=NULL)
              {
                     numStates++;
                     head=head->next;
              }
              numStates++;

              //add current state to end of list
              currentState->next=NULL;
              head->next=currentState;
       }
              return numStates;
}

//individually define I & V magnitudes for a state. time is in ms. Returns the number of
states in the linked list
int addState(state **init_state, double Vphases[], double Iphases[], double vAngle[],
double iAngle[], int time)
{
       int numStates=1; //counter to determine the number of states
```

```
	state *currentState, *head;

	currentState=(state*)malloc(sizeof(state));
	head=*init_state;  //copy 1st state into temp state that points to begining of
linked list

	//define state variables
	currentState->voltages[0]=Vphases[0];
	currentState->voltages[1]=Vphases[1];
	currentState->voltages[2]=Vphases[2];
	currentState->currents[0]=Iphases[0];
	currentState->currents[1]=Iphases[1];
	currentState->currents[2]=Iphases[2];
	currentState->Vangle[0]=vAngle[0];
	currentState->Vangle[1]=vAngle[1];
	currentState->Vangle[2]=vAngle[2];
	currentState->Iangle[0]=iAngle[0];
	currentState->Iangle[1]=iAngle[1];
	currentState->Iangle[2]=iAngle[2];
	currentState->time=time;
	currentState->Vall=-1;  //all 3 outputs declared separately so these variables are
not used
	currentState->Iall=-1;


	if(*init_state==NULL)
	{
		*init_state=currentState;  //current state becomes first state in list
(pre-fault state)
		(*init_state)->next=NULL;  //dereferencing a double pointer to modify a
struct variable
	}
	else
	{
		//traverse list
		while(head->next !=NULL)
		{
			numStates++;
			head=head->next;
		}
		numStates++;

		//add current state to end of list
		currentState->next=NULL;
		head->next=currentState;
	}
		return numStates;
}


//takes the head of the linked list and executes all states pointed to by the head node
//tests Phase IOC if testSelect=0 and Phase Distance if testSelect=1
//If IOC trips breaker, function returns address to current value for all "in phase"
quantities, returns 3D array for individually configured phases
//outputs to file 3D array for phase Distance testing, 1st entry is 1 for coordinates
inside MHO circle, and -1 for outside
```

```
//next two entries contain the x and y coordinates of the impedance for plotting MHO
CIRCLE
//returns -1 if IOC fails to trip breaker or on completion of Phase Distance Testing
double* exec_states(state *head, int testSelect, CM *CMC)
{
	int DistanceTesting=testSelect;  //if set to one, function outputs to file
cordinates in MHO graph, if 0 return instance of IOC values
	int readBinary;
	int counter=0;  //counter used for timing the length of states and the repetition
of checking for breaker opening
	int state=1;  //counter to determine which state trips the breaker
	head->binaryAck=0;  //declare that current state has not tripped the breaker
	double x,y;  //used to perform polar to rectangular conversion
	double *funcReturn;
	double *coords=(double*)malloc(3*sizeof(double));  //hold the polar coordinates
for the MHO plot
	double fail=-1.0;  //return value when the Phase IOC fails to trip breaker
	double pass=1.0;  //return value upon completion of Distance testing
	clock_t start_time;  //timer variable used in specifying length of time per state

	FILE * csv=fopen("mhoCircle.csv","w");  //open file to plot MHO circle coordinates
	if(csv==NULL)
	{
		printf("file could not be opened\n");
		funcReturn=&fail;
		return funcReturn;
	}

	//check to see if breaker is closed
	readBinary=decodeBinary(BinIn(1,CMC));
	if(readBinary==1)
	{
		printf("Breaker already opened\n");
		closeBreaker(CMC);  //reset the breaker to begin testing
	}

	while(head != NULL)
	{
		counter=0;  //set timer to 0
		head->binaryAck=0;  //declare that current state has not tripped the
breaker
		if(head->Vall==-1 || head->Iall==-1 )  //if true -> individual phase
amplitudes are configured
		{
			Vphase(1, head->Vangle[0], head->voltages[0], CMC);
			Vphase(2, head->Vangle[1], head->voltages[1], CMC);
			Vphase(3, head->Vangle[2], head->voltages[2], CMC);
			Iphase(1, head->Iangle[0], head->currents[0], CMC);
			Iphase(2, head->Iangle[1], head->currents[1], CMC);
			Iphase(3, head->Iangle[2], head->currents[2], CMC);
		}
		//I & V are in phase
		else
		{
			Vphase(head->angle, head->Vall, CMC);
			Iphase(head->angle, head->Iall, CMC);
		}
		readBinary=decodeBinary(BinIn(1,CMC));  //read before the beginning of loop
then read every millisecond
		start_time=head->time + clock();  //start_time holds the maximum time that
will elapse before moving to next state
		while(start_time > clock())  //execute state for specified length of time
		{
			counter++;
			if(counter%10000==0)  //check for binary input approximately every
milisecond
				readBinary=decodeBinary(BinIn(1,CMC));

