A CASE-BASED INTELLIGENT TRAINING AGENT WITH AN APPLICATION TO POWER SYSTEM RESTORATION

A Thesis Submitted to the College of
Graduate Studies and Research
in Partial Fulfillment of the Requirements
for the Degree of Master of Science
in the Department of Electrical Engineering
University of Saskatchewan
Saskatoon, Saskatchewan

by

Bixia Zhou

July 1997

PERMISSION TO USE

ACKNOWLEDGMENTS

The guidance and assistance throughout the course of this work from the author's supervisor, Dr. Nurul Chowdhury is thankfully acknowledged.

The author would like to express her heartfelt gratitude and a deep feeling of indebtedness to her husband, Mr.Yusheng Wu, for his consistent support, encouragement and tolerance of her long absence from home while working on this thesis.

Financial assistance in the form of SaskPower Graduate Study Scholarship and University Teaching Fellowship is acknowledged.

DEDICATION

This thesis is dedicated to the author's little lovely daughter C.C.

A CASE-BASED INTELLIGENT TRAINING AGENT WITH AN APPLICATION TO POWER SYSTEM RESTORATION

Candidate: Bixia Zhou
Supervisor: Dr. Nurul Chowdhury
Master of Science Thesis Submitted to the
College of Graduate Studies and Research
University of Saskatchewan
Summer 1997

ABSTRACT

A speedy restoration of a power system can minimize the consequences of a total or partial blackout. Due to the increase in system reliability, major disturbances occur less frequently and consequently system operators receive little experience in restoration. Power system operators seldom get working knowledge of restoration of a whole system from their daily work. However, system operators can be trained to handle the necessary steps of restoration with the help of a simulation program. This research attempts to address this need.

In this research, a simulation program named as Intelligent Training Agent (ITA) has been developed to train system operators the steps of power system restoration. An object-oriented approach has been utilized to develop the graphical user interface of the simulation program. A case-based reasoning approach has been applied to form the required knowledge base of the simulation program and to perform the reasoning task. Several numerical analysis algorithms have been used to verify the restoration actions. By using the case-based reasoning mechanism, the proposed intelligent agent can provide a restoration plan when the initial system conditions are known and the goal state of the system is submitted to the ITA. System operators, therefore, can interact with the agent through the object-oriented graphical interface. The interface will enhance the operator's learning process. The structure of the agent consists of an object-oriented graphical interface block, a case-based reasoning block, and an action check block.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF SYMBOLS AND ABBREVIATIONS

ITA	Intelligent Training Agent
CB	Circuit Breaker
PSR	Power System Restoration
GUI	Graphical User Interface
OWL	Object Window Library
OOA	Object-Oriented Analysis
OOD	Object-Oriented Design
OOI	Object-Oriented Implementation
GDI	Graphical Development Interface
CBR	Case-Based Reasoning
DNP	Decoupled Newton-Raphson Method
EMS	Energy Management System
PQ	Load Bus
PV	Generation Bus
$\bar{I}$	Current
$\overline{V}$	Voltage
$\lvert V_{i\min} \rvert$	Low limit of voltage magnitude at node i
$\lvert V_{i\max} \rvert$	Upper limit of voltage magnitude at node i
$\lvert V_i \rvert$	Voltage magnitude at node i
ΔV_i	Voltage magnitude mismatch at node i
θ_i	Phase angle at node i
θ_{ij}	Phase angle between node i and node j

$\Delta\theta_i$	Phase angle mismatch at node i
P	Power
P_i	Power injection at node i
P_{is}	Specified power at node i
ΔP_i	Power mismatch at node i
P_{ij}	Power flow between node i and node j
$P_{ij\max}$	Upper limit of power flow between node i and node j
Q	Reactive power
Q_i	Reactive power injection at node i
Q_{is}	Specified reactive power at node i
Q_{ij}	Reactive power flow between node i and node j
ΔQ_i	Reactive power mismatch at node i
$\overline{S}$	Complex power $\overline{S} = P + jQ$
S_{ij}	Complex power flow between node i and node j
G	Nodal Conductance
G_{ij}	Conductance between node i and node j
B	Nodal Susceptance
B_{ij}	Susceptance between node i and node j
Y	Nodal Admittance Y=G+jB
R_{ij}	Resistance between node i and node j
X_{ij}	Reactance between node i and node j
H_{ij}	Partial derivative of power mismatch at node i to phase angle at node j
H_{ii}	Partial derivative of power mismatch at node i to phase angle at node i
N_{ij}	Partial derivative of power mismatch at node i to voltage magnitude at node j
N_{ii}	Partial derivative of power mismatch at node i to voltage magnitude at node I

J_{ij}	Partial derivative of reactive power mismatch at node i to phase angle at node j
J_{ii}	Partial derivative of reactive power mismatch at node i to phase angle at node i
L_{ij}	Partial derivative of reactive power mismatch at node i to voltage magnitude at node j
L_{ii}	Partial derivative of reactive power mismatch at node i to voltage magnitude at node i
H	Submatrix of Jacobian matrix
N	Submatrix of Jacobian matrix
J	Submatrix of Jacobian matrix
L	Submatrix of Jacobian matrix
K	Transformer Ratio
Yc	Line to Ground Capacitance

1. INTRODUCTION

1.1 Electric Power System Restoration Issues

It is well known that an electric power system has three operating states: normal, emergency and restorative states. Most of the time a power system operates in a normal state. A large disturbance may, however, put the system into an emergency state. A large disturbance, for example, a fault in a power system that triggers the fault clearance protection relay or failure extension protection relay to trip the circuit breaker (CB) to clear the fault, may cause some degree of collapse, or blackout. The extreme consequence of this occurrence is total blackout. In a total blackout condition, a system basically has no sustainable generation connected to its transmission system.

The system then enters into a restorative state in which system operators try to reenergize the area where the power is down. The control process whereby a system is reenergized is referred to as power system restoration (PSR).

The problem of power system restoration following a complete or partial collapse is practically as old as the electric utility industry itself. It is recognized that the effects, both real and potential, of a prolonged blackout on the public, the economy, and on the power system equipment make a rapid, effective restoration very important. On November 9 of 1965, a blackout in Northeastern portion of the United States, including New York City and six other states, brought a loss of 21,000 MW load to an area of approximately 200,000 km^2. The total cost of the blackout has been estimated to be more than \$100 million [1]. A blackout in France on December 19, 1978, affected three-quarters of the country -a loss of total 29,000 MW load and cost of approximately \$200 million [1]. It was reported that 67% of Sweden lost power, costing Skr 300 million during a blackout that occurred in Sweden on December 27 of 1983 [1]. From the facts stated in references, it has been found that due to the size and complexity of modern power systems and our growing dependence on electric energy, costs of blackouts have vastly increased. Although there has been an increase in power system reliability and major disturbances occur less frequently, the absolute security of system can hardly achieved. In a reliable system, operators receive little experience in restoration.

Power system restoration is a process of switching operations and scheduling generations while considering operating limits of generators, transmission lines, transformers, bus voltage, line flow, etc. These limits are rather complex and difficult to be characterized in a fixed form with conventional optimization. Therefore, it is a typical field where, under actual operation, troubles are remedied via the experience of skilled operators. If system operators are well prepared and adequately trained for power system restoration, the cost of system blackouts can be reduced significantly.

1.2 Power System Restoration Training

During restoration, system operators are faced with a state of their system that is quite different from that they are accustomed in day-to-day operation, and for which the Energy Management System (EMS) application programs at their disposal were not designed and are not well adapted for power system restoration.

There are distinct differences between a normal state and a restorative state in terms of relevant models, objectives, and the information available at the control centers to support the models. In a normal state, the primary objective is to minimize the cost of production, subject to certain security constraints. EMS application programs that help to attain this goal incorporate models that primarily represent the steady state behavior of a power system.

By contrast, during a restoration process, the objective is to minimize restoration time and the amount of unserved energy, with security as a subsidiary objective, and without regard to production cost. Many dynamic phenomena, ignored during normal operation, play a critical role during restoration.

It is also important to note that during the restart and reintegration phases of PSR, a power system often consists of some islands. Most of the automatic controls have tripped or are deactivated and the system is primarily under manual control. During these early phases of PSR, wide voltage and frequency variations are tolerated. Under large perturbations of long duration, the models and simulations which have primarily been developed for small perturbations during a normal state will not accurately represent the behaviour of the power system and its components.

To help power system operators in dealing with system restoration, most power companies maintain restoration procedures developed as part of a general system restoration plan which is usually developed well ahead of time based on postulated scenarios, certain operating philosophies and practices. System operators should be trained to make them familiar with the restoration fundamentals, restart capabilities of their power plant and the reintegration peculiarities of their systems. They must also be capable of handling a myriad of unpredictable events which may occur during restoration.

In order to make system operators aware of the difficulties that may arise during system restoration and to help them improve their ability to handle those difficulties, all possible activities related to restoration process and their cause-effect must be addressed in a PSR training process. The activities of common concern in a PSR training generally include: how to assess the status of the collapsed system and deactivate automatic switching equipment; how to black-start units to supply station service to plants and substations, how to start-up the thermal units; how to energize large sections of transmission lines avoiding violation of sustained overvoltages and pick up load in large increments without declining the system frequency; how to reduce the standing phase angles when synchronizing subsystems and how to coordinate power plant start-up timings with load pick-ups to bring generators to their stable levels and within the range of major analog controllers.

1.3 Power System Restoration Training Tools

As mentioned earlier, due to the increase in power system reliability, major disturbances occur less frequently and consequently system operators receive little experience in restoration. The need to train power system operators to restore their system is well recognized. A variety of tools are being used to develop and deliver system restoration training programs. The salient features of some of these tools are explained below.

Self-Study Methods

In a self-study method, portions of restoration training are handled with written material or video tape presentations. Self-study methods comprise a relatively small portion of the overall training effort. Although an operator may learn some of the fundamentals through these methods, more practical means are needed where an operator would learn to apply the restoration principles.

Classroom Techniques

Classroom style training usually is conducted by a combination of system operation, generation plant, and field operation personnel. "What if" type situations usually are discussed in detail. These programs basically provide operators with a practical understanding of the basic principles involved in system restoration so that unexpected problems can be solved. The utility's restoration policy, and its goals, are the core of these programs.

The topics of classroom techniques fall into two categories:

1. Generic -This area includes necessary bulk power system restoration fundamentals, and addresses major issues that the operator needs to consider during a restoration.
2. Utility Specific -Training addresses a utility's restoration plan and its priorities, assessment methodologies, specific job responsibilities, communication techniques, and notification requirements are covered.

Drills

Drills are basically role-playing exercises which usually are expanded to include field, plant, and other utility personnel.

A key to a good drill lies in the extent of the exercise. It should involve as many people and events as would be involved in an actual bulk power system restoration. A major problem is the difficulty of making all kinds of personnel available for the drill and thus the exercises cannot be run frequently. Therefore, there is a growing attention on various applied tools and techniques used to develop and implement effective bulk power system restoration training programs.

Computer-Based Training

Computer-based training is receiving growing attention largely due to its ability to include simulations of restoration process in the training by utilizing a graphical user interface. Operators can make some interactions with simulation programs during their training process, visualize the process of cause-effect and then discover what works and what does not work. In an interactive environment, their learning abilities are significantly enhanced. Another advantage of computer-based training is that it can be designed as a student-paced, self-learning method. Operators can decide their own time of engagement in training; there is no need to coordinate and schedule a class with an instructor and other students. Since system operators typically work on rotating shifts, the ability to allow a student to schedule a training session at odd hours is invaluable. Very little work has been done in the computer-based training, the research work documented in this thesis is armed to meet this requirement.

1.4 Overview of the Research

The goal of the research is to develop a computer-based training simulator to train system operators in the necessary steps of power system restoration. There are several objectives which are related to reach the goal. The first objective is to develop an graphical user interface (GUI) which provides a basic foundation for a user to interact with and to visualize the actions of the simulator. The second objective is to develop a reasoning system which generates restoration plans. The final objective is to develop a validation system which provides system profiles to assess the restoration plans. Although traditional expert system techniques have proved to be more effective in proposing restoration plans than the analytical methods [2,3], the knowledge acquisition bottleneck and maintenance have become the major barriers in the application of the expert system techniques. A new expert system approach, called case-based reasoning (CBR), has been applied in this research. A CBR system proposes a solution to the problem on the basis of the past cases saved in it's library. Each subsequent solution becomes a new case and is saved in the case library. A CBR approach, therefore, can overcome the disadvantages of the traditional expert systems. CBR approaches can be utilized to solve a wide range of problems in various fields. Very little has been done to implement CBR approaches in the area of power system restoration, this research begins to fill the void by demonstrating that a CBR architecture is useful for generating restoration plan.

A computer-based simulation program called an intelligent training agent (ITA) has been designed and developed for training power system operators to handle restoration condition. The ITA has three subsystems shown in Figure 1.1: a graphical user interface (GUI), an expert system, and an action checker. An object-oriented approach has been used to develop the GUI. Various power system elements can be drawn, displayed and manipulated on the screen with the help of the GUI. A CBR approach has been utilized to develop the expert system. Some restoration scenarios from the past and their corresponding remedial actions have been collected and stored in the case library.

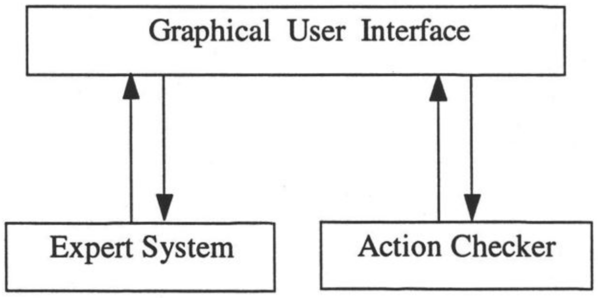


Figure 1.1 Architecture of the Intelligent Training Agent

When a new situation is submitted to the ITA, the ITA will retrieve the best matched case and provide a restoration plan. Based on the system information, the action checker subsystem will run a load flow program and provide a system profile to the operators as a form of feedback to their actions.

1.5 Structure of the Thesis

This thesis is organized into seven chapters. The first chapter introduces the restoration training problem, reviews briefly the concept of the problem and discusses the different kinds of training tools available at current time and presents the methodology which is used in this research. In Chapter 2, one of the object-oriented approach methods, called Fusion, is introduced [4], and its concept, terminology and notations are explained in detail. Chapter 3 lays out the issues in applying the Fusion techniques to develop a graphical user interface and presents the software implementation of the proposed techniques. Chapter 4 introduces the philosophy of CBR methodology and its architecture in more detail to allow understanding of the complexities involved and illustrates the benefits of CBR. Chapter 5 describes the design of the case-based reasoning system, strategy and algorithm which are used in the implementation and shows some test results. Chapter 6 presents the concept of the action checker subsystem and the algorithms which have been used in the software implementation. A brief summary of the work and some conclusions are provided in Chapter 7.

2. OBJECT-ORIENTED APPROACH

2.1 Introduction

In recent years, object-oriented approaches have become a popular choice for software developers. Software developed using an object-oriented approach is generally user-friendly and provides graphical interpretations of input/output and resulting system states. In an object-oriented approach, objects become the "atoms" of object-oriented computation, which are utilized to map elements in the physical world. The objects, not only consist of data to present their characteristics and state, but also, embed methods to describe their behaviour. Messages are sent between objects resulting in the invocation of methods which perform the necessary actions. The sender of the message does not need to know how the recipient organizes its internal state, except it's responses. This produces a programming model remarkably different from the traditional "functional" programming. In a traditional programming approach, the abstract nature of the programming method allows itself to fit in almost any domain, but there is no direct mapping to the physical world.

A traditional function-oriented computational model is shown in Figure 2.1. In this model, the atoms of computation are the functions which operate on a single shared state. The shared state is usually static, and it's structure does not change during execution, although the values in the structure do. Even if the system can be divided into independent modules, each with control over some part of the state, a function-oriented approach has no underlying model to determine how this division should be accomplished.

An object-oriented model is illustrated in Figure 2.2. The large rectangles represent the objects, and the arrows between them represent the messages. The state of an object, represented by the ovals on the diagram, is concealed inside the object. The only way to read or change an object's state is by sending messages, and each object "takes responsibility" for how it responds to a message.

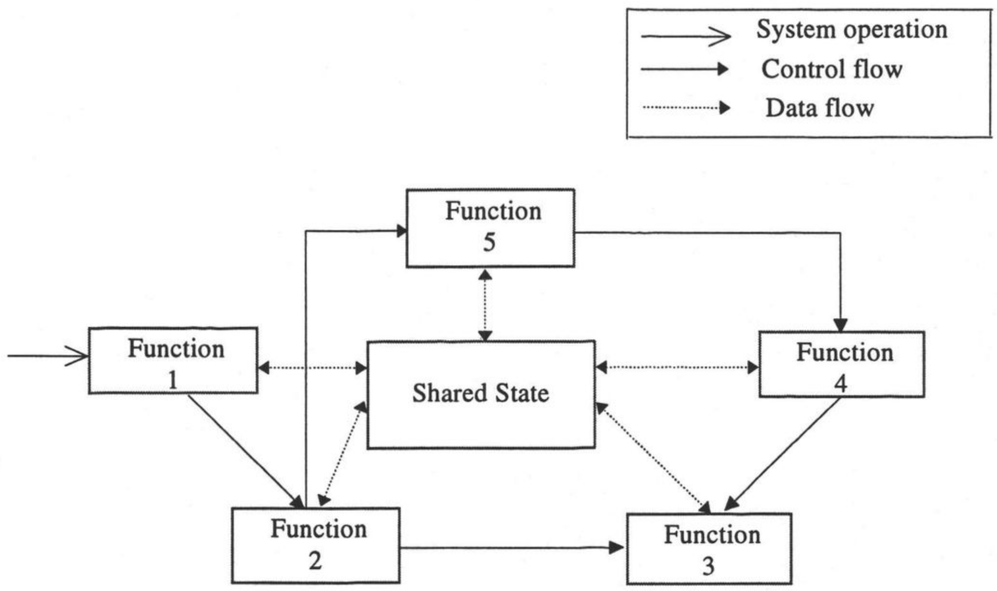


Figure 2.1 Function-Oriented Computational Model

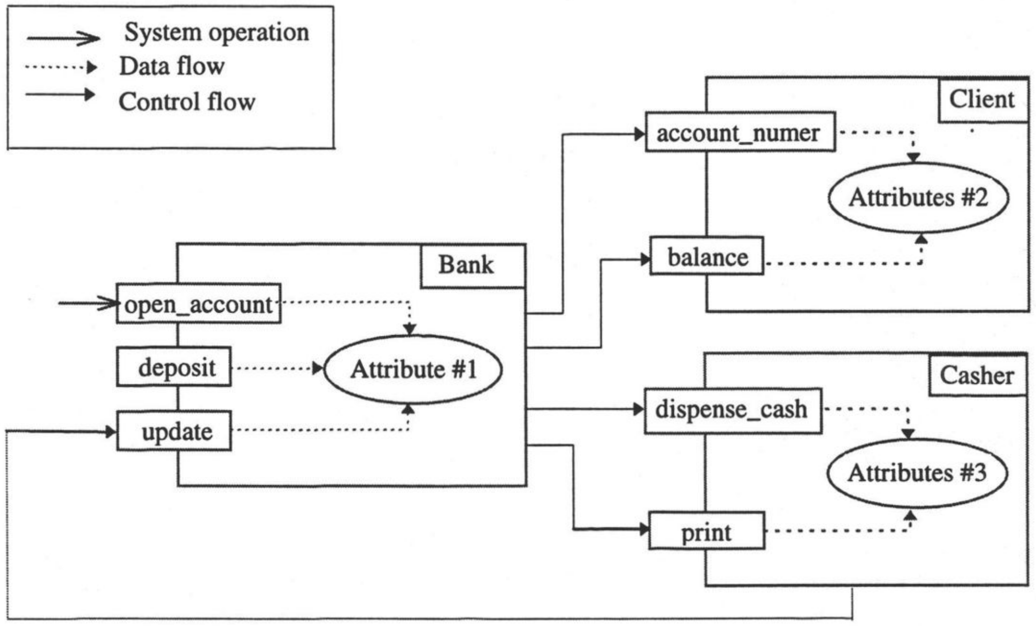


Figure 2.2 Object-Oriented Computational Model

Objects are grouped into classes when they have the same data structure and share the same interface, that is, respond to the same messages in the same ways. Classes are the building blocks of the object-oriented programming languages.

When an object-oriented program is executed, objects come into existence, perform actions, and then cease to exist or become inaccessible. An object-oriented computational model is, therefore, essentially dynamic in nature.

2.2 Object-Oriented Development Methods

Since the introduction of "object-oriented software construction" (OOSC) by Bertrand Meyer in 1988 [5], several object-oriented approaches (OOA) have been developed by computer-aided software engineering (CASE) vendors. The notable object-oriented development methods are: Booch [6], Object Modeling Technique (OMT) [7], Business Object Notation (BON) [8], Class Responsibility Collaborator (CRC) [9], and Fusion. Although these methods use different notations to describe their paradigms, the software development process in each case can be divided into three stages. These three phases are: Object-Oriented Analysis, Object-Oriented Design, and Object-Oriented Implementation. The output from each phase is the basic input for the next. Different problems are addressed in different phases and each phase has its own conceptual models, notations and heuristics.

Object-Oriented Analysis: In this phase, a specification will be produced to document the deliverable of a system under design based on the original requirements. Classes of objects that exist in the system and their relationships to each other are identified. Operations that can be performed on the system and their sequences are produced.

Object-Oriented Design: In the second phase, the specification document is taken as input. It delivers models that show how system operations are implemented by interacting objects and how classes are referenced with respect to each other. During this process, the system operations that have to be implemented by the run-time behaviour of interacting objects, and the appropriate inheritance relationships between classes are defined.