			if(readBinary==1)  //relay acknowledges fault
			{
				head->binaryAck=1;
				closeBreaker(CMC);  //clears the fault and energizes the
relay's close circuit
				//record the coordinates in the MHO circle
				if(DistanceTesting==1)
				{
					coords[0]=1.0;
					coords[1]=(head->voltages[0])/(head->currents[0]);
					coords[2]=(head->Vangle[0])-(head->Iangle[0]);
					coords[2]=(coords[2]*M_PI)/180;  //convert degrees to
radians
					x=coords[1]*cos(coords[2]);
					y=coords[1]*sin(coords[2]);
					fprintf(csv,"%f,%f,%f\n",coords[0],x,y);
					break;  //break out of the inner while loop that was
used as a timer
				}
				//return current amplitude if Phase IOC testing tripped the
breaker
				else if(head->Vall==-1 || head->Iall==-1 )  //true if
individual phases are configured
					return head->currents;
				else
					return &(head->Iall);   //return fault current value

			}
		}

		//record coordinates of state outside of MHO circle
		if(DistanceTesting==1  && head->binaryAck==0)  //return coordinates outside
of MHO circle
		{
			coords[0]=-1.0;
			coords[1]=(head->voltages[0])/(head->currents[0]);
			coords[2]=(head->Vangle[0])-(head->Iangle[0]);  //perform
rectangular to polar conversion
			coords[2]=(coords[2]*M_PI)/180;  //convert degrees to radians
			x=coords[1]*cos(coords[2]);
			y=coords[1]*sin(coords[2]);
			fprintf(csv,"%f,%f,%f\n",coords[0],x,y);
		}

		head=head->next;
		state++;  //increment state counter (primarily used for debugging)
	}

	fclose(csv);
	if(DistanceTesting==1)
	{
		funcReturn=&pass;
		return funcReturn;
	}
	else //Phase IOC states failed to trip the breaker
	{
		funcReturn=&fail;
		return funcReturn;
	}
}

//print out info from each state
void print_states(state *init_state)
{
	state *head=init_state;
	int i=1;
	while(head != NULL)
	{
		printf("State %d:V:%f,T:%f\n", i,head->Vall,init_state->time);
		head=head->next;
		i++;
	}
}
```

APPENDIX E

MAIN METHOD TO SET UP TESTS AND ISSUE COMMANDS TO BEGIN EXECUTION

```
//main.cpp
///////CMEngine Test Code//////
/*
Before compiling:
	Under Configuration Properties->General:
		-Make sure Common Language Runtime support is set to /clr
		-Make sure Character set is changed to the option "Use Unicode Character
Set"

Additional Details:
	-C++ wrappper class for VB strings _bstr_t is used instead of native C++ strings
(you can thank Omicron for that!)
	-_bstr_t strings require _tprintf(_T("")) (included in atlbase.h) instead of
printf to properly print LPTSTR strings (in a VC++ fashion)
*/

//include forward declarations and CMEngine functions class, as well as CMfunctions
wrapper class
#include "wrapper.h"


void main()
{
	//struct holds device instances
	CM *cmc= (CM *) malloc (sizeof(CM));
	//linked list struct contains quantities at each state
	state *IOCinit_State=(state*) malloc(sizeof(state));
	IOCinit_State=NULL;  //list of states is empty until defined
	state *Dist_init_State=(state*) malloc(sizeof(state));
	Dist_init_State=NULL;  //list of states is empty until defined