Object-Oriented Implementation: The third phase takes the design document as input and encodes the design in a programming language. The outputs is software code. Inheritance, reference, and class attributes are implemented in programming language classes and object interactions are encoded as methods belonging to a selected class.

Since the Fusion method is developed from the first generation of object-oriented methods, it integrates some of the best aspects from each method. The core features of Fusion show reflections of some features of OMT, Booch and CRC. Therefore the Fusion method has been chosen as a tool to develop the GUI in this research. The Fusion's concept and notation are described hereafter:

2.2.1 Object-Oriented Analysis

The goal of an object-oriented analysis model is to capture as many requirements of a system as possible in a complete, consistent, and unambiguous fashion in the specification. It accomplishes its goal by constructing two models: an object model and an interface model of the system. These models concentrate on describing what a system does rather than how it does.

2.2.1.1 Object Model

The purpose of an object model is to capture the concepts that exist in the domain of the problem and the relationships between them. At the appropriate level of abstraction, almost anything can be considered to be an object and can be identified. In addition to being able to be identified, an object can have one or more values associated with it. These values are called attributes. The attribute values of an object can be changed, but the numbers and names of the attributes of an object are fixed. An object has two fundamental properties. First, it is always possible to distinguish distinct objects even if they have the same attribute values. Second, the identity of an object cannot be changed. Objects are grouped into sets, called classes. A class is an abstraction that represents the idea or general notion of a set of similar objects. Different approaches that are used to classify object are illustrated in Appendix A-2.

2.2.1.2 Interface Model

An interface model defines the input and the output communications of a system. A system is modeled as an active agent that interacts with an environment. An environment is a set of entities which usually are human users, or other hardware or software systems. The basic characteristics of an entity is that it must be active when it communicates with an agent. An event is an instantaneous and atomic unit of communication between an agent and its environment.

An input event to an agent and the effect it causes are called a system operation. A system operation may create a new instance of a class, change the value of an attribute of an existing object, add or delete some roles of objects from a relationship, or send an event to the environment. Only one system operation can be active at any time.

The interface between an agent and its environment is defined by a set of system operations to which the agent can receive input and the set of events that it can output to. There must be at least one schema for each system to describe its operation. The syntax of the various components of a schema is shown in Appendix A-3.

2.2.2 Object-Oriented Design

During the design stage, software structures are introduced to satisfy the abstract definitions produced from analysis. The output of a design phase is an object-oriented software structure that contains the same information as in the object model and the functionality defined in the interface model.

Four models are developed during the design phase.

- Object interaction graphs which describe how objects interact with each other or with the environment at run-time to support the functionality specified in the interface model.
- Visibility graphs which describe the object communication paths.
- Class descriptions which provide a specification of a class interface, data attributes, object reference attributes, and method signatures for all classes in a system.
- Inheritance graphs which describe the class/subclass inheritance structures.

2.2.2.1 Object Interaction Graph

In the design stage, an object interaction graph is constructed to implement each system operation which has been defined in a corresponding analysis stage. An object interaction graph defines the sequences of messages that occur between a collection of objects to realize a particular system operation.

Development of an object interaction graph starts with the identification of the objects involved in a schema. The role of each object in implementing a specified operation must be decided next. The controller in an object interaction graph receives the request to invoke a system operation, and therefore, system operation is a part of the method interface of a controller. Other objects which collaborate and cooperate with the controller to implement the system operation are identified as collaborators. Object interaction graphs are developed for each system operation during the design.

2.2.2.2 Visibility Graphs

In the second stage of the design process, the communication paths between the objects in the system operation are realized by developing visibility graphs. The purpose of a visibility graph is to define the reference structure of classes in a system. Objects must have access to other objects to allow the communication required by the object interaction graphs developed in the first stage of the design process.

A visibility graph is represented as a diagram whose components are client boxes, server boxes, and visibility arrows. A client box represents the class requiring an access and is a rectangular box containing the name of the class. A server box represents the object being accessed and is a rectangular box containing the name of the object and its class. A visibility arrow is a directed arrow from a client class to a server object. The arrow means that the client has access to instances of the server via an access path.

An access from a client to a server may have four types of visibility relationships: reference lifetime, server visibility, server binding, reference mutability.

Reference Lifetime of an access can be either dynamic or permanent. When a client needs to send a message to a server in the context of a single method invocation, the access can be given through a parameter or by a local variable of the method. In this situation, a dynamic reference is specified on the visibility graph. Conversely, when reference to the server object persists between method calls, a permanent reference is required.

Server Visibility can be defined as either exclusive or shared. An exclusive visibility reference guarantees that at the time of any method invocation, only one client will have a reference to the server. Otherwise, the shared visibility of a server is accessed by several clients.

Server Binding indicates a lifetime relationship between a server and a client. When the lifetime of a server is related to the lifetime of a client in such way that if the client is deleted then the server should also be deleted, the lifetimes of the server and the client are bound.

Reference Mutability indicates whether a reference can be reassigned after initialization. If a reference is not assignable after initialization, it has constant mutability.

All representations of class visibility are shown in Figure 2.3.

2.2.2.3 Class Descriptions

After the development of the visibility graphs for all classes, the next step is to collate information from the system object models, object interaction graphs, and visibility graphs into class descriptions, one for each class. During the development of a class description, the methods, data attributes, and object-valued attributes for each class are established.

Each method is introduced by keyword **method** followed by the name of the method and its signature. Methods are derived from object interaction graphs. All object interaction graphs in which the objects of a class participates should be considered together. Messages sent to the objects (i.e., the incoming arrows) are collected together to form the interface of the class.

2.2.2.4 Inheritance Graphs

After the initial class descriptions are produced, the inheritance structures required in the system are designed to position the classes in the inheritance graph.

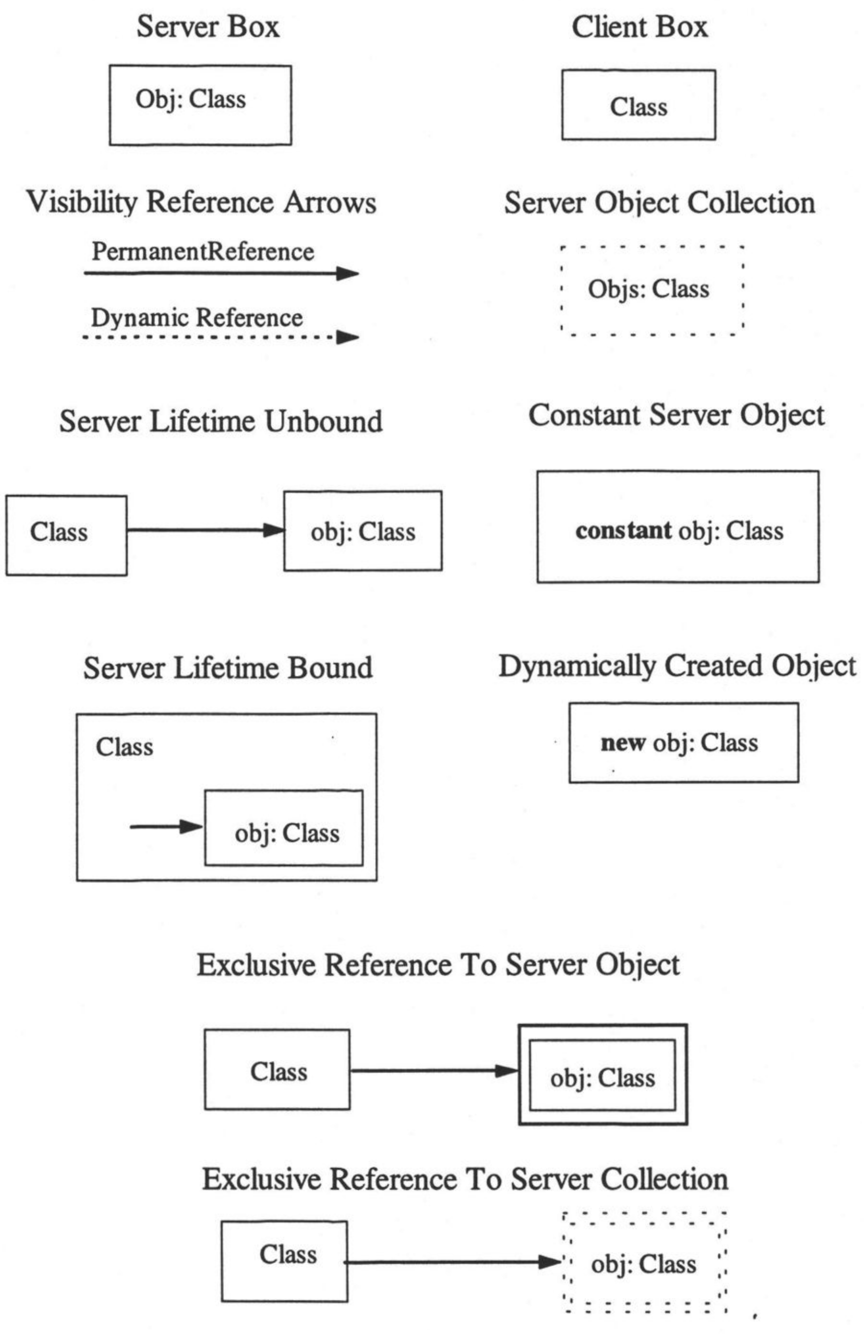


Figure 2.3 Representations of Visibility Graph

2.2.3 Object-Oriented Implementation

In the final stage of an object-oriented approach, the design results are mapped into the soft code by using specific programming language. In this research, the mapping of a class description into softcode has been realized by using C++. The class descriptions generated during the design phase provide most of the class specifications such as attributes, method signatures, and position in inheritance hierarchy.

There are three types of attributes defined in class descriptions and they are, namely, object-value attributes, collection attributes and the plain attributes. Attributes of class descriptions will be class members in C++. An object-valued attribute is usually implemented by using a pointer or a reference type in C++. A collection attribute is usually implemented as a set, bag, list, or a vector and a plain attribute can be implemented in all kinds of data types that C++ permits. Attribute Qualifiers, Mutability, Binding and Sharing, can also be implemented in C++. The "const" attribute of C++ has the same meaning of Mutability. An Unbound object is implemented by a pointer in C++. A Bound object can be implemented by using a class type that embeds the server in the client. A Shared attribute can be freely passed as a parameter, returned as a result, or assigned into other variables in C++.

The methods of class descriptions are implemented as method members of a class and redefinable methods are declared as virtual methods in the base class in C++.

The "is a" clause of class descriptions is implemented by naming the given class followed by the parents names in C++.

2.3 Summary

In this chapter, the object-oriented methodology of the Fusion approach has been introduced. The issues that have been covered so far include the three stages of software engineering. The three stages are:

1. Object-oriented analysis, which produces two models: the object model and the interface model.
2. Object-oriented design, which introduces the object interaction graph, the visibility graph, the class description, and the inheritance graph.
3. Object-oriented implementation, which transfers the result of the design into soft code.

3. APPLICATION OF OBJECT-ORIENTED APPROACH

3.1 Introduction

Power system components can be modelled as objects in an object-oriented approach. The objects representing power system components then can be combined to form an object with a higher hierarchy with the attributes of a section of a power system. The objects representing various sections of a power system can be combined to form a super object with the attributes of an entire power system. The objects and their various combinations can be displayed on a computer screen with the help of a graphical user interface (GUI). A GUI is the centre piece of an object-oriented program. A user can provide input to the system through a GUI and obtain various outputs through the same GUI. A GUI can be used to manipulate the objects representing power system components to investigate various system configurations. By utilizing a GUI, system analyses can be performed on different configurations of a power system in an interactive manner. The graphical displays of system configurations and the interactive nature of an object-oriented approach will make it an effective tool for the training of power system operators.

3. 2 The Draw Function: Requirements

A system operator can draw single-line diagrams of various sections of a power system by utilizing the Draw function of a GUI. In this research, a generic Draw function has been designed. The function has been implemented in C++ programming language.

The proposed Draw function should allow a user to accomplish various tasks through a mouse-driven direct manipulation mode of operation. When a user starts the Draw function, a blank window will appear on the screen. A user can draw the shapes of power system elements such as generators, transformers, circuit breakers, transmission lines and buses on the screen. By drawing, moving and rotating these shapes, a single-line diagram of a power system can be formed which is shown in Figure 3.1. The connections between power system components to form necessary paths from electric generators to electric consumers are shown on single-line diagrams. When a single-line diagram is formed, the user can create a file and save the diagram into a file for future retrieval and modification. All elements which appear on the single-line diagram have their own attributes, and can be accessed through dialogue boxes.

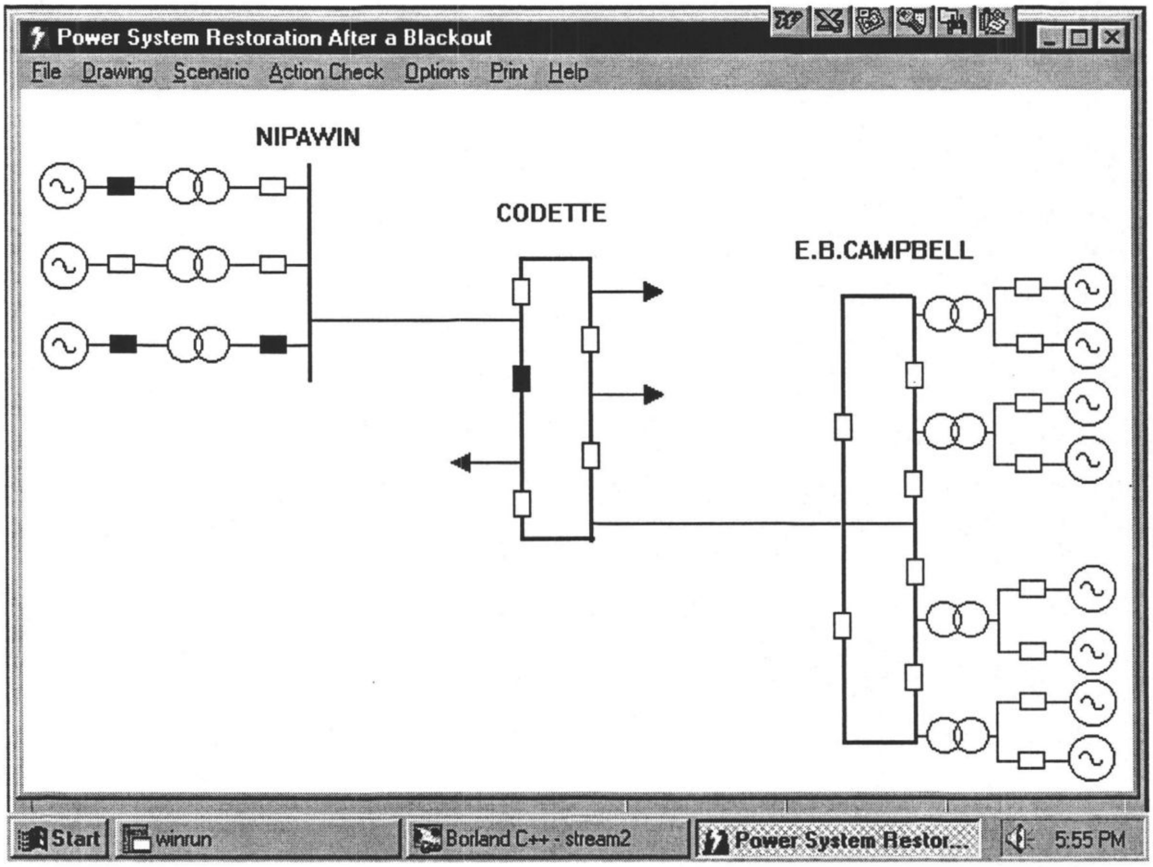

Power System Restoration After a Blackout
File Drawing Scenario Action Check Options Print Help
NIPAWIN
CODETTE
E.B.CAMPBELL
Start
winrun
Borland C++ - stream2
Power System Restor...
5:55 PM

Figure 3.1 A Power System Single-Line Diagram

3.3 The Draw Function: Analysis

In the analysis phase, a precise description of the Draw function is developed. Based on the description of the Draw function, object models and an interface model are developed.

3.3.1 Object Models for the Draw Function

When a user selects the Draw function, a Window is displayed on the screen. The user can select various commands from the menu bar to accomplish various tasks. When the user selects the "Exit" command from the Window, the Draw program is stopped. From this part of the requirement, the three classes and their relationships can be defined in the following manner.

Classes: User, Draw, Window.

Relationships: A user runs Draw; Draw creates a Window. A user selects commands on the Window; and upon selecting "Exit" from the Window, the Window sends a stop message to Draw function. With the information of classes and their relationships, an initial object model has been drawn and is shown in Figure 3.2.

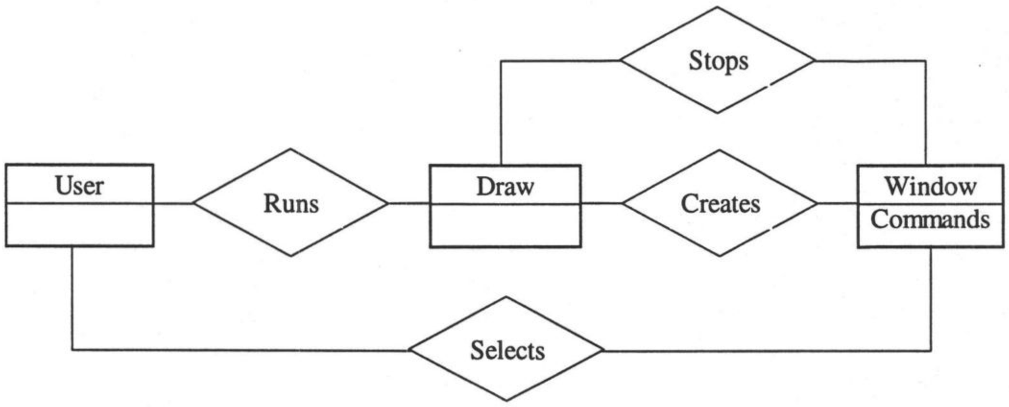


Figure 3.2 Object Model for Draw Function

Several other classes, that interact with the Window class to perform the Draw function, are defined. These classes are GraphicalObject, ObjectBuilder, WindowDC, Point, Pen, Brush, Input and Output. ObjectBuilder class shows temporary changeable shapes of power system elements on the screen. GraphicalObject class draws a permanent shape of elements on the screen. WindowDC class includes essential information about the device context with which a user works on, such as the default foreground and background colors, font, which are used for drawing elements. Point class contains the x and y coordinates of the screen where the elements will be located and drawn. Pen and Brush classes contain the size of a pen and a brush. Input class reads data from a file, and Output class writes data to a file.

When a user uses the Window, the Window creates the ObjectBuilder, the ObjectBuilder records the Point where mouse is located and creates the GraphicalObject. The Input reads data from a file and the output writes data to a file. This information, as shown in Figure 3.3, allows the GUI to draw an object on the Window.

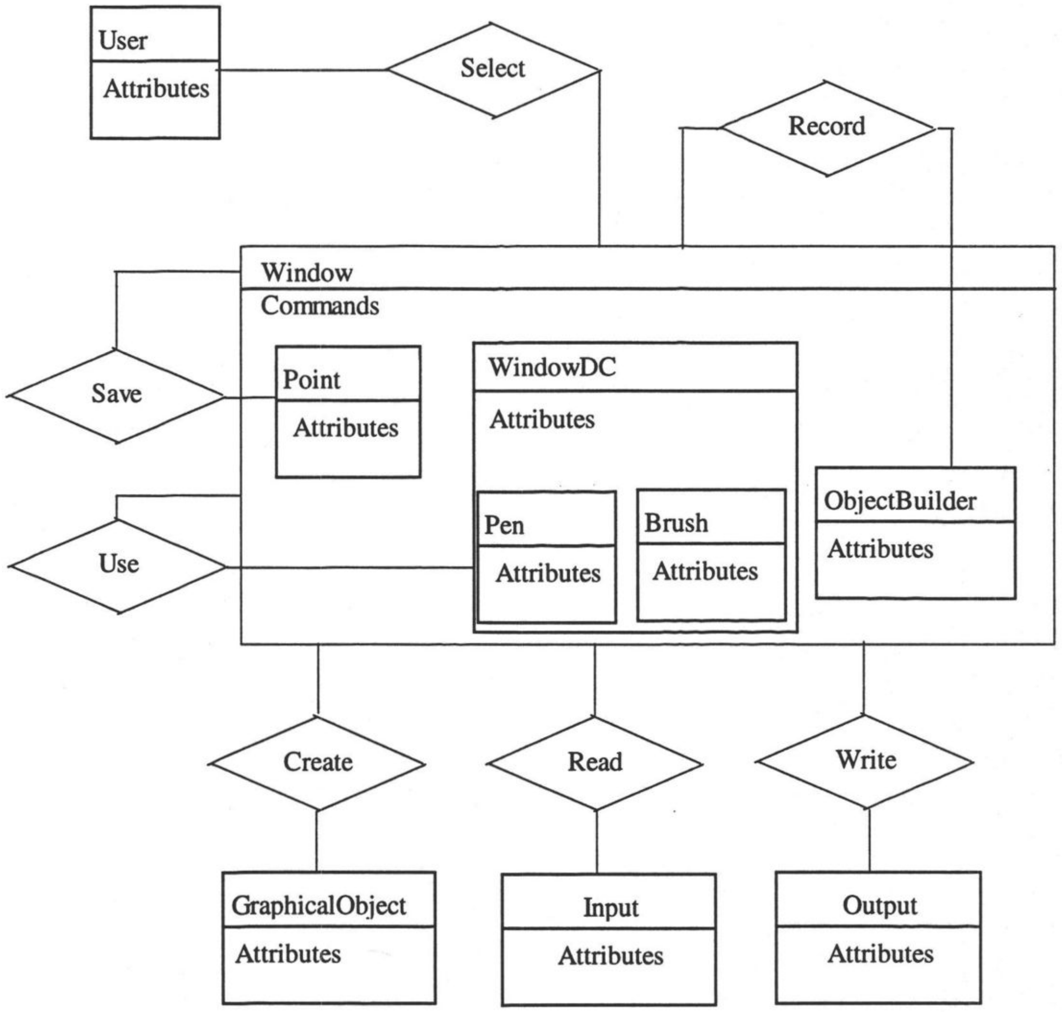


Figure 3.3 Object Model for Window Class

The GraphicalObject and the ObjectBuilder have been defined as superclasses that encapsulate most of the common features related to the graphical display of power system elements. Each of these superclasses is made up of several classes directly related to the modelling and drawing of individual power system elements. The inheritance relationships between ObjectBuilder, GraphicalObject and their children classes are shown in Figures 3.4 and Figure 3.5. Figures 3.2 to 3.5 constitute the object models for the Draw function and its environment.

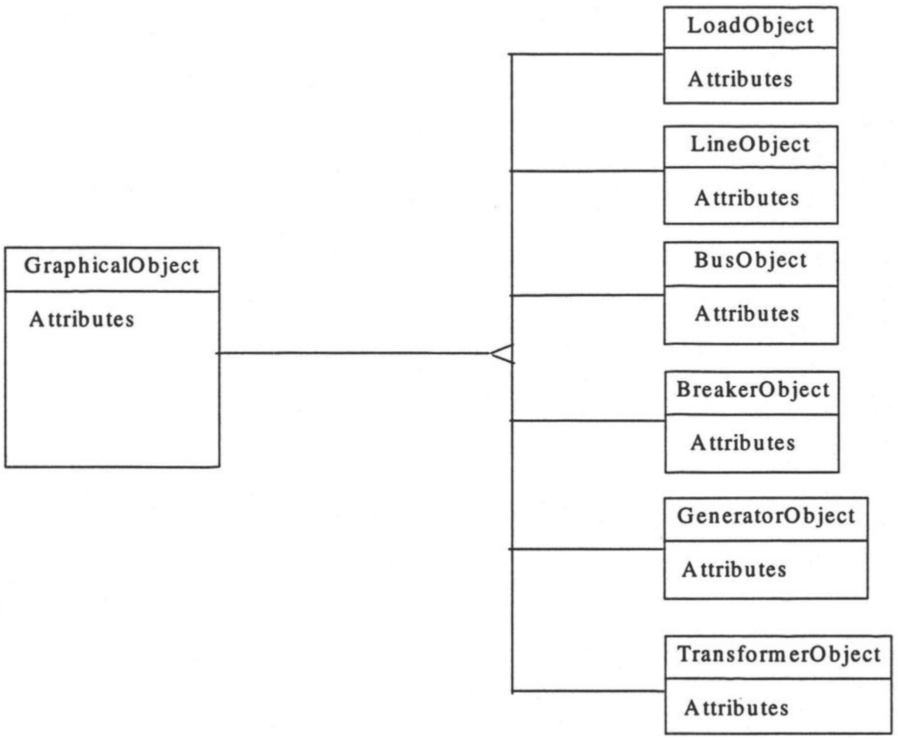


Figure 3.4 Inheritance Relationship for GraphicalObject

3.3.2 Interface Model for the Draw Function

The object model developed up to this point covers the Draw function and its environment. The next step is to determine the boundary between the Draw and its environment. In this case, the system boundary between the Draw and its environment is quite clear. A user interacts with the Draw function, therefore, he/she belongs to the environment.

3.3.3 The Draw Function: Operations

The interactions between the Draw and the environment are accomplished by a set of system operations. These operations are:

- Draw_Breaker
- Draw_Generator
- Draw_Transformer
- Draw_Bus

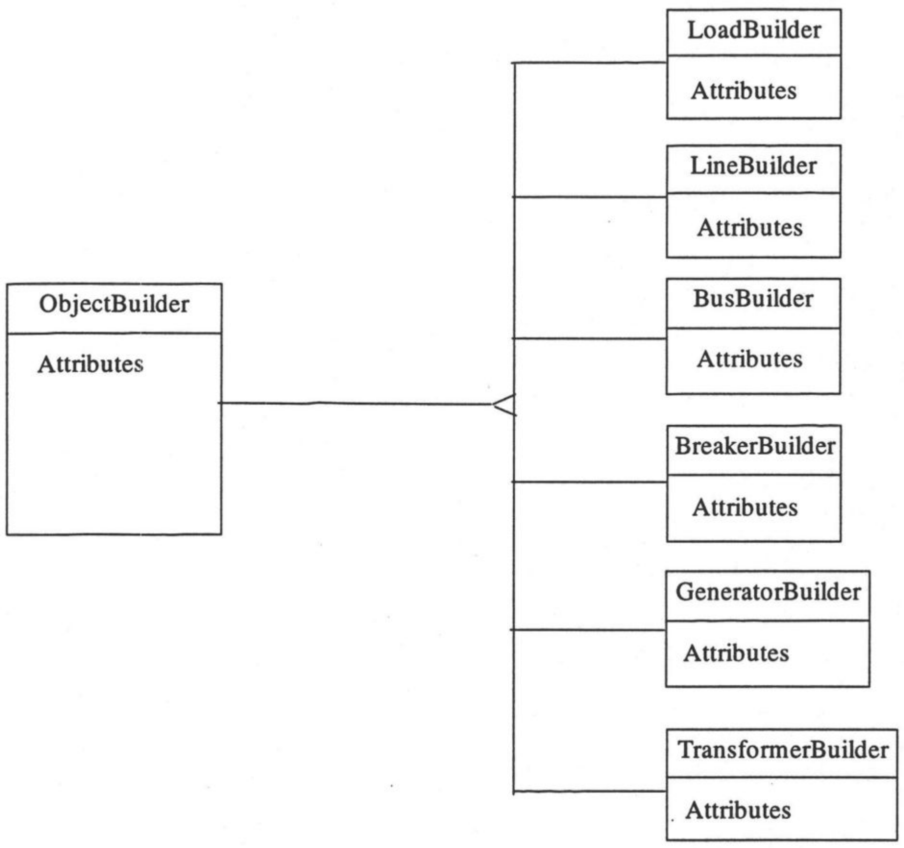


Figure 3.5 Inheritance Relationship for ObjectBuilder Class

- Draw_TransmissonLine
- Draw_Load
- Draw_Text
- Open_File
- Save_File
- LeftButtonDown
- LeftButtonUp
- RightButtonDown
- MouseMove
- Rotate_Shape_Right
- Rotate_Shape_Left
- Rotate_Shape_UpsideDown

The schemata for some of the system operations of the Draw function are described in the following paragraphs. The syntax of the various components of a schema is shown in Appendix A-3.

When a user selects "Load" command from the menu of the Draw function, the Draw_Load operation is activated, which in turn creates the LoadBuilder object and sends a new shape of the cursor to the screen. The clause Changes specifies this creation. Since the operation does not access any values, the Reads clause is empty. When the Loadbuilder is created, the corresponding cursor is used to substitute the arrow cursor and appears on the screen. The Result clause shows this change. Since there is no precondition that has to be matched, the Assumes clause is empty. Figure 3.6 shows the complete schema for the system operation Draw_Load.

Operation: Draw_Load
Description: Draw power system load shape on screen

Reads:
Changes: **new** LoadBuilder
Sends: **new** cursor
Assumes:
Result: Cursor is changed to new shape

Figure 3.6 Schema of Draw_Load

The LeftButtonDown operation is activated when a user clicks and holds the left button of the mouse down. The program anchors the coordinate of the current mouse location if an object of the ObjectBuilder already exists, or finds out the object of the GraphicalObject that contains the current mouse coordinate. The precondition for this system operation is the existence of an object of the ObjectBuilder or an object of the GraphicalObject. If there is no new creation of ObjectBuilder object, then the rotation flag is set. All of the information is presented in the Assumes clause. The system operation extracts the coordinate of the mouse specified in the Reads clause. It is recorded in the Changes clause, whether anchor is set or the current object is found. Since the operation does not send anything to the screen, the Sends clause is empty. The Results of the operation is that either an anchor is set or an existing object is found. Figure 3.7 shows the complete schema for this operation.

Operation:	LeftButtonDown
Description:	Anchor the coordination of the screen if an object of ObjectBuilder exists, otherwise find the object of GraphicalObject which contains the coordination
Reads:	Coordination of mouse
Changes:	ObjectBuilder.SetAnchor or CurrentObject equal to Find.Object
Sends:	
Assumes:	Builder exists, or not
Result:	

Figure 3.7 Schema of LeftButtonDown

The schema of LeftButtonUp system operation is shown in Figure 3.8. When a user releases left button of the mouse, the coordination of the mouse is recorded in Reads clause. The shape of an element is created on the screen if an object of the ObjectBuilder exists. When no object of the ObjectBuilder exists, the system releases the mouse capture and sets the dragging flag to false. The activities are recorded in Changes clause. When a new object of the GraphicalObject is created, repainting flag is set. The Sends clause will send a message to inform the window to repaint itself. The Results clause delivers the shape of the corresponding object to be displayed on screen.

Operation:	LeftButtonUp
Description:	Create the object of GraphicalObject on screen if a builder exists, otherwise release mouse capture.
Reads:	Coordination of mouse
Changes:	**new** graphicalobject
Sends:	Repaints window messsage
Assumes:	
Result:	**new** shape is created, or old shape is moved to new place

Figure 3.8 Schema of LeftButtonUp

The system operation Open_File is quite straightforward: a user selects the "Open" command from the File menu of Window, and inputs the file name to the Window. The operation reads each object from the file into a container which is a collector of the GraphicalObject (for the time being it is assigned the name of the container) and displays all of the objects on screen. The Reads receives a file name, the Changes clause records that a new container is created. The objects of the GraphicalObject are added to the container and a message that the Window needs to be repainted is posted to the Window's paint operation. Objects read from the file are displayed on the screen through Sends clause. Since there is no need for a precondition, the Assumes clause is empty. The objects displayed on screen are recorded in Result. The schema for Open_File is shown in Figure 3.9.

Operation:	Open_File
Description:	Open a file, read each object into container, inform the Window to display objects on screen
Reads:	File Name
Changes:	Create a container, add each object into it
Sends:	Send paint message to the Window
Assumes:	
Result:	

Figure 3.9 Schema of Open_File

Once a power system single-line diagram is created, a user can save it. This activity is recorded in the schema of Figure 3.10. The system operation Save_File is activated by selecting the Save_File command on the Window menu bar. The command Reads the file name and write each object from the container into the file. Since this operation does not send any event to the environment, both Sends and Changes clauses are empty. The Assumes clause is empty too because there is no need for precondition.

Operation:	Save_File
Description:	Open a file, write each object into it
Reads:	Attributes of each object
Changes:	Write each object from container into file
Sends:	
Assumes:	
Result:	

Figure 3.10 Schema of Save_File

MouseMove operation can be activated when a user selects the Draw and then clicks the left button of the mouse or when a user points the cursor at one of the elements on single-line diagram and then clicks and holds the mouse. In the Reads clause the mouse coordination is recorded, and the Offset from the current position to original position is calculated. The system Changes the position of the element, and Sends repainting message to the window object to update the location of the element. The assumption that the left button of the mouse has to be pushed down is recorded in Assumes. The result that either the size of the shape is changed or the shape is relocated, is recorded in Results. The schema of this operation is shown in Figure 3.11.

Operation:	MouseMove
Description:	Show temporary shape of the object on screen if a builder exists, or moves an existed object on the screen.
Reads:	Mouse coordination
Changes:	Offset of the shape position or set drag
Sends:	Set repaint window message
Assumes:	
Result:	

Figure 3.11 Schema of MouseMove

3.4 The Draw Function: Design

In the design phase of the Draw function, the abstract definitions which have been developed in the analysis phase are converted into object-oriented software structures. Following models have to be developed during the design phase for the Draw function:

- Object interaction graphs,
- Visibility graphs,
- Class descriptions for each class, and
- Inheritance graphs.

3.4.1 Object Interaction Graphs

An object interaction graph shows what objects are involved in the computation and defines how they collaborate in response to an operation. The first step is to identify the objects that are involved in the realization of an operation. For example, in the operation Draw_Load, the objects involved are a window object, a builder object and a point object. In the second step, the role of each object in implementing the operation is decided and an object is identified to play the role of a controller. In the case of the Draw_Load, the window object is the controller, since it receives the messages related to the operation. The third step is to decide how the functionality of the operation is distributed among the various objects involved. In the final step, the distribution of the functionality of the system operation is recorded in an object interaction graph. The object interaction graph for system operation Draw_Load is shown in Figure 3.12. The syntax of an object interaction graph is shown in Appendix A-4.

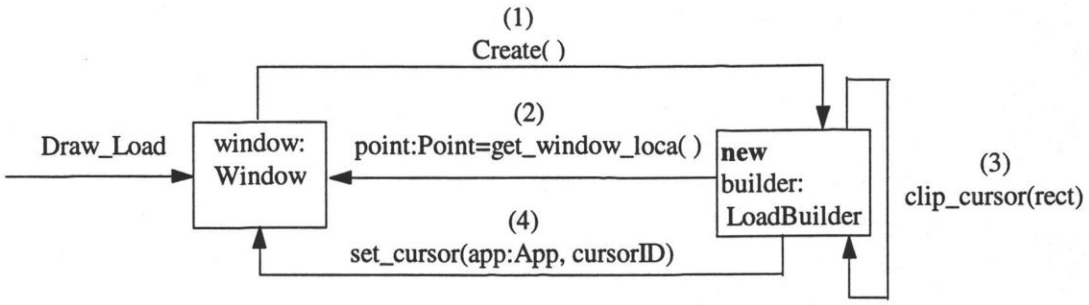


Figure 3.12 Object Interaction Graph for Draw_Load

In the operation LeftButtonDown, the window object is the controller. It collaborates with builder and container objects to execute the operation. When the message of LeftButtonDown is sent to the window, the window considers two situations. In the case of the existence of a builder, it passes the current position of the mouse to the builder and asks the builder to anchor the position. Then the builder creates a new object called dc from BuilderDC class to initialize the device context for settings of color, pen and brush. In the case of the absence a builder, the window inquires the container to iterate until the right object containing the mouse position is found. This object interaction is shown in Figure 3.13.

The object interaction for the operation LeftButtonUp is shown in Figure 3.14. When this operation is invoked, the window object checks the current state of the Draw function. If no builder object exists, the window sends itself a message to release mouse capture. Otherwise, the window sends a message to the builder to create a new object of the GraphicalObject. Then the window object sends a message to add this newly-created object into the container. After the task is done, the window object sends itself a repainting message to refresh the window appearance on screen with the newly-created object.

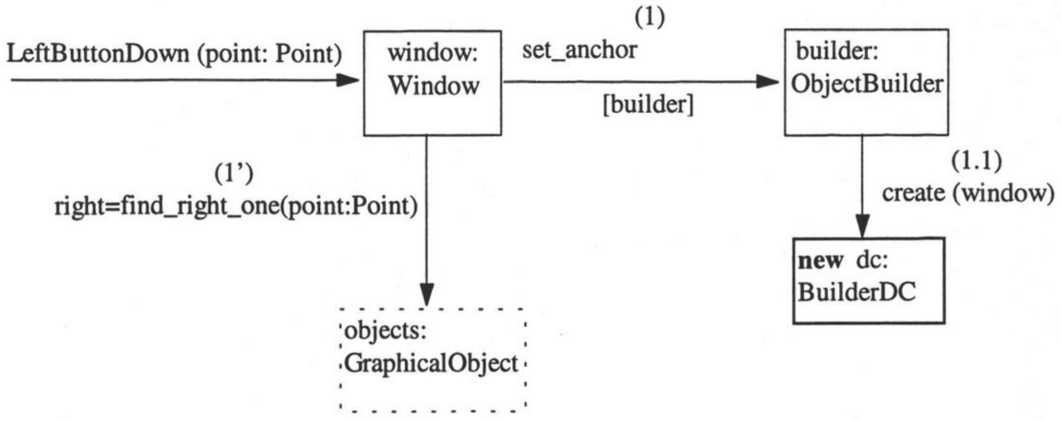


Figure 3.13 Object Interaction Graph for LeftButtonDown

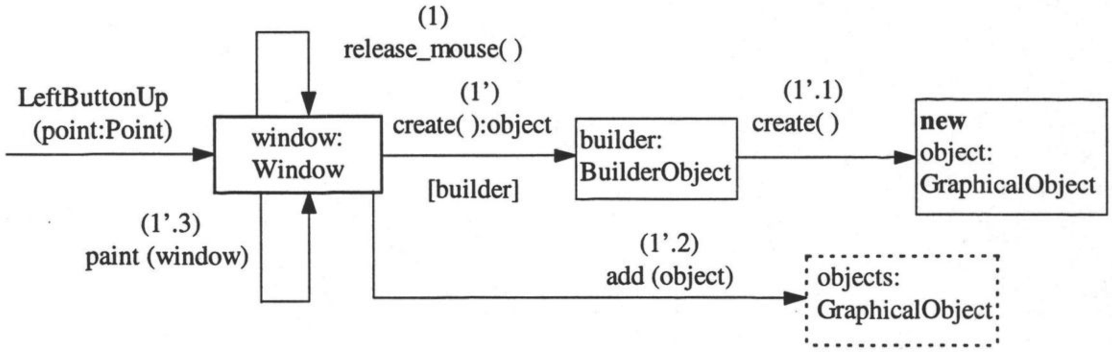


Figure 3.14 Object Interaction Graph for LeftButtonUp

When the window object receives the invocation of Open_File operation, it checks the system state. If there is an open file, it innovates Save_File message to save that file first. After that file has been saved, the window object sends a message to a dialogue object to get a file name from the user and creates an object called in from the Input class. It sends in object a message to open the file, and read data into an GraphicalObject, and then the window sends a message to the container to add that GraphicalObject into it. Finally, the window sends itself a repaint message to refresh the screen. The object interaction graph for this operation is shown in Figure 3.15.

In Save_File operation, several objects are involved: a window object which acts as a controller, a dialogue object from the Dialog class, an out object from the Output class and a container of the GraphicalObject. When a window object receives a Save_File message, it sends a message to the dialogue, asks it to get the file name from a user. The window object will either create or open a file depending upon the existence of the file. After this task is done, the window sends a message to the container, and asks it to write all current GraphicalObjects from the container into the file. The object interaction graph is shown in Figure 3.16.

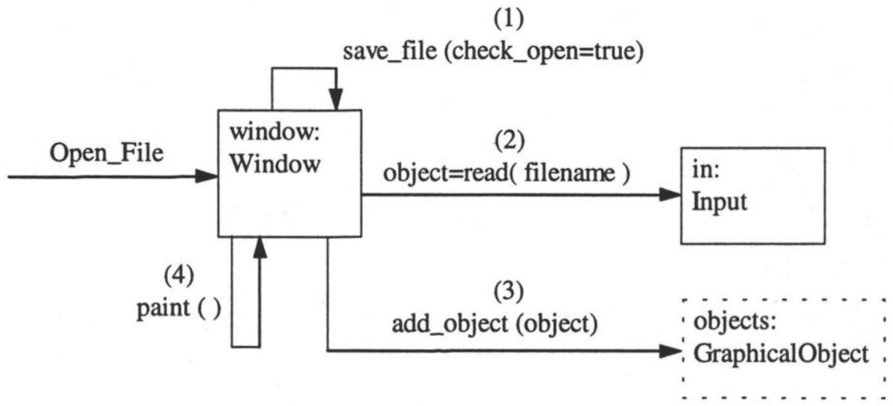


Figure 3.15 Object Interaction Graph for Open_File

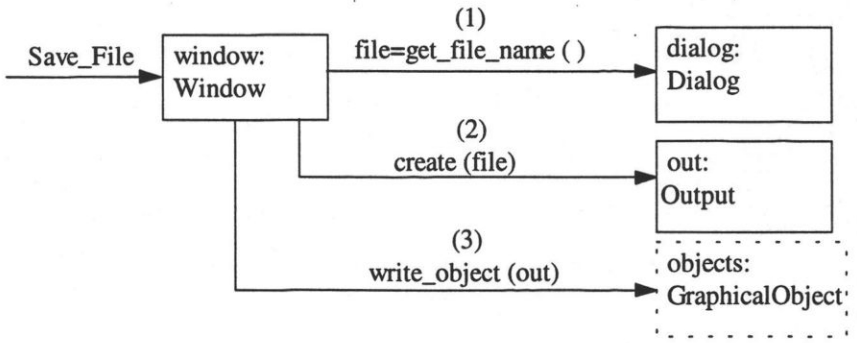


Figure 3.16 Object Interaction Graph for Save_File

3.4.2 Visibility Graphs

A visibility graph shows the communication paths between objects necessary to realize programming operations. A visibility graph can be used to identify the objects that have to be referenced to perform a given operation. The same graph can also be used to identify the type of those references. The development of a visibility graph for the Window class is discussed in the following paragraphs.

Depending upon its lifetime, a reference to an object can either be of permanent or dynamic type. If a client needs to send a message to a server in the context of a singular method invocation, access can be given through a parameter or by a local variable of the method. A dynamic reference will be specified on the visibility graph to represent the message passing. The Window sends messages to the ObjectBuilder in several object interaction graphs. This reference is, therefore, useful in several contexts, and should be of permanent type. It is the same for the Container, the WindowDC and the Point. As for the GraphicalObject, it only acts as a parameter of one method, and therefore, is defined as a dynamic reference. The Input, the Output and the Dialogue classes are used once as local variables, and therefore, are shown as dynamic references. Since the WindowDC class defines the device context for the Window itself, it should be a permanent reference.