	FILE * csvIOC=fopen("phaseIOC.csv","w");
	int numStates;		//used to hold return value from add_State to hold the number
of entries in the linked list
	int fail; //check to see if CMC was locked for execution
	double *Istep;  //used for return value for phase IOC testing
	//counters for looping over MHO circle
	double mag;
	int ang;
	//counters for phase IOC
	double i;
	int j;
	double *Coord=(double*)malloc(3*sizeof(double));  //Array to hold MHO coordinates,
should they need to be returned to main
	double V[3]={0,0,0};  //at (0,0) V=0 and incresases uniformly in all directions
	double I[3]={1.324,1.324,1.324};  //I magnitude stays constant for Distance
testing
	double vAngle[3]={0,-120,120};     //V angle stays constant for Distance testing
	double iAngle[3]={0,-120,120}; //at x=0 I angle is 0,-120,120 for phase A,B, & C
respectively and increases in clockwise direction

	//create Engine instance before passing Engine variable to lock the CMC
	engine_object(&cmc->Engine);
	//modifys the device ID used to reference the CMC, returns -1 if function fails
	fail=lock_CMC(cmc->Engine, &cmc->deviceId);
	//check to see if CMC was locked for sending commands before continuing
	if(fail<0)
```

```
		exit(0);

	Exec(_T("out:aux(1):a(110)"),cmc);  //turn breaker simulator on

	//Distance testing
	numStates=addState(&Dist_init_State,V,I,vAngle,iAngle,300); //first state at (0,0)
has only 1 set of phase angles to be tested
	for(mag=0.1;mag<2.6;mag+=0.1)  //loop over the voltage values in MHO circle
	{
		V[0]=mag;
		V[1]=mag;
		V[2]=mag;
		iAngle[0]=0;
		iAngle[1]=-120;
		iAngle[2]=120;

		//because signs swap on phase angles in Omicron, a different or loop is
used for either side of the axis to be consistent
		for(ang=0; ang<180; ang++)
		{
			iAngle[0]++;
			iAngle[1]++;
			iAngle[2]++;
			//MHO circle lies primarily in upper two quadrants so the lower 2
are not tested, but they could uncomment the statement below should they want to test
reverse directional faults
			//numStates=addState(&Dist_init_State,V,I,vAngle,iAngle,100);
//begin after 0 degrees since angles are reversed and swept over to 0
		}
		//swap phase angle signs to stay consistent with Omicron software
		iAngle[0]-=360;
		iAngle[1]-=360;
		iAngle[2]-=360;
		for(ang=0; ang<180; ang++)
		{
			iAngle[0]++;
			iAngle[1]++;
			iAngle[2]++;
			numStates=addState(&Dist_init_State,V,I,vAngle,iAngle,100);
		}
	}
	Coord=exec_states(Dist_init_State,1,cmc);  //perform distance testing, Coord
receives -1 upon completion


	//Phase IOC testing
	//set up states with 3 phase I & V to execute at speed of 100ms
	for(i=.9; i<1.1;i+=.001)
	{
		for(j=0;j<3;j++)
		{
			I[j]=i;
			V[j]=1.5;
		}
		numStates=addState(&IOCinit_State,V,I,vAngle,iAngle,100);
	}
```

```
        for(j=0;j<1000;j++)
        {
                Istep=exec_states(IOCinit_State,0,cmc);
                fprintf(csvIOC, "%f\n", *Istep);  //records value of A phase current since
current is equal across all phases
                Sleep(300); //allow mechanical operation of Omicron's CMC to reset
        }


        allOff(1,cmc);
        //Exec(_T("out:aux(1):off"),cmc);  //once turned off you have to use def(>0) and
restart machine

        fclose(csvIOC);
        unlock_CMC(cmc->Engine, cmc->deviceId);  //release control of the Omicron CMC256-6

        //Memory cleanup, release Engine instance
        (cmc->Engine)->Release();  //parenthesis used just to clarify the struct reference
        CoUninitialize();
        free(cmc);
        free(IOCinit_State);
        free(Dist_init_State);
        free(Coord);

}
```

VITA

Jason Johnston attended UTC for his undergraduate degree in Computer Engineering. After honing software development skills he discovered a passion for more abstract coding practices. He worked with a previous UTC Computer Science department head, Dr. Andrew J. Novobilski, using a parallel programming environment made available through GPU cores on an Nvidia graphics card and accessed by the CUDA API to implement Bayesian networks. As a graduate student he attended a parallel programming course with the Computational Engineering department at UTC and gained a desire to further his skills in this area. He also has a strong desire to pursue a career in hardware design and development after taking courses with Dr. Stephen Craven on hardware description languages. At the time of his thesis submittal he was employed as an intern with the Power Quality group at Tennessee Valley Authority. Working primarily outside of his desired field of study in this power engineering environment he kept himself occupied teaching himself Python and performing various other software development tasks.