In the object interaction graphs, the ObjectBuilder, the Container, the WindowDC receive messages only from the Window, and thus each of these references is exclusive to the Window. Since the GraphicalObject, the Dialogue, the Point, the Input, and the Output may receive messages from classes other than the Window, they are shown as shared references.

The next step is to consider the binding of the server objects. The Container and the WindowDC have the same lifetimes as that of the Window, therefore, they are bound to the Window. The Input, the Output, the ObjectBuilder, the Point, and the GraphicalObject objects have lifetimes that are separated from that of the Window, and therefore, are unbound and appear outside the Window.

Finally, the mutability of the references to the server are considered. Since the above mentioned references are reassignable after initialization, they are not constant. The visibility graph for the Window class is shown in Figure 3.17.

3.4.3 Class Descriptions

Class descriptions specify the internal state and external interface of a class. The development of the class description for the Window is illustrated in this section. Class descriptions for other classes can be derived in similar ways. From the object interaction graphs which involve the Window, the following methods of the Window are yielded:

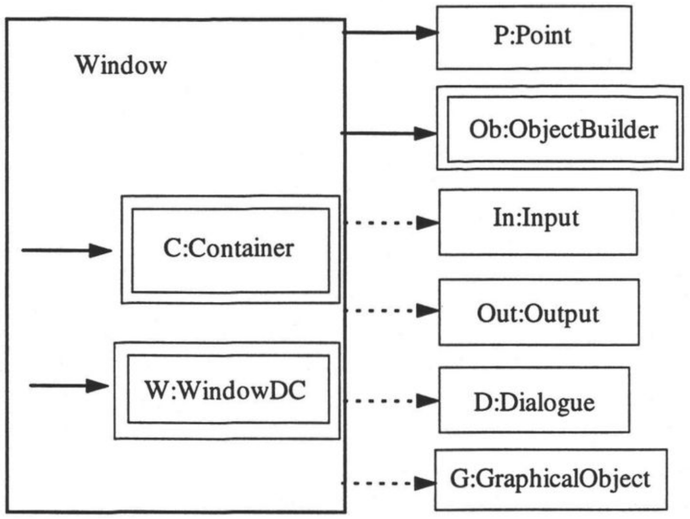


Figure 3.17 Visibility Graph for Window

- LeftButtonDown ( )
- LeftButtonUp ( )
- MouseMove ( )
- RotateShapeRight ( )
- RotateShapeLeft ( )
- RotateShapeUpsidedown ( )
- Draw_Load ( )
- Draw_Generator ( )
- Draw_Transformer ( )
- Draw_Breaker ( )
- Draw_Text ( )

There are four permanent references in the visibility graph of the Window and they have been defined as object-valued attributes in the class description of the Window, the Container, the WindowDC, the ObjectBuilder and the Point. There are four dynamic references in the visibility graph of the Window and they become local variables in some of the Window methods. With this information, the initial Window class description has been completed as shown in Figure 3.18.

```
Class Window
        attribute windowDC: exclusive bound WindowDC
        attribute container: exclusive bound Container
        attribute objectbuilder: exclusive ObjectBuilder
        attribute graphicalobject: GraphicalObject
        attribute point: bound Point
        method draw_load ( )
        method draw_generator ( )
        method draw_ tranformer ( )
        method draw_ breaker ( )
        method draw_ tranmission line ( )
        method rotate_shape_right ( )
        method rotate_shape_left ( )
        method rotate_shape_upsidedown ( )
        method left_button_down ( )
        method left_button_up ( )
        method mouse_move ( )
        method open_file ( )
        method save_file ( )
endclass
```

Figure 3.18 Class Description for Window

3.4.4 Inheritance Graphs and Updating Class Descriptions

The final step in a design process is to build an inheritance graph for the classes identified by the object model and the object interaction graphs, and documented in the class descriptions. Since the Draw function in this research has been developed based on the Borland object window library (OWL) environment [10], several classes that belong to OWL have been utilized in developing the Draw. These classes are explained below:

- TApplication acts as an object-oriented stand-in for an application module. It supplies the basic behaviour required of an application, and its member functions create instances of a class, create main window, and process messages. The Draw class is derived from the TApplication.
- TWindow provides window-specific behaviour and encapsulates many functions that control window behaviour and specify window creation and registration attributes. It has been used as the base class for Window class.
- TStreamableBase supplies the basic functionality of a streamable object. Objects that are created when an application runs are temporary objects. They are constructed, used and destroyed as the application proceeds. Objects can appear and

disappear as they enter and leave their scope, or when the program terminates. By making objects streamable, they can be saved either in memory or file, so that they persist beyond their normal lifespan. TStreamableBase provides the function of saving the objects permanently on a disk using file streams. They can then be read back and restored by the same application, or by other instances of the same application, or by other application. Since each created object of GraphicalObject needs to be stored as a persistent object for later use, the GraphicalObject is derived from TStreamableBase.

- TClientDC provides access to the client area owned by a window and the WindowDC, the BuilderDC and the GraphDC are derived from it.
- TDialog encapsulates the behaviour of dialogue boxes, it supports the initialization, creation, and execution of all types of dialogue boxes. The Dialogue class used in the GUI is derived from TDialog.
- TDC: When working with the Window™ Graphical Development Interface (GDI), a device context has to be used to access all devices, from windows, printers and plotters. The device context is a structure maintained by GDI that contains essential information about the devices with which a user is working, such as the default foreground and background colors, font, palette, and so on. TDC class contains most of the device-context functionality and the TClientDC is derived from it..
- TRect encapsulates the properties of rectangles with sides parallel to the x- and y-axes. In ObjectWindows, these rectangles define the boundaries of windows, boxes, and clipping regions. It contains four data members which represent the top left and bottom right (x, y) coordinates of the rectangle.
- TPoint encapsulates the notion of a two-dimensional point that represents the coordinates x and y on the screen. TPoint has two data members,. Member functions and operators are provided for comparing, assigning, and manipulating the coordinates. The Point class is derived from it.
- TPen encapsulates a logical pen. It contains a color for the pen's "ink", the pen width, and the pen style and the Pen class is derived from it.

- TBrush encapsulates a logical brush. It contains a color for the ink, a brush width, the brush style, the pattern and the Brush class is derived from it.
- ipstream is an input stream class for reading streamable objects and the Input class is derived from it.
- opstream is an output stream class for writing streamable objects and the Output class is derived from it.

The ObjectBuilder class is a combination of the following subclasses: TransmissionLineBuilder, BreakerBuilder, GeneratorBuilder, TransformerBuilder LoadBuilder, and BusBuilder. Although these subclasses overwrite some virtual functions of the ObjectBuilder, they have no functionality and attributes beyond the ObjectBuilder and therefore, their class descriptions can be dispensed with. The subclasses derived from the GraphicalObject class are: LoadObject, LineObject, BusObject, BreakerObject, GeneratorObject and TransformerObject. These classes need more attribute values to describe their characters and connections with other elements.

3.5 The Draw Function: Implementation

The models developed in the analysis and the design phases have been implemented in C++. Some of the C++ features that have been utilized in the implementation are:

- inheritance, which is realized by parent and children classes relationships;
- encapsulation, which is realized by making a class member private, protected, and public; and
- polymorphism, which is realized by using a single name to denote objects of many different classes that are related by common superclass. Any object denoted by this name is able to respond to some common set of operation.

3.5.1 Implementation of Class Descriptions

The information needed for the implementation of the Window, the ObjectBuilder and the GraphicalObject has been collated in their respective class description. The implementation stage is essentially the translation of the class description into a programming language.

The attributes of different classes in this research have been translated into C++ programming language. The translation of the Window class is illustrated in the following paragraphs. All other objects and classes have been translated in a similar fashion.

The Window class has five object-valued attributes: the WindowDC, the GraphicalObjects, the Point, the ObjectBuilder and the Container. The WindowDC and Container are bound to the Window class in an exclusive manner, therefore, these attributes are embedded in the Window instances not as pointers. The Point is bound to the Window class and is embedded to the Window class despite being shared. In contrast, the attributes of the ObjectBuilder and the GraphicalObject are not bound to the Window, they become C++ pointers. All these attributes are made private, since there is no derivation of the Window. The next step is to develop the declarations for the methods of the Window, some of them have parameters and return values. Since most of them interact with the environment, they have been defined as public function. The resulting C++ codes for the Window class are shown in Figure 3. 19.

3.5.2 Implementation of Method Bodies

The implementation of two member functions of the Window, the LeftButtonDown and the LeftButtonUp, will be illustrated in this section. Other member functions can be implemented in a similar fashion.

The LeftButtonDown is a virtual function of TWindow and it must be overwritten in the Window class. In this function, the iteration through all objects stored in the container object is implemented by using one of the iterate templates of OWL that matches with the template applied for the container in the class declaration as local variables of the function. The code for the LeftButtonDown method is shown in Figure 3. 20.

The LeftButtonUp operation can be implemented by overwriting the LeftButtonUp function of the TWindow. This operation has no parameters, because all the objects it sends messages to are attributes of the Window. Since the lifetime of the objects created from the ObjectBuilder end after this operation, the objects have to be deleted from memory. The codes for the LeftButtonUp are shown in Figure 3.21.

```
class Window: public Twindow {
        private:
                TPoint mouseCoordinate;
                GraphicalObject *rightObject;
                ObjectBuilder *Builder;
                TISetAs Vector<GraphicalObject> Object;
                WindowDC thisDC;

                void AddObject (GraphicalObject *obj );
        public:
                Window ( );
                void CmBus ( );
                void CmLoad ( );
                void CmLine ( );
                void CmBreaker ( );
                void CmGenerator ( );
                void CmTransformer ( );
                void CmText ( );
                void CmRotateRight ( );
                void CmRotateLeft ( );
                void CmRotateUpsideDown ( );
                void EvLButtonDown ( TPoint & point );
                void EvLButtonUp ( TPoint & point );
                void EvMouseMove (TPoint & point );
};
```

Figure 3. 19 C++ Window Class in C++

3.6 Summary

The development of the Draw function has been illustrated in this chapter. Fusion method has been applied to develop the Draw function. The user interaction with the GUI has been implemented based on a mouse-driven direct manipulation mode of operation. The GUI has been designed to conform with the current GUI trends based on how mouse actions are translated into display effects. Uniformity throughout the GUI has been achieved by the use of a small number of mouse manipulations that trigger the same class of response for each object type. The applicable mouse actions are select, drag and double-click. The select operation is accomplished by clicking the left mouse button with the cursor on the desired object. Once the desired objects are selected, the menus or further mouse actions may be used to alter the object's attributes. The drag action is performed by holding down the left mouse button while moving the mouse. It is used to move selected objects or to connect objects on the display. One of interfaces shown in Figure 3.22 is achieved by a set of mouse manipulations. The double-click action is used to pop-up a breaker's dialogue boxes to display and alter the breaker's attributes. A pop-up attribute box for the breaker object shows the breaker's name and allows the user to open or close the breaker and set its normal condition. From the dialog box, another dialog box can be displayed which displays the breaker's connection in the single-line diagram.

```
Void Window :: EvLButtonDown ( TPoint & point ) {
        if ( builder == 0 )
         { TISetAsVectorIterator<GraphicalObject> Find (Objects);
           { while (Find ! = 0 )
             { if ( Find.Current ( ) -> box.Contains (point ))
                 {
                   { if ( Rotate != 0 )
                      {  Find.Current ( ) -> RotateShape ( Rotate );
                         Invalidata ( );
                         Rotate = 0; }
                     else
                      {  mouseCoord = point;
                         dragging = TRUE;
                         SetCapture ( );
                         rightObject = Find.Current ( );
                       }
                   }
                   Find ++;
                 }
               }
             }
        else
                 Builder-> SetAnchor ( point );
  }
```

Figure 3.20 LeftButtonDown method in C++

```
Void Window : : EvLButtonUp (TPoint & point ) {

        if ( Builder == 0 )
         { if (dragging )
            {     dragging = FALSE;
                  ReleaseCapture ( ) ;
                  rightObject = 0;
            }
         }
        else
          {
                  AddObject (Builder-> CreateObject ( ) );
                  delete  Builder;
                  Builder = 0;
                  Invalidate ( );
            }
}
```

Figure 3.21 EvLButtonUp Method in C++

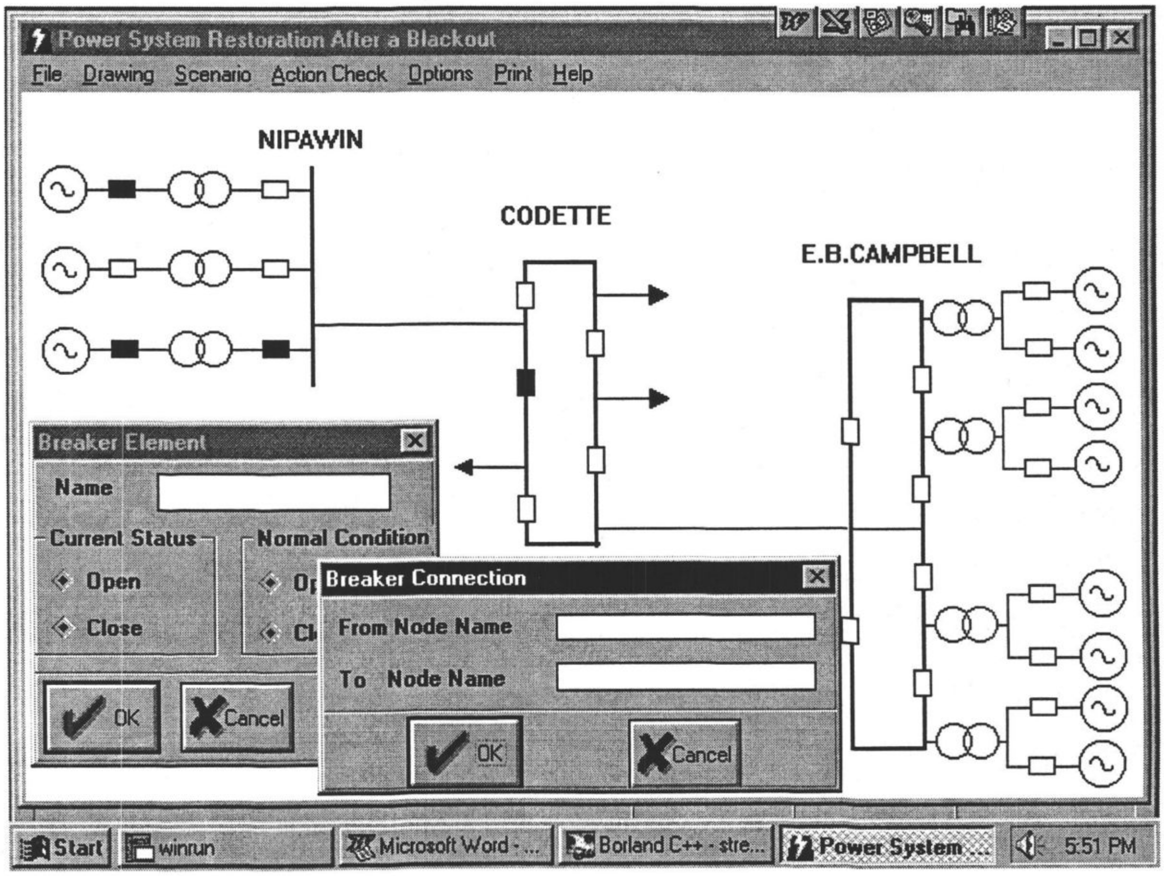


Figure 3.22 An Interace of the GUI

4. CASE-BASED REASONING

4.1 Introduction

Case-based reasoning (CBR) is a technique that allows a system to reason from past experience rather than from first principles. CBR is an attempt to model human beings' ability to reason from experience. When people encounter situations in everyday life, they tend to relate it to their previous experiences. Each new experience is processed and understood in terms of things that have happened before. This makes the task of handling a new situation much simpler than if no prior experience is used.

Human experts in most fields seem to be able to relate their knowledge in terms of experiences they have had. Designers of traditional rule-based expert systems, for example, must try to distill general recurrent experience from these various experiences. CBR attempts to reason directly from the experiences themselves, rather than from the rules derived from the experiences.

The basic idea of CBR is to retain a library of experiences, or "cases", and to use them in future reasoning. A CBR reasoner examines the current situation or problem and extracts its main features. When required, it searches its library of cases and retrieves, via some appropriate indexing scheme, the relevant past experience. The system then adapts the previous experiences to form a solution to the problem at hand. After suggesting a solution, the reasoner criticizes the solution. If more changes are necessary, it adapts the solution further and criticizes it again. This is repeated until the reasoner is satisfied with the solution. It then applies the solution in the domain for which the system is designed and evaluates the results.

A typical CBR reasoner consists of several components: a case-retrieval component, a solution proposal component, solution adaptation and criticism components, a solution application component, an evaluation component and a storage/generalization component. A typical CBR process is illustrated in Figure 4.1.

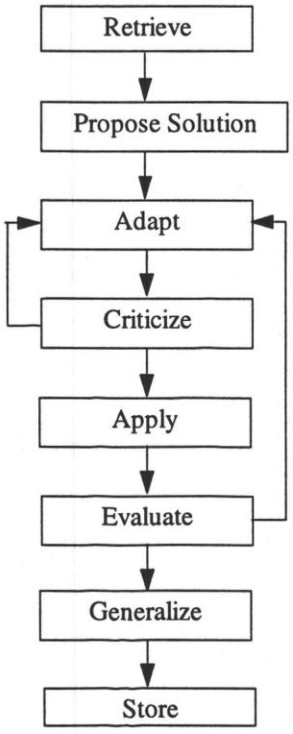


Figure 4.1 The CBR Process

A detailed discussion of each of the components in a CBR system and various approaches to some of the problems will be discussed in the following sections.

4.2 Case Representation

A case is a contextualized piece of knowledge which represents an experience and situation which is related to that experience. Case representation involves determining what to store in a case and how to store it efficiently. Kambhampati [11] suggests that the important issues in case representation are functionality and ease of acquisition; that is to say, the components of the case should be useful to the reasoning system and they should be easy to derive from the current situation.

A case usually has three component parts: a situation or problem description, a solution, and an outcome which sometimes may be omitted. The situation description records the state of the world at the time the case was taking place and the problems that needed to be solved at that time. The solution part records the solution to the problem specified in the problem description. The outcome records the resulting state of the world after the solution was carried out.

The components of a problem description are: goals to be achieved in solving the problem, constraints on those goals which are conditions, and features of the problem situation relevant to achieving the goals.

The solution is the concepts or objects that achieve the goals which are set in the problem description, taking into account the specified constraints and other contextual features. The solution to a design problem is the artifact that was designed. The solution to a planning problem is the derived operation sequence.

The outcome of a case specifies what happened as a result of carrying out the solution or how the solution performed. Outcome includes both feedback from the world and interpretation of that feedback.

There are several representational formalisms (e.g., frames, semantic networks, predicate notation, rules) that have been used to represent the contents of a case. CASEY [12] is a system of diagnosing heart problems. A case in CASEY is a frame with three major slots in it: a slot describing the problem, a slot giving the solution, and a slot giving justification for the solution. MEDIATOR [13], which has been used to solve everyday resource disputes in a commonsense way, represents its case in a highly structured frame. CHEF [14] also uses a frame representation to organize case information. JULIA [15], a design problem solver, uses frames to structure its case and divides the slots in frames into units according to their functional roles. REMIND [16] facilitates formlike representations in building case-based decision aiding systems. The formlike representation is a relatively simple structure as the structure is not embedded, and all representations are flat.

4.3 Case Retrieval

Case retrieval is the process of retrieving a case or a set of cases from memory. In general, it consists of two substeps: recalling previous cases and selecting the best subset.

1. **Recalling previous cases**. The goal of this step is to retrieve "good" cases that can support the reasoning that comes in the next steps. Good cases are those that have the potential to make relevant predications about the new case. Retrieval is done by using features of the new case as indexes into the case library. Cases, labeled by subsets of those features or by features that can be derived from those features, are recalled.
2. **Selecting the best subset.** This step selects the most promising case or cases to reason with from those generated in step 1. The purpose of this step is to narrow down the set of relevant cases to a few most-on-point candidates worthy of intensive consideration.

A retrieval algorithm relies very heavily on the situation assessment and proper indexing of cases. Based on the assessment and indexing of various cases, a case library is built. A suitable algorithm has to be developed to retrieve appropriate cases.

4.3.1 Indexing Scheme

Indexing scheme is the methodology of assigning labels to cases at the time when they are entered into the case library to ensure that they can be retrieved at appropriate times. Most of the significant literature [15-20] suggest that the indexing scheme is one of the most important components in the design of a case-based reasoned. The ability to access a few of the cases which are most relevant to the situation at hand in an efficient manner is almost entirely dependent on the effectiveness of the indexing scheme.

The most important question in the design of an indexing scheme is what constitutes a good index. Barletta & Mark [15] and Ashley [17] suggest the use of a domain theory to predict the important features of a case. Charles Martin [18,19] suggests the use of indices to incorporate the most important features of the current case. Others suggest that the current goals of the reasoner are more important than any of the details of the case [20].

4.3.2 Situation Assessment

The goal of a situation assessment is to extract the important features of a situation. These features will be used to retrieve cases from the case library. It is, therefore, important to select the features that can be used as indices into the case library. It is very closely related to the selection of an appropriate indexing scheme.

4.3.3 Organizational Structures and Retrieval

A case library is a special kind of database. It stores a number of cases and its retrieval algorithms must be able to find the appropriate case when queried. Because no case in the case library can ever be expected to match a new situation exactly, a search would usually result in the retrieval of a case with close partial match. The algorithm used to search a database will not work for searching a case library, since database algorithms require the fields of a query to match items in the database exactly or to be instances of the types specified in the query.

Algorithms for searching a case library are associated with a different organizational structure. Kolodner [21] suggests storing cases in a hierarchically-organized structure and using a parallel concept refinement retrieval algorithm to retrieve cases. Domeshek [22] suggests generating indices and retrieving cases in parallel processing. Another way to improve the efficiency is to simplify the retrieval process.

Some systems use a directed network or a tree form to store cases [23]. These approaches tend to be hierarchical in nature and there are many algorithms available for traversing networks in an efficient manner. As well, network traversal algorithms are easily adapted for parallel processing. The main disadvantage of a hierarchical structure is that the type of indexing and retrieval are constrained to the current structure of the network. As well, retrieval can be very inefficient in the worst case when the desired cases are at the leaf level of a directed network.

Schank [24] proposes a theory of dynamic memory which uses various structures to process and store experiences. These structures include scripts, scenes and memory organization packets (MOPs). These structures are hierarchical and each has its own specialized purpose.

An important aspect of a retrieval algorithm is that it must be fast. Ashley [25] suggests a fast, multi-step retrieval based on simple features after making reasonable assumptions to simplify the comparison of cases. The information obtained from a retrieval is added to the search criteria and another retrieval is performed. Further refinements are performed until a suitable number of cases have been selected. Genter [26] discusses the tradeoffs between doing deep and shallow reminding. Shallow reminding involves the use of the surface features of a case for retrieval. This allows fast retrieval, but the retrieved cases may not be very useful to the reasoner. Deep reminding involves the use of the complex features of a case, such as underlying themes or morals, for retrieval. A case retrieved in such a way will be very relevant to the current situation. However, deep reminding is computationally expensive. Another approach suggests the use of qualitative criteria or multiple attributes to compare cases [27].

4.4 Proposing a Solution

In the next step, relevant portions of the cases selected during retrieval are extracted to form a draft solution to the new case. This process normally involves selecting a solution from the old problem, or some pieces of it, as a draft solution to the new one according to the interpretations or solutions they predict.

4.5 Adaptation

The draft solutions are then used as inspiration for solving new problems. Because the situations at hand rarely match old ones exactly, adaptation is used to combine and modify the draft to derive a solution to the current problem. In a planning application, CHEF shows how to ensure that all of the goals of a plan are satisfied, how to reorder plan steps so that they do not interfere with each other and how to add steps while meeting all of the preconditions for each step. Hinrichs [28] suggests several requirements for an adaptation component. Adaptation should support learning by allowing the inferences made during the adaptation to be reused later. One approach of adaptation is to store the required information in the case itself. Turner [29] uses flexible plan preconditions to determine when a case should be adapted. If a normal precondition is violated, the plan is discarded; if a flexible precondition is violated, the reasoner will use heuristics stored with the precondition to adapt the case in such a way as to avoid the violation. Hendler & Kambhampati [30] use annotations on their plans to determine when a plan can be reused.

Another approach is to retrieve cases from memory which will help to avoid the conflicts or to aid in performing the adaptation. Barletta & Hennessy [31] use pieces of other cases to avoid conflicts with the current case. By combining pieces of various cases, it is possible to exploit the strengths of each case while avoiding the weaknesses. Sycara [32] uses other cases to guide the adaptation process: a case library is searched for substitutions, generalizations and specifications of parts of the current case which will avoid the conflicts of the case. Adaptations which worked well in the past in similar situations are reused while those which did not work are avoided.

Collins [33] uses a set of domain-independent transformation rules to perform adaptations. PLEXUS [34] uses heuristics for adapting plans. These approaches tend to be static and inflexible. A better approach would perform adaptation dynamically and learn which adaptations are useful.

4.6 Criticism and Justification

In this stage, a solution is justified before being tried out in the real world. If all knowledge necessary for evaluation is known, it can be referred as a validation step. The process of criticizing solutions can be achieved by comparing and contrasting the proposed solution with other similar solutions. Another way to criticize a solution is to run a simulation and check the results.

The goal of the criticism component is to suggest partitions of the solution that the adapter should be concerned with. Although Rissland & Ashley [35] attempt to symbolically compare features in order to determine the cause of failures. CHEF contains an ASSIGNER component which is used to assign the blame to the part of the plan which caused the problem. Very little work has been done on the criticism component.

4.7 Solution Application

The application of a CBR solution in a domain is the responsibility of the system making the use of the CBR reasoner. The main concern of a CBR system in the application phase is the effective communication with the rest of the system. As an example, JULIA uses a blackboard system to communicate with a constraint satisfier, a problem solver, a constraint propagator and a conversational controller.

4.8 Evaluation

In the stage of evaluation, the results of the reasoning are tried out in the real world. Feedback from what happened during the execution of a solution is obtained and analyzed by evaluation which judges the goodness of a proposed solution. Evaluation usually includes explaining differences (i.e., between what is expected and what actually happens), justifying differences (i.e., between a proposed solution and one used in the past), projecting outcomes, comparing and ranking alternative possibilities. Evaluation can point out the need for additional adaptation, or repair of the proposed solution. An evaluator must be able to analyze the feedback from the user system, determine when failure has occurred, explain the cause of the failure or assign the blame to a particular component of the system and then suggest ways to avoid the same failures in the future.

4.9 Memory Update and Storage

The new case is then stored appropriately in the case memory for future use. The most important process at the time of memory update is choosing the ways to "index" the new case in the memory. Indices must be chosen such that the new case can be recalled during later reasoning at times when it can be most helpful. This means that the reasoner must be able to anticipate the importance of the case for a later reasoning.

4.10 Summary

A CBR system attempts to model the ability of humans to reason from experience. In order to do this, it is important to index each past experience in such a way that it can be retrieved quickly when needed. A CBR system should be able to adapt the knowledge gained from past experiences to apply it to new situations which may be different from the retrieved experiences.

A CBR system has the ability to learn besides its ability to reason. It becomes more efficient and more competent as a result of storing its experiences and referring to them in later reasoning.

The reasoning quality of a case-based reasoner depends on the experiences it has had, its ability to understand new situations in terms of those old experiences, and its ability to adapt in unknown situations.

5. A CASE-BASED MODEL FOR POWER SYSTEM RESTORATION

5.1 Introduction

Although electric utilities constantly strive to maintain the stability and integrity of power system networks, absolute security can never be guaranteed. A catastrophic sequence of events that defeats the preventive measures and overcomes the corrective actions will occasionally infould in the network and cause a collapse of the entire power system. If such a blackout does occur, electric services must be restored as soon as possible to minimize the economic loss and the inconvenience to the public. At the same time, this restoration must proceed carefully to avoid endangering lines, damaging equipment or jeopardizing the progress already accomplished in the re-energized system.

There are two restoration periods following a major power disturbance. In the first period, several islands, which include initial sources of power, are established. Re-integration of a skeleton of the bulk power system is performed and the minimization of the unserved loads is considered in the second stage. These periods respectively correspond to section and execution of restoration subsystems, establishment and achievement of the post-restoration target. The postrestoration system, referred to as the goal state of a particular restoration process, is not necessarily identical to the predisturbance system. Some electric equipment may have been damaged, or may not be available for timely restart, or the underlying load demand may have also changed significantly due to the passage of time. A postrestoration system state can be achieved by following a logical order of switching operations. Since the set of all possible postdisturbance conditions that might occur is unmanageably large and there are a great number of candidate switches in the system, restoration is a complicated combinatorial optimization problem. It may take a long time to reach a feasible restoration plan which satisfies all the practical requirements if it is solved by combinatorial optimization algorithms. Therefore, a restoration operation requires mastering complicated operational rules and accurate judgment of situations, and depends largely on the judgment and experience of the operators. The restoration process of a power system is composed of tasks involving reasoning, heuristic search, and/or empirical judgment.

Sakaguchi and Matsumoto [2] were the first to suggest that the expert system can be used for planning switching operations. Since then this concept has been extended to the restoration of distribution systems [3] and generalized to create optimized switching sequences in substations [36]. Other authors [37,38] have proposed frameworks for high voltage networks. Reference [39] shows how an expert system has been used to assist in the automation of the restoration of a small power system. The Artificial Intelligence (AI) technique used to develop these expert systems, so far, is rule-based. In a rule-based system, the rules are small and independent but consistent pieces of domain knowledge that are in the pattern of "if-then" form. Rules are retrieved when they match the input exactly. These characteristics of an expert system have led to the difficulty in knowledge acquisition, since it has been shown that it is frustrating to articulate a set of rules from domain experts that can be used in a rule-based expert system (ES). Sometimes the experts do not remember all the details of how they solve a particular problem until they have to solve it again [40]. The knowledge acquisition bottleneck is difficult to overcome. The other disadvantage of a rule-based ES is that the addition of one rule often requires modification of several other rules and a fix inevitably interferes with the part of the system that has previously worked correctly. The time required to trace and fix a new problem is prohibitive. This maintenance is required to continue throughout the life time of the system. An alternate approach that overcomes the disadvantages associated with the development and the maintenance of rule-based ES has been presented in the following sections.

5.2 Applying Case-Based Reasoning in Planning

In artificial intelligence terminology, the restoration of a power system is classified as a planning problem. There are two concerns that must be dealt with while planning. The first concern is the problem of protection: plans should be sequenced, the later steps in the plan should not undo the results of the earlier steps, and preconditions of the later steps in the plan are not violated by the results of the earlier steps. This requires that the effects of plan steps be projected into the rest of the plan. The second concern is the problem of precondition: a planner must make sure that the preconditions of any plan step are fulfilled before scheduling that plan step. Thus, planning involves scheduling steps that achieve preconditions in addition to scheduling major steps themselves. Together, these two problems when solved by traditional nonlinear planning methods require considerable computational effort. As the number of plan steps increases, the computational complexity of projecting effects and comparing preconditions increases exponentially [41].

CBR deals with these planning problems by providing plans that have already been used and in which these problems have already been worked out. The planner is required only to make relatively minor fixes in those plans rather than having to plan from scratch. In a complex problem, the number of goals competing for achievement at any time can be quite high and if each one is achieved independently from others, the total planning and execution time are at least the sum of the time needed to achieve each one, and probably more, because there are interactions. If the mechanism of achieving several goals simultaneously or in conjunction with each other is used, the execution time can be cut significantly. In a CBR system, previously used plans can be saved and indexed by the conjunction of goals they achieve. If the conjunction of goals is repeated, the old plan that achieved them together can be recalled and repeated.

5.3 Applying Case-Based Reasoning in Power System Restoration

The earliest case-based planner is CHEF. A recipe of CHEF is an already-worked-out plan that provides an ordering of steps that plans for and protects preconditions of each of its steps. CHEF creates new recipes based on those it already knows about. Its first step is to find a single plan (old recipe) that satisfies as many of its active goals as possible. Continuation of altering or adapting the old plan is then carried out until all goals of the plan are satisfied. After trying out its plans, it repairs them appropriately using general planning knowledge.

The CBR mechanism applied to stipulate the power system restoration plan is similar to the architecture of CHEF. The process begins with the analysis and diagnosis of the current situation. After that, the main features of the current situation are extracted to determine the index for the situation and the case analyzer then traverses the case library with the index. As the retrieval algorithm searches, it calls on a matching procedure to assess the degree of match along each individual dimension of the index, and returns a list of partially-matching cases, each of which has at least some potential to be useful in the case-based reasoning. A ranking procedure analyzes the set of cases to determine which of them has the most potential to be useful and recommends the best matching one. Since the problem description of the best matching case is not exactly the same as the current situation, the case adapter replaces one or more pieces of a solution in the best matching case with the new pieces of solution which will fit to the current situation. The case executor then applies this restoration plan to the current situation and gets feedback from the power system. Finally the new case with the record of result will be added into the case library by case organizer. The block diagram of the CBR system used in this research is shown in Figure 5.1.

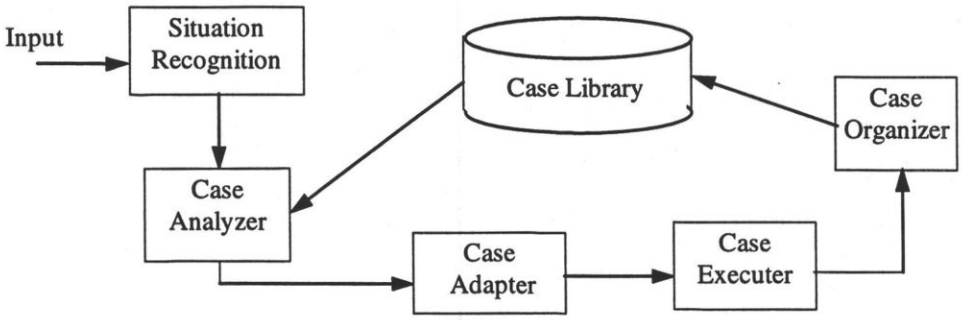


Figure 5.1 The CBR architecture

The remainder of the chapter deals with the following concerns: what kind of case language has to be developed to present the restoration knowledge? what kind of retrieval algorithm has to be used in case analyzer? how the cases have been organized in the case library, how the old solution in the prior case has been modified in an attempt to construct a specific solution for the new problem and finally how the solution has to be tried in the power system restoration domain.

5.3.1 Case Representation

The initial task in the application of the CBR to power system restoration domain is to construct a case structure and design a suitable case language for describing this case structure.

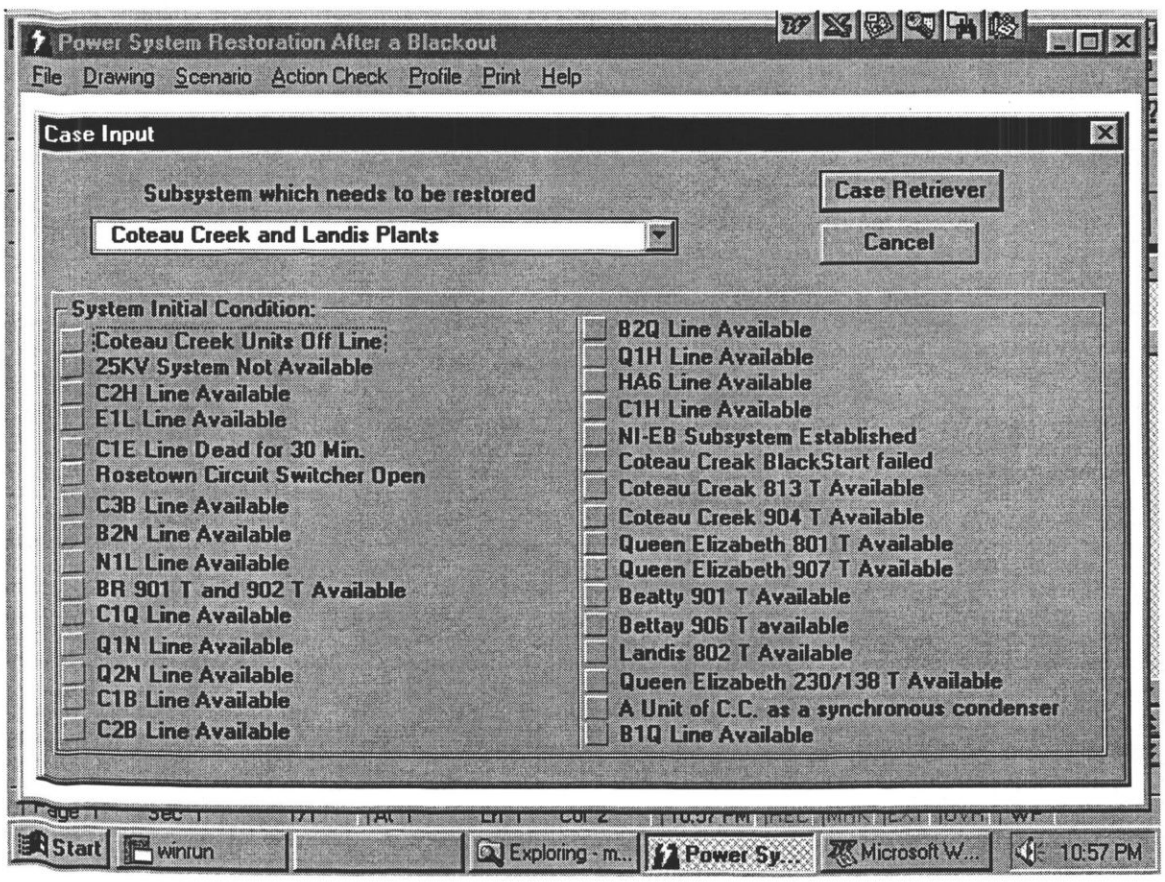
In this research, the case structure has been defined as a frame which consists of three components:

1. a primary slot that holds the main task specification,
2. a number of basic slots that represent background information related to the main task,
3. a method used to determine an answer to the main question.

During the restoration of a power system, the main task is to restore the system to achieve the goal state or postrestoration state. The background information is referred to as postdisturbance conditions. Power system restoration is a carefully stipulated operation sequence which, once executed, will lead the postdisturbance state to the postrestoration state (goal state).

The case language that has been used is the enumeration of the names of the slots in a case structure and an enumeration of the values that may be entered in the slots.

In order to speed up the restoration process, a two-phase strategy has been used in the restoration process. A power system under blackout condition is divided into several restoration subsystems which can be energized at the same time. Some subsystems are generating stations which include blackstart units, while others are major switching substations. This restoration planning strategy has been acquired and presented as a decomposition of the whole restoration process into a set of goals or objectives. Examples of goals include blackstarting generation units, energizing substations and supplying cranking power to generating stations. In the case structure, themes have been used to reflect the decomposition. The theme of a case structure is an organizational principle that allows some degree of coherence among the case slots. Some of the slots are organized around one theme and other slots are organized around another theme. The themes of the case structure are goals which have been produced from the decomposition. Around each theme, there are some slots that represent postdisturbance conditions related to the theme. Figures 5.2 and 5.3 show two themes and their related slots. The upper part of each of the figures shows the main theme or goal and the lower part shows a checklist which consists of slots that present all possible background information related to the goal state. Examples of background information are the status of system elements and operation constraints. System elements are circuit breakers, generating units, major transformers and transmission lines between major substations.


Power System Restoration After a Blackout
File Drawing Scenario Action Check Profile Print Help
Case Input
Case Retriever
Cancel
Start
winrun
Exploring - m...
Power Sy...
Microsoft W...
10:57 PM

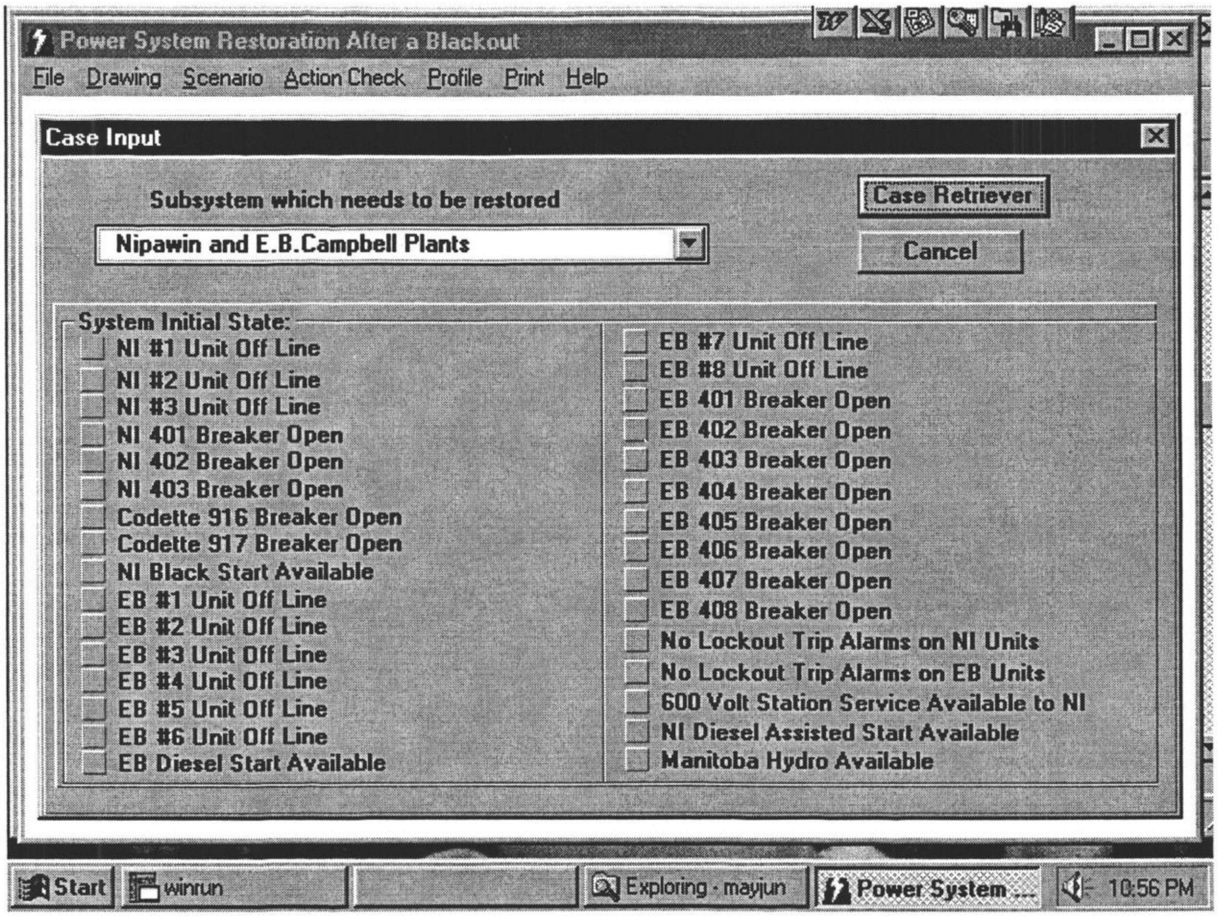
Subsystem which needs to be restored

Coteau Creek and Landis Plants

System Initial Condition:

- Coteau Creek Units Off Line
- 25KV System Not Available
- C2H Line Available
- E1L Line Available
- C1E Line Dead for 30 Min.
- Rosetown Circuit Switcher Open
- C3B Line Available
- B2N Line Available
- N1L Line Available
- BR 901 T and 902 T Available
- C1Q Line Available
- Q1N Line Available
- Q2N Line Available
- C1B Line Available
- C2B Line Available
- B2Q Line Available
- Q1H Line Available
- HA6 Line Available
- C1H Line Available
- NI-EB Subsystem Established
- Coteau Creak BlackStart failed
- Coteau Creak 813 T Available
- Coteau Creek 904 T Available
- Queen Elizabeth 801 T Available
- Queen Elizabeth 907 T Available
- Beatty 901 T Available
- Bettay 906 T available
- Landis 802 T Available
- Queen Elizabeth 230/138 T Available
- A Unit of C.C. as a synchronous condenser
- B1Q Line Available

Figure 5.2 Theme #1 of Case Structure


Power System Restoration After a Blackout

File Drawing Scenario Action Check Profile Print Help

Case Input

Subsystem which needs to be restored

Nipawin and E.B.Campbell Plants

Case Retriever

Cancel

System Initial State:

- ☐ NI #1 Unit Off Line
- ☐ NI #2 Unit Off Line
- ☐ NI #3 Unit Off Line
- ☐ NI 401 Breaker Open
- ☐ NI 402 Breaker Open
- ☐ NI 403 Breaker Open
- ☐ Codette 916 Breaker Open
- ☐ Codette 917 Breaker Open
- ☐ NI Black Start Available
- ☐ EB #1 Unit Off Line
- ☐ EB #2 Unit Off Line
- ☐ EB #3 Unit Off Line
- ☐ EB #4 Unit Off Line
- ☐ EB #5 Unit Off Line
- ☐ EB #6 Unit Off Line
- ☐ EB Diesel Start Available
- ☐ EB #7 Unit Off Line
- ☐ EB #8 Unit Off Line
- ☐ EB 401 Breaker Open
- ☐ EB 402 Breaker Open
- ☐ EB 403 Breaker Open
- ☐ EB 404 Breaker Open
- ☐ EB 405 Breaker Open
- ☐ EB 406 Breaker Open
- ☐ EB 407 Breaker Open
- ☐ EB 408 Breaker Open
- ☐ No Lockout Trip Alarms on NI Units
- ☐ No Lockout Trip Alarms on EB Units
- ☐ 600 Volt Station Service Available to NI
- ☐ NI Diesel Assisted Start Available
- ☐ Manitoba Hydro Available

Start | winrun | Exploring - mayjun | Power System ... | 10:56 PM

Figure 5.3 Theme #2 of Case Structure

The solution part consists of textualized information which describes a restoration sequence in a step by step fashion. A case is an instance of the case structure after values are assigned to the case slots by using the case language.

An example case for restoring a subsystem of SaskPower cooperation is shown below. The subsystem has two plants and both of them have blackstart capability. One is called Nipawin (NI) and the other is called E.B.Campbell (EB). There is one transformer substation called Codette (CE) between these two plants. The single-line diagram of the subsystem is shown in Figure 5.4.

Case #3:

Goal: (Establishing the Nipawin-E.B. Campbell power island);

Operation Constraints: (Manitoba Hydro not available);

Initial Situation:

(Nipawin #1 generator off line, available for blackstart)

(Nipawin #2 generator off line, available for blackstart)

(Nipawin #3 generator off line, available for blackstart)

(E.B.Campbell #1 generator off line)

(E.B.Campbell #2 generator off line)

(E.B.Campbell #3 generator off line)

(E.B.Campbell #4 generator off line)

(E.B.Campbell #5 generator off line)

(E.B.Campbell #6 generator off line)

(E.B.Campbell #7 generator off line)

(E.B.Campbell #8 generator off line)

(Nipawin breaker #401 open)

(Nipawin breaker #402 open)

(Nipawin breaker #403 open)

(Codette #916 breaker open)

(Codette #917 breaker open)

(No 600 volt station service to Nipawin plant)

....

Solution:

Step 1: (Black Start one of Nipawin generators which was on line most recently)

Step 2: (Normal Start second Nipawin generator using normal procedure)

Step3: (Make alive Codette station service)

Step4: (Adjust 72 KV voltage to 72 KV and energize Codette/C1T line)

Step 5: (Pick up approximately 3 MW of load)

Step 6: (Energize E2C line to E.B.Campbell)

Step 7: (Make alive E.B.Campbell generator transformer to supply station power)

Step 8: (Start and synchronize at least 4 small E.B.Campbell units)

Step 9: (Start and synchronize third Nipawin unit using the normal start up procedure)

Step 10: (Energize CE4 line, pick up approximately 20 MWs of load to stabilize the on line units)

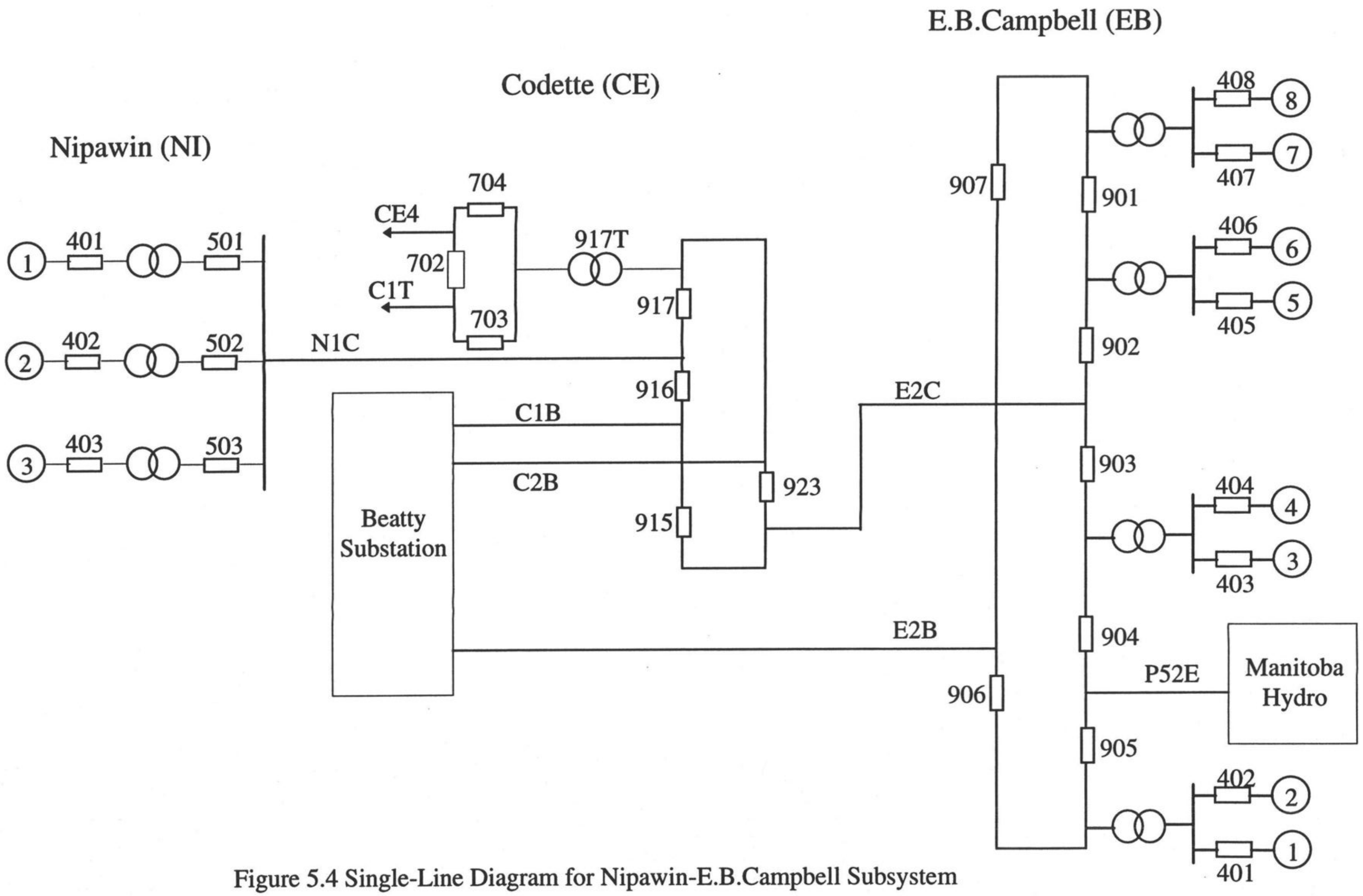


Figure 5.4 Single-Line Diagram for Nipawin-E.B.Campbell Subsystem

5.3.2 Situation Recognition

Once a case is cast from the case structure, some descriptors have to be attached to it as indices for the retrieving algorithm to find the cases from the case library. The retrieving process looks through the case library and selects those cases that match with the current problem descriptors. The ability to access efficiently a few of the cases which are most relevant to the situation is almost entirely dependent on the effectiveness of the indexing scheme i.e. how the situation is recognized. Situation recognition is the process of sizing up a situation and determining what the indices for a similar case would be if it were in the case library. It includes figuring out what kind of situation the new one is, what is important about it, and what else may be true about it.

On one hand, the representation of the situation may not sufficiently match anything in the case library. This happens when a new situation is described along dimensions different from those used as indices or descriptive features of cases in the case library. Elaborating the description of the new situation in order to bridge the gap between its description and descriptions of cases in the case library is a necessary prerequisite to retrieve any cases. On the other hand, the representation of a new situation sometimes is so general that it matches a large number of items in the case library that are very different from each other, and there is no good way of determining which ones are better than the others. Elaborating on a situation makes it possible to distinguish which of the many partially-matching cases have the potential to be most useful. Cases have to be indexed by features and feature combinations that are both predicative and concrete enough to be recognized. In situation recognition, the details of the initial situations are interpreted and analyzed, and index is produced.

Although a well-structured language has been utilized for categorizing different kinds of restoration problems, as shown in Figures 5.2 and 5.3, this checklist style of describing problems does not work efficiently due to its inability to discriminate the important dimensions from the unimportant ones. This style results in overindexing and retrieval of too many cases. To overcome these problems, a selection of problems has been examined to determine the features, or dimensions, that tend to be effective for good retrieval and to filter out the unimportant or irrelevant features from the checklist by applying heuristic rules.

Since cases record the experiences in achieving the goals that have worked and have not worked, and goals specify how to configure pieces of background information with respect to each other, the goal state is a major part of an index of a case. The tasks for achieving a goal state have been determined and indexed as features.

The initial conditions of power system elements, some operating constraints, and knowledge of restoration priorities have been analyzed to determine the features that will make useful predictions and are easily recognizable for each task. Some of the features are relevant to a particular kind of problem, and others tend to be predictive of other features. These features have been recorded in relevance rules. A relevance rule is a structure of the form, "If A then B", where A represents a problem type, and B indicates a set of parameters that are required to find a solution to A.

Examples of relevance rules which have been applied to index features are:

If a generating unit has both diesel assisted start and blackstart available
then chose the diesel assisted start.
If a neighbouring power source is available, then energize the transmission lines that connects neighbouring system to the subsystem under restoration.

5.3.3 Case Analyzer

The functionality of the case analyzer is to find prior cases whose problem description exhibit maximum similarity to the problem description of the current case. It has three functions: case retrieving, case matching and case ranking. Using the previously stated indexing scheme, the case analyzer performs an exercise in matching each feature of the index of the new case with that of the cases in the case library to determine which cases best match the current case.

The retrieval component of the case analyzer must run quickly and retrieves a reasonable number of relevant cases. In this research, the retrieval component is implemented as a two step process. Rather than searching the entire case library for matching, a shallow search has been applied in the first step and a deep search in the second step. A set of relevance rules has been used to select cases that are worth applying the matching functions to because these cases share important features with the new situation. The relevance rules are as follows:

If the main feature is of type 1, then match on subset case S1 of the case library.
If the main feature is of type 2, then match on subset case S2 of the case library.
If the main feature is of type 3, then match on subset case S3 of the case library.

The retrieval algorithm puts pointers on all cases that match with the main feature type in a buffer. The major benefit of this search strategy is that the first retrieval can provide insights into the important features of the current case in order to lessen the burden on the retrieval algorithm and to alleviate the inefficiencies.

The case analyzer then performs the matching and ranking routine on retrieved cases within the buffer with the increasing precision of the problem. The technique utilized by the matching and ranking algorithm is the greatest number of matching features. It searches between the proposed case and the cases in the buffer with respect to a list of features and scores the cases. The flowchart of the matching algorithm is shown in Figure 5.5.

The ranking routine then sorts each scored case and takes the case with the greatest score as the recommendation. The flowchart of the ranking algorithm is shown in Figure 5.6.

5.3.4 Case Adapter

In this phase, the solution that was used in the old case is examined to determine whether the same solution would work in a new situation.

Since a new situation is rarely the same as the case retrieved from the case library, the solution part of the best matched case is adapted before submitting it to the user. The null adaptation technique has been used in this research. The null adaptation is a case adaptation technique in which a solution in a retrieved case is mapped directly to a new case, with no adjustments or modifications to the solution. Null adaptation is necessary to start the learning process of a CBR system. At the beginning of the application of a CBR system in a particular domain, a seed case library has to be developed. These cases act as seeds in the library and basically offer solutions to general problems that occur during restoration.

The solution proposal after adaptation is displayed on the screen with single-line diagrams as instructions to the user. A solution proposal for the restoration of a subsystem of a power system is shown in Figure 5.7.

5.3.5 Case Executor

After retrieving a solution from the CBR system, the solution is adapted to fit a specific problem. The adapted solution in the form of a proposal is then executed to restore a power system. A manual execution method has been applied to execute the restoration sequence. A user interprets a solution proposal made by the CBR system and decides whether to follow the operation sequence, and makes the actual switching actions on the related single-line diagrams.

5.3.6 Case Organizer

The final step in a CBR process is to understand the notion of acquiring experience and to accumulate the experience in a formal way. This step is construed as executing a proposed solution, evaluating the results and embedding an experience with a proposed solution in the case library.

A possible approach to the evaluation of results is to periodically review the behaviour of the CBR system. This review consists of studying the cases that have been added to the library since the previous review. If the system does not perform well on some cases, those cases should be modified and then stored back in the library. This way, any bad cases will be removed from the library and in their place, good cases for use in the future will exist. The evaluation component has not been implemented in this research, but will be a part of the future research.

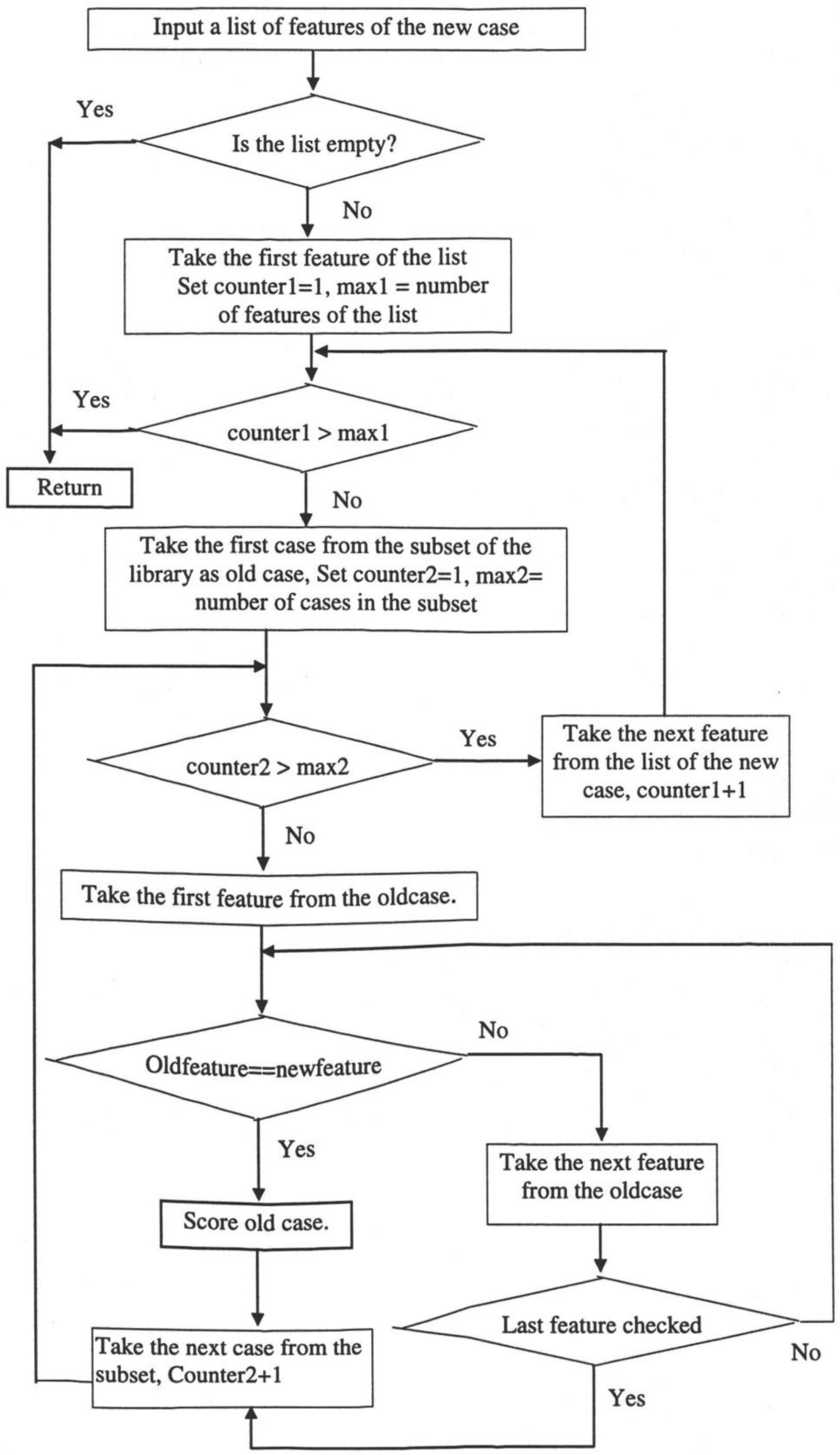

Input a list of features of the new case
Is the list empty?
Yes
No
Take the first feature of the list Set counter1=1, max1 = number of features of the list
counter1 > max1
Yes
Return
No
Take the first case from the subset of the library as old case, Set counter2=1, max2= number of cases in the subset
counter2 > max2
Yes
Take the next feature from the list of the new case, counter1+1
No
Take the first feature from the oldcase.
Oldfeature==newfeature
No
Yes
Take the next feature from the oldcase
Score old case.
Last feature checked
No
Yes
Take the next case from the subset, Counter2+1

Figure 5.5 The Flowchart of the Matching Algorithm

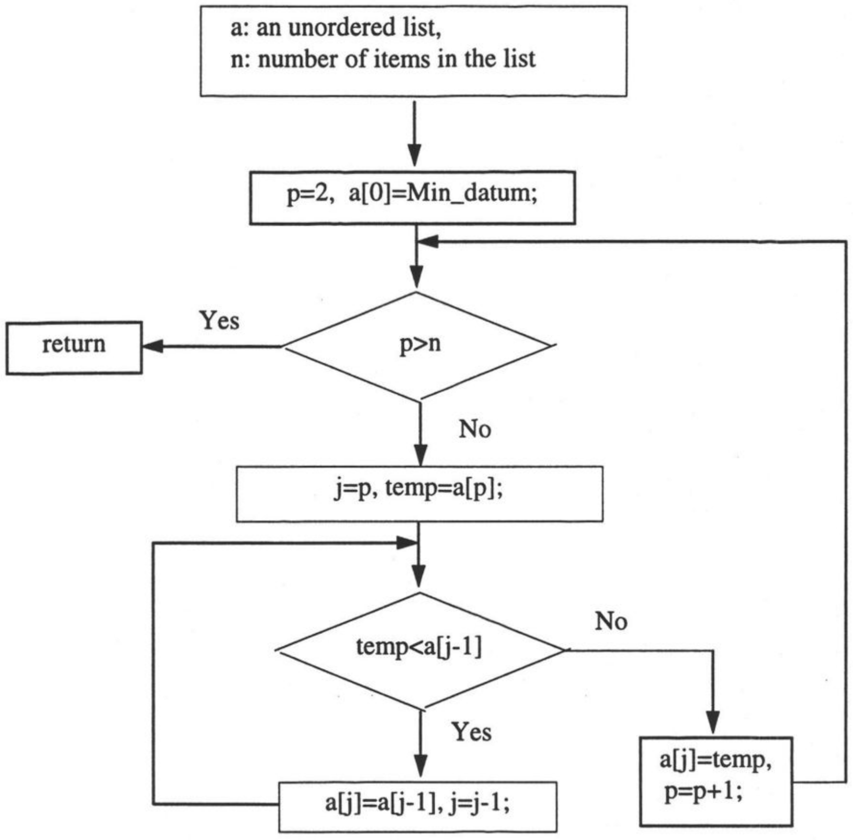
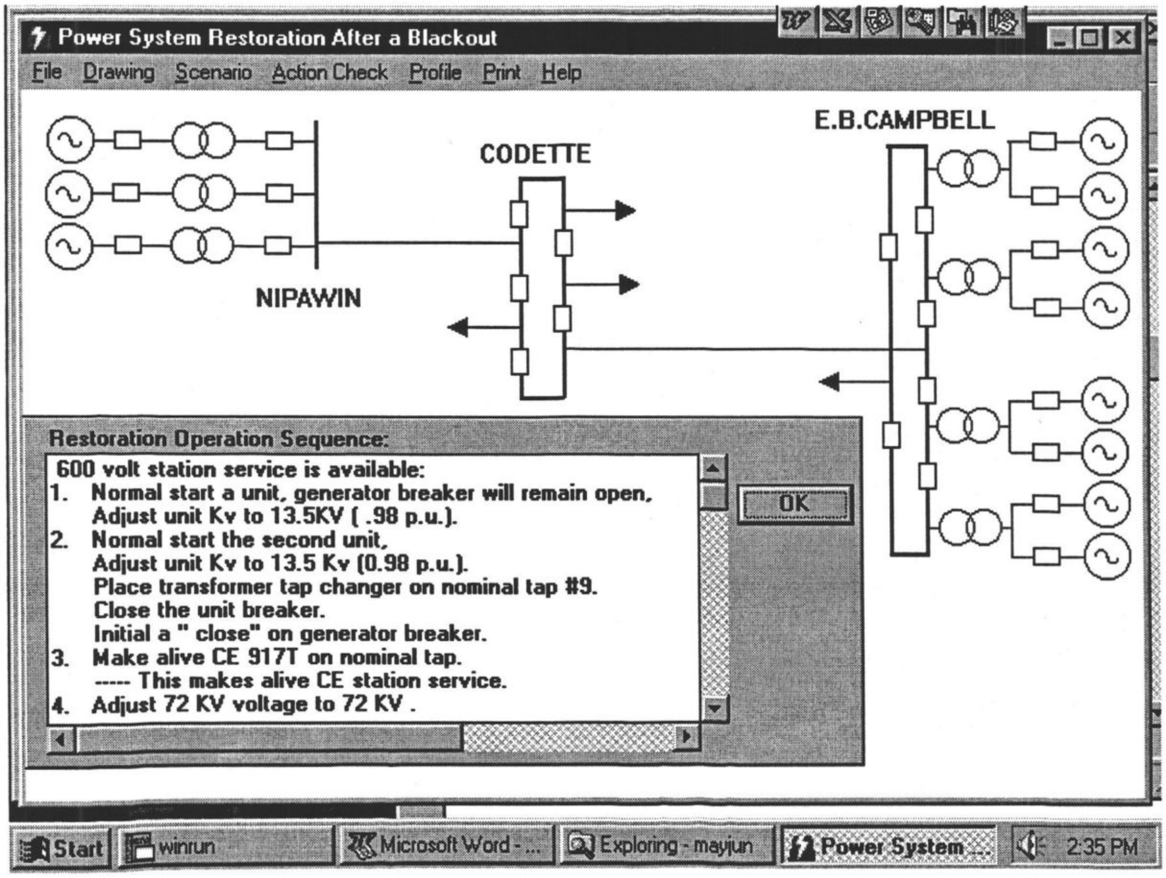


Figure 5.6 The Flowchart of the Ranking Algorithm

A flat memory structure has been used to store cases in the case library in a sequential manner.

5.4 Test System

The network of SaskPower Cooperation has been selected for studying the steps of restoration. SaskPower system has three major power plants which have two thirds of the system capacity and are located far south in the province of Saskatchewan. SaskPower has three hydro plants which have blackstart capability. These blackstart units are located far north in the province. The system has a long corridor from the major plants in the south to the blackstart plants in the north with four more plants between them. A total blackout of the system has been assumed as the postdisturbance condition. The postrestoration system has been divided into 7 target subsystems according to the blackstart capability and geographical locations of the generating units and some of them can be executed simultaneously. The corresponding cases for each subsystem are stored in the case library. Two parts of the case structure for the restoration of Nipawin to E.B. Campbell subsystem and Coteau Creek to Landis subsystem are shown in Figures 5.2 and 5.3 respectively. A user describes the initial situation by marking the status of the related system components on the slots of the lower part of the case structure. After the initial situation is described, the button captioned "Case Retriever" needs to be clicked to start the situation recognition program to index the current situation, then the case analyzer performs the process of searching, matching and ranking the cases in the library. Finally, the case adapter submits the proposal to the user. Several typical seed cases related to the restoration of the SaskPower subsystems have been illustrated below.



Restoration Operation Sequence:

600 volt station service is available:
1. Normal start a unit, generator breaker will remain open,
   Adjust unit Kv to 13.5KV (.98 p.u.).
2. Normal start the second unit,
   Adjust unit Kv to 13.5 Kv (0.98 p.u.).
   Place transformer tap changer on nominal tap #9.
   Close the unit breaker.
   Initial a " close" on generator breaker.
3. Make alive CE 917T on nominal tap.
   ----- This makes alive CE station service.
4. Adjust 72 KV voltage to 72 KV .

Figure 5.7. Restoration Proposal

Situation 1:   Restoration of Nipawin Plant and E.B. Cambell Plant subsystem
Postdisturbance condition: Nipawin #1-- #3 units off line

Nipawin # 401-- #403 breakers open

E.B.Campbell #1-- #8 units off line

E.B.Campbell #401--#408 breakers open

Codette 916 & 917 breakers open

Manitoba Hydro Available

The case instance to reflect this situation is shown in Figure 5.8 and the solution proposal in terms of a restoration sequence is shown in Figure 5.9.

Situation 2:   Restoration of Nipawin Plant and E.B. Campbell Plant subsystem

Postdisturbance condition: Nipawin #1-- #3 units off line

Nipawin # 401-- #403 breakers open

E.B.Campbell #1-- #8 units off line

E.B.Campbell #401--#408 breakers open

Codette 916 & 917 breakers open

Nipawin blackstart available

The case instance is shown in Figure 5.10 and the solution proposal which is provided by the CBR system is shown in Figure 5.11.

Situation 3:   Restoration of Nipawin Plant and E.B. Cambell Plant subsystem
Postdisturbance condition: Nipawin #1-- #3 units off line

Nipawin # 401-- #403 breakers open

E.B.Campbell #1-- #8 units off line

E.B.Campbell #401--#408 breakers open

Codette 916 & 917 breakers open

Nipawin diesel assisted start available

Nipawin blackstart available

Figure 5.12 shows the case description for Situation 3, and the corresponding solution proposal is shown in Figure 5.13.

Situation 4: Restoration of Nipawin Plant and E.B. Cambell Plant subsystem
Postdisturbance condition: Nipawin #1-- #3 units off line

Nipawin # 401-- #403 breakers open

E.B.Campbell #1-- #8 units off line

E.B.Campbell #401--#408 breakers open

Codette 916 & 917 breakers open

Nipawin units blackstart available

Nipawin diesel assisted start available

600 Volt station service available to Nipawin

The corresponding case description and solution proposal are shown in Figures 5.14 and 5.15 respectively.

Situation 5: Restoration of Nipawin Plant and E.B. Cambell Plant subsystem
Postdisturbance condition: Nipawin #1-- #3 units off line

Nipawin # 401-- #403 breakers open

E.B.Campbell #1-- #8 units off line

E.B.Campbell #401--#408 breakers open

Codette 916 & 917 breakers open

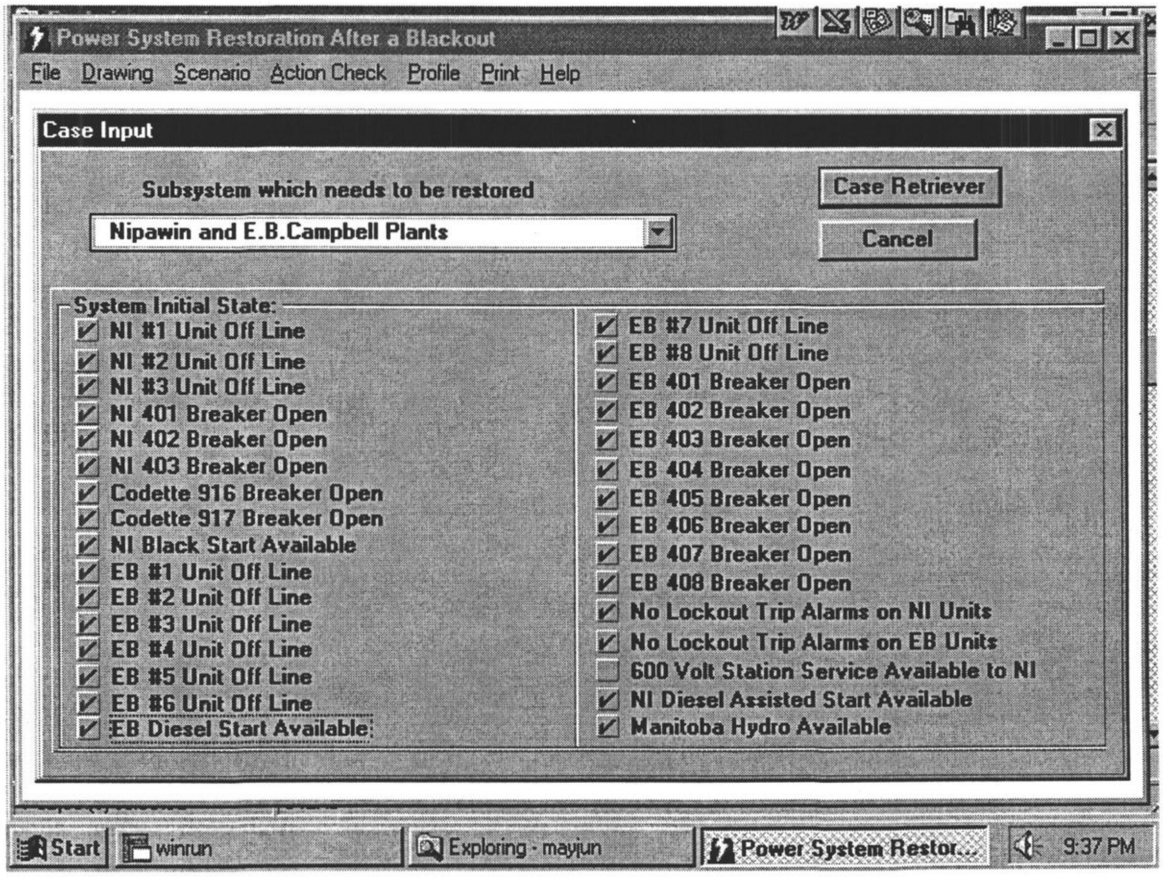
E.B.Campell diesel assisted start available

The corresponding case description and solution proposal are shown in Figures 5.16 and 5.17 respectively.

5.5 Summary

A CBR mechanism solves problem by using experience rather than the first principle. It reduces the need to acquire and explicitly represent domain knowledge of the problem and, therefore, the knowledge acquisition bottleneck can be overcome. During new problem solving, the case which is most similar to the current problem is retrieved from a case library and its solution is modified to become a solution to the current problem.

The CBR component of the ITA has been designed and implemented to make proposals of restoration sequences after the postrestoration state and postdisturbance situation are known. It has been tested by using the SaskPower network.



Power System Restoration After a Blackout

File Drawing Scenario Action Check Profile Print Help

Case Input

Subsystem which needs to be restored

Nipawin and E.B.Campbell Plants

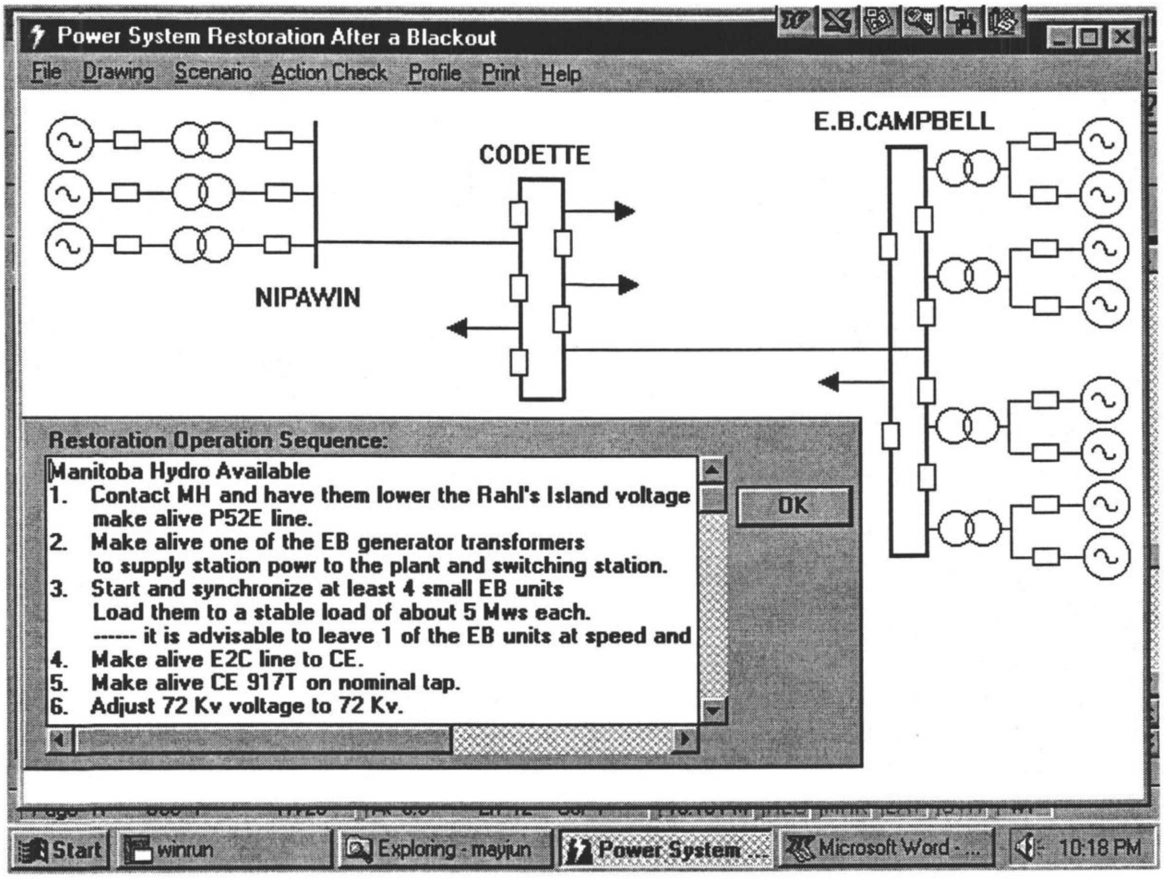
Case Retriever

Cancel

System Initial State:

- [✓] NI #1 Unit Off Line
- [✓] NI #2 Unit Off Line
- [✓] NI #3 Unit Off Line
- [✓] NI 401 Breaker Open
- [✓] NI 402 Breaker Open
- [✓] NI 403 Breaker Open
- [✓] Codette 916 Breaker Open
- [✓] Codette 917 Breaker Open
- [✓] NI Black Start Available
- [✓] EB #1 Unit Off Line
- [✓] EB #2 Unit Off Line
- [✓] EB #3 Unit Off Line
- [✓] EB #4 Unit Off Line
- [✓] EB #5 Unit Off Line
- [✓] EB #6 Unit Off Line
- [✓] EB Diesel Start Available
- [✓] EB #7 Unit Off Line
- [✓] EB #8 Unit Off Line
- [✓] EB 401 Breaker Open
- [✓] EB 402 Breaker Open
- [✓] EB 403 Breaker Open
- [✓] EB 404 Breaker Open
- [✓] EB 405 Breaker Open
- [✓] EB 406 Breaker Open
- [✓] EB 407 Breaker Open
- [✓] EB 408 Breaker Open
- [✓] No Lockout Trip Alarms on NI Units
- [✓] No Lockout Trip Alarms on EB Units
- [ ] 600 Volt Station Service Available to NI
- [✓] NI Diesel Assisted Start Available
- [✓] Manitoba Hydro Available

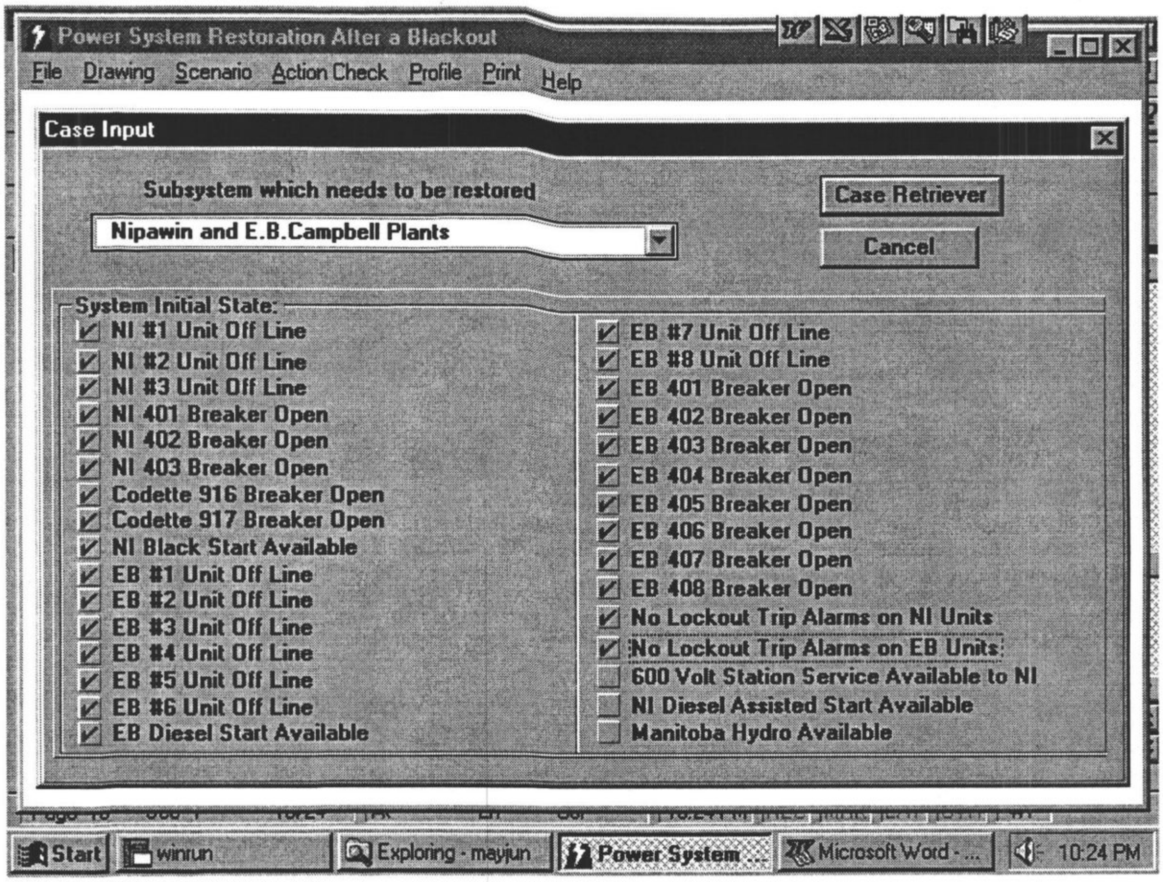
Figure 5.8 Case Description for Situation 1



Restoration Operation Sequence:

Manitoba Hydro Available
1. Contact MH and have them lower the Rahl's Island voltage
make alive P52E line.
2. Make alive one of the EB generator transformers
to supply station powr to the plant and switching station.
3. Start and synchronize at least 4 small EB units
Load them to a stable load of about 5 Mws each.
------ it is advisable to leave 1 of the EB units at speed and
4. Make alive E2C line to CE.
5. Make alive CE 917T on nominal tap.
6. Adjust 72 Kv voltage to 72 Kv.

Figure 5.9 Solution Proposal for Situation 1

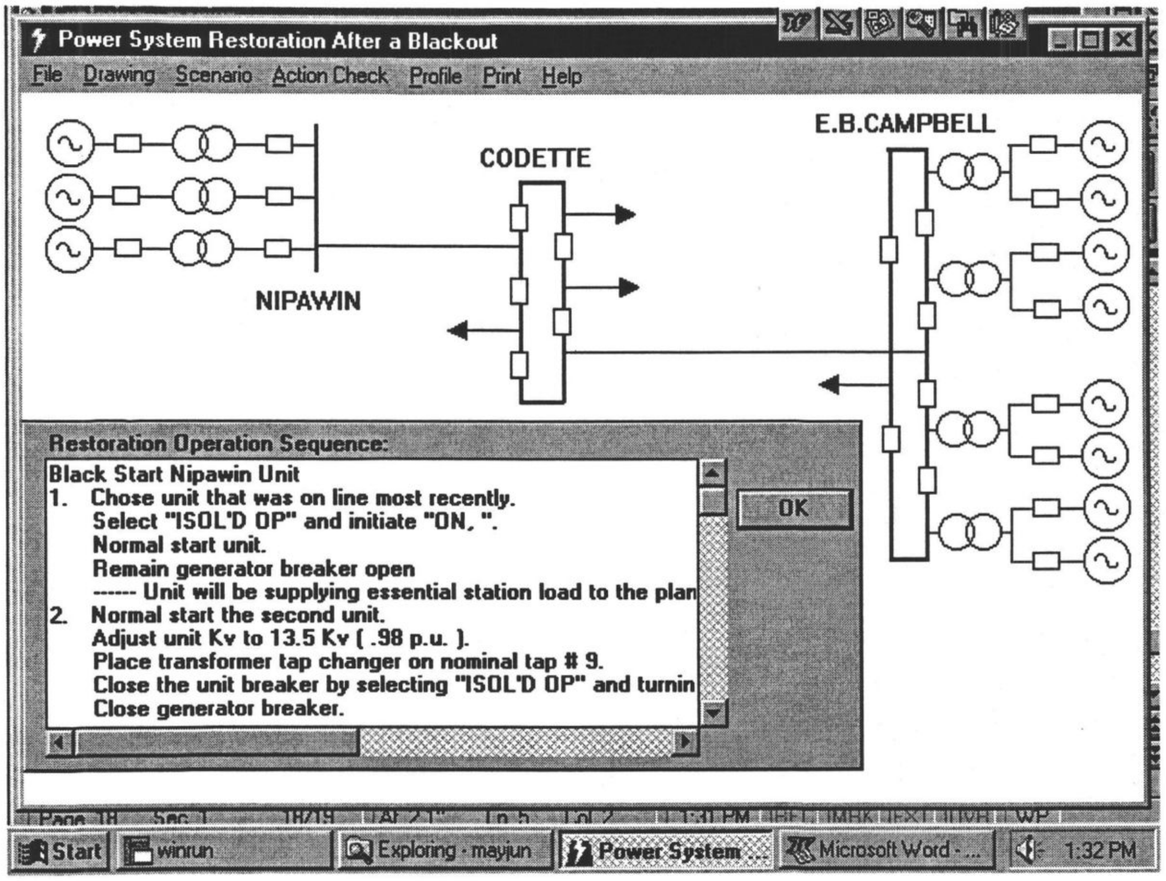


Subsystem which needs to be restored: Nipawin and E.B.Campbell Plants

System Initial State:

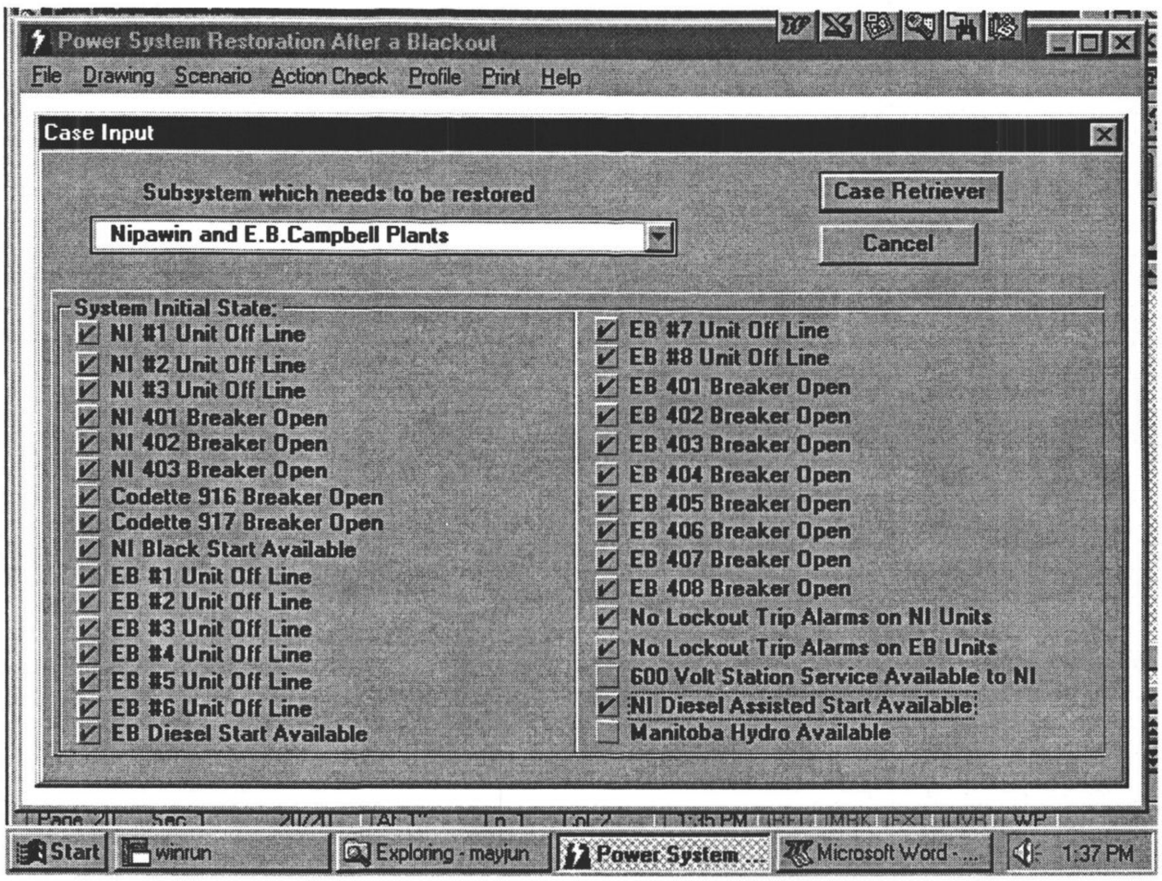
- [✓] NI #1 Unit Off Line
- [✓] NI #2 Unit Off Line
- [✓] NI #3 Unit Off Line
- [✓] NI 401 Breaker Open
- [✓] NI 402 Breaker Open
- [✓] NI 403 Breaker Open
- [✓] Codette 916 Breaker Open
- [✓] Codette 917 Breaker Open
- [✓] NI Black Start Available
- [✓] EB #1 Unit Off Line
- [✓] EB #2 Unit Off Line
- [✓] EB #3 Unit Off Line
- [✓] EB #4 Unit Off Line
- [✓] EB #5 Unit Off Line
- [✓] EB #6 Unit Off Line
- [✓] EB Diesel Start Available
- [✓] EB #7 Unit Off Line
- [✓] EB #8 Unit Off Line
- [✓] EB 401 Breaker Open
- [✓] EB 402 Breaker Open
- [✓] EB 403 Breaker Open
- [✓] EB 404 Breaker Open
- [✓] EB 405 Breaker Open
- [✓] EB 406 Breaker Open
- [✓] EB 407 Breaker Open
- [✓] EB 408 Breaker Open
- [✓] No Lockout Trip Alarms on NI Units
- [✓] No Lockout Trip Alarms on EB Units
- [ ] 600 Volt Station Service Available to NI
- [ ] NI Diesel Assisted Start Available
- [ ] Manitoba Hydro Available

Figure 5.10 Case Description for Situation 2


Power System Restoration After a Blackout
File Drawing Scenario Action Check Profile Print Help
E.B.CAMPBELL
CODETTE
NIPAWIN
Restoration Operation Sequence:
OK
Start
winrun
Exploring - mayjun
Power System ...
Microsoft Word - ...
1:32 PM

Black Start Nipawin Unit
1. Chose unit that was on line most recently.
Select "ISOL'D OP" and initiate "ON, ".
Normal start unit.
Remain generator breaker open
------ Unit will be supplying essential station load to the plan
2. Normal start the second unit.
Adjust unit Kv to 13.5 Kv (.98 p.u.).
Place transformer tap changer on nominal tap # 9.
Close the unit breaker by selecting "ISOL'D OP" and turnin
Close generator breaker.

Figure 5.11 Solution Proposal for Situation 2

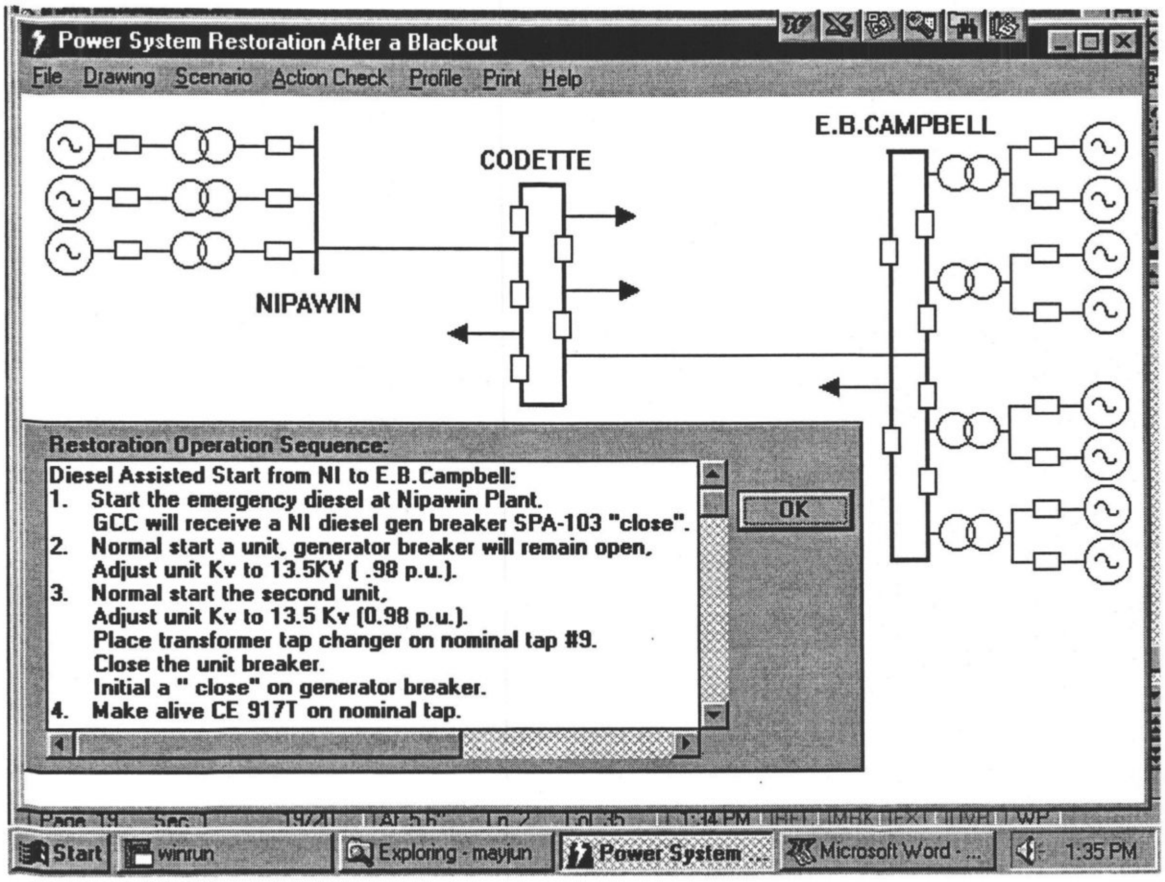


Subsystem which needs to be restored: Nipawin and E.B.Campbell Plants

System Initial State:

- [✓] NI #1 Unit Off Line
- [✓] NI #2 Unit Off Line
- [✓] NI #3 Unit Off Line
- [✓] NI 401 Breaker Open
- [✓] NI 402 Breaker Open
- [✓] NI 403 Breaker Open
- [✓] Codette 916 Breaker Open
- [✓] Codette 917 Breaker Open
- [✓] NI Black Start Available
- [✓] EB #1 Unit Off Line
- [✓] EB #2 Unit Off Line
- [✓] EB #3 Unit Off Line
- [✓] EB #4 Unit Off Line
- [✓] EB #5 Unit Off Line
- [✓] EB #6 Unit Off Line
- [✓] EB Diesel Start Available
- [✓] EB #7 Unit Off Line
- [✓] EB #8 Unit Off Line
- [✓] EB 401 Breaker Open
- [✓] EB 402 Breaker Open
- [✓] EB 403 Breaker Open
- [✓] EB 404 Breaker Open
- [✓] EB 405 Breaker Open
- [✓] EB 406 Breaker Open
- [✓] EB 407 Breaker Open
- [✓] EB 408 Breaker Open
- [✓] No Lockout Trip Alarms on NI Units
- [✓] No Lockout Trip Alarms on EB Units
- [ ] 600 Volt Station Service Available to NI
- [✓] NI Diesel Assisted Start Available
- [ ] Manitoba Hydro Available

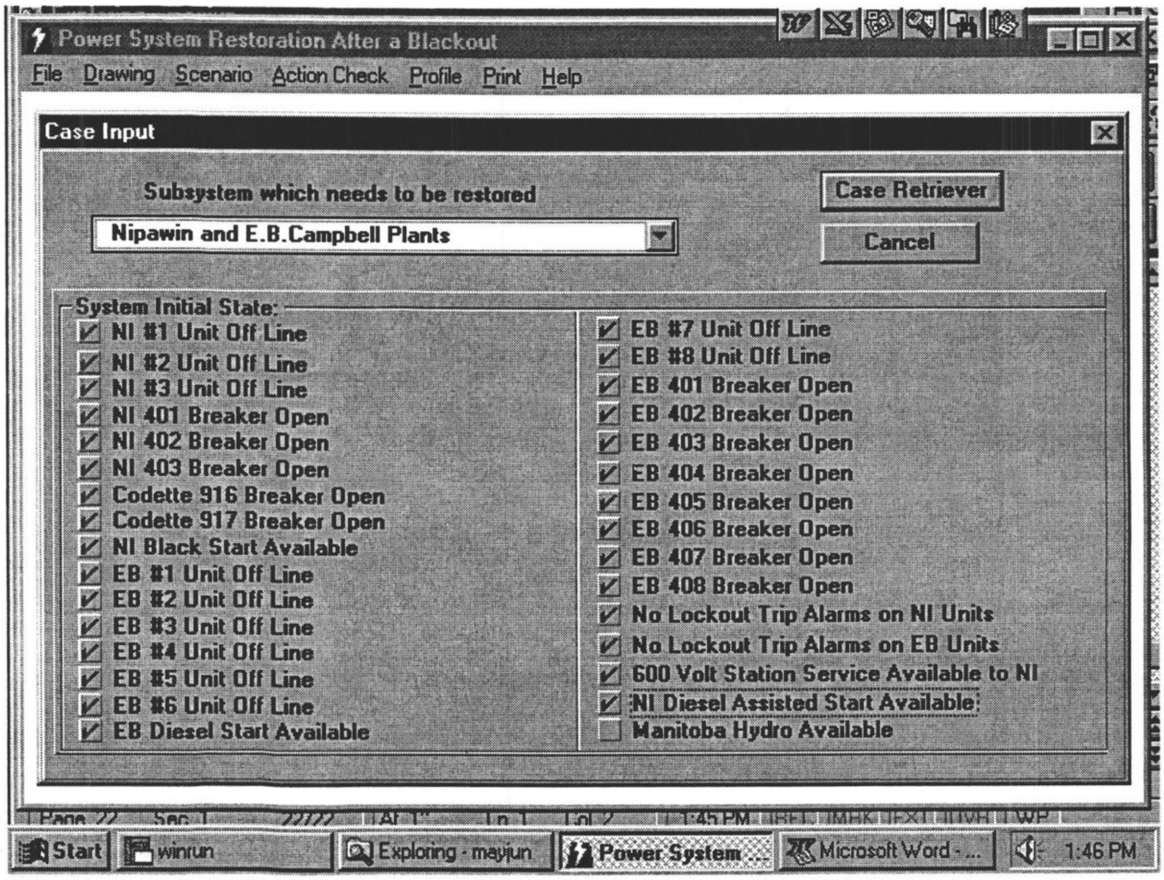
Figure 5.12 Case Description for Situation 3



Restoration Operation Sequence:

Diesel Assisted Start from NI to E.B.Campbell:
1. Start the emergency diesel at Nipawin Plant.
   GCC will receive a NI diesel gen breaker SPA-103 "close".
2. Normal start a unit, generator breaker will remain open,
   Adjust unit Kv to 13.5KV (.98 p.u.).
3. Normal start the second unit,
   Adjust unit Kv to 13.5 Kv (0.98 p.u.).
   Place transformer tap changer on nominal tap #9.
   Close the unit breaker.
   Initial a " close" on generator breaker.
4. Make alive CE 917T on nominal tap.

Figure 5.13 Solution Proposal for Situation 3



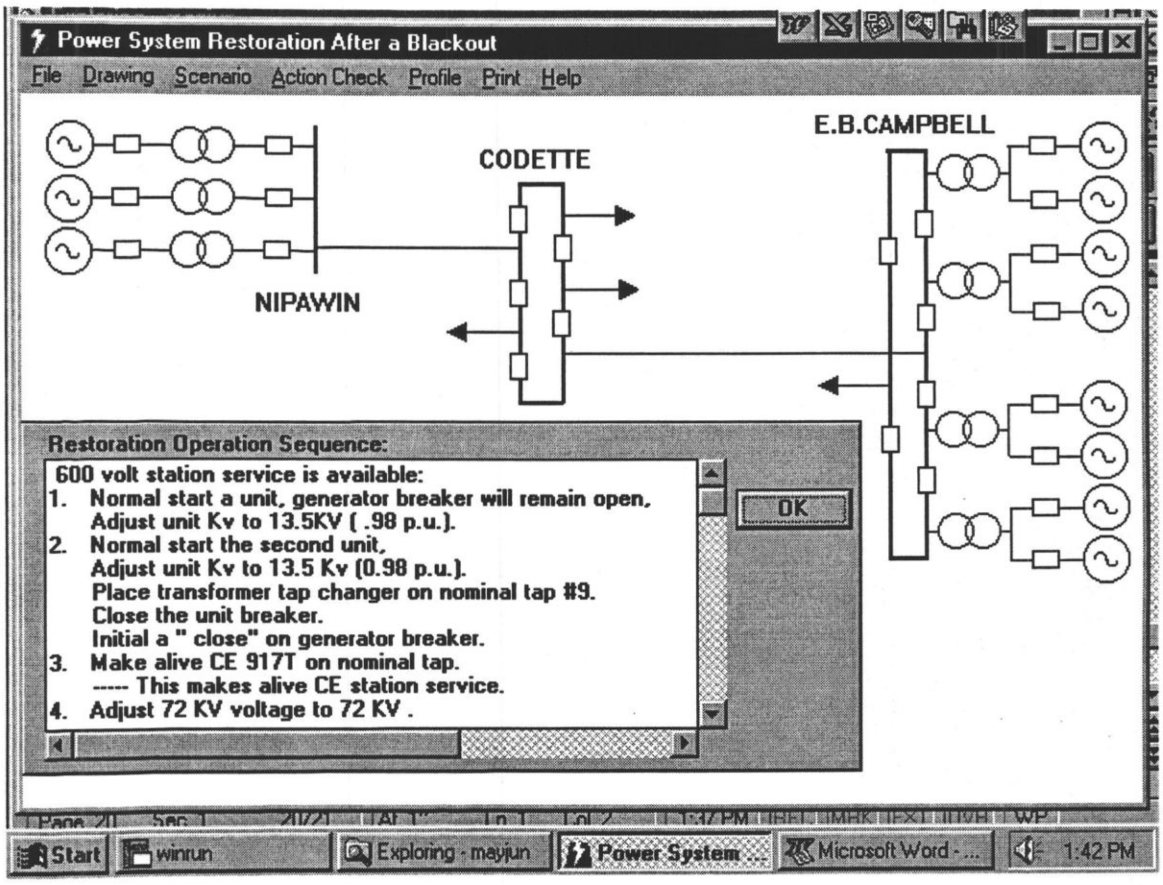
Subsystem which needs to be restored

Nipawin and E.B.Campbell Plants

System Initial State:

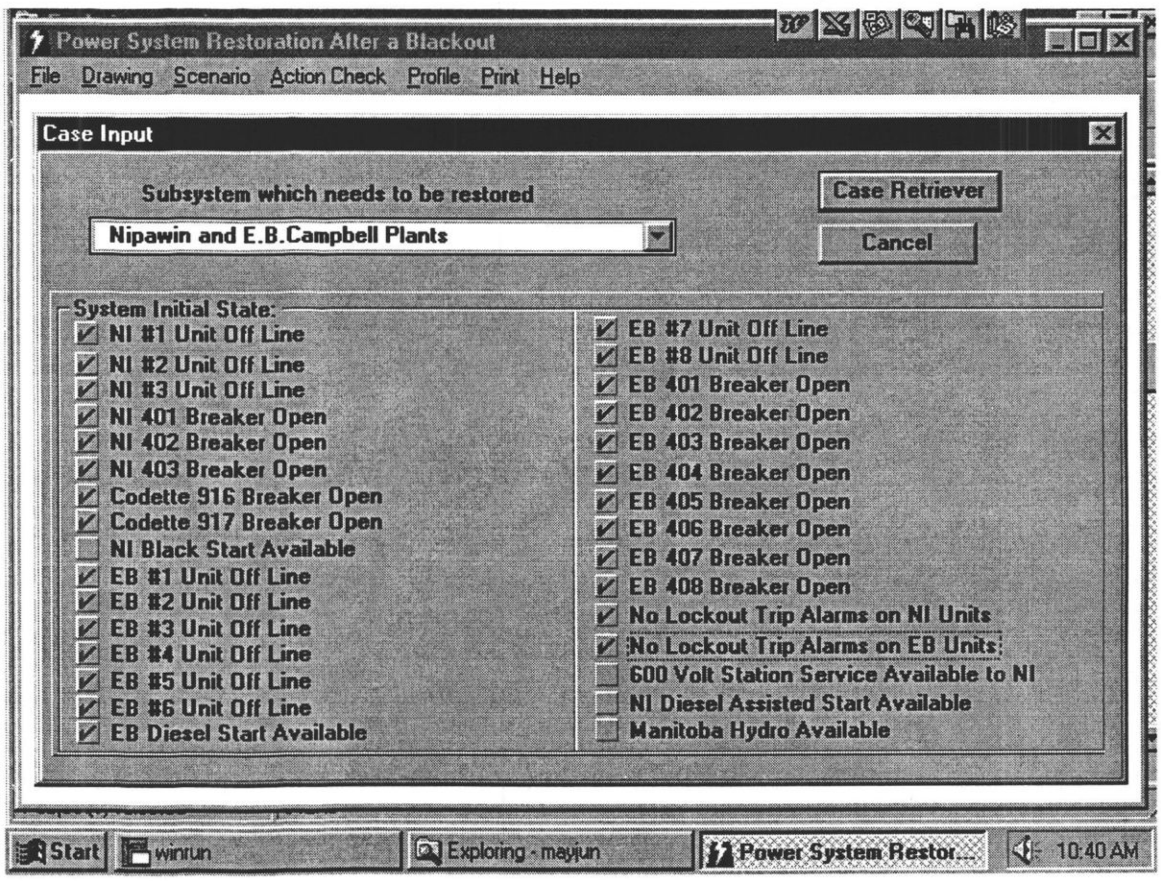
- [x] NI #1 Unit Off Line
- [x] NI #2 Unit Off Line
- [x] NI #3 Unit Off Line
- [x] NI 401 Breaker Open
- [x] NI 402 Breaker Open
- [x] NI 403 Breaker Open
- [x] Codette 916 Breaker Open
- [x] Codette 917 Breaker Open
- [x] NI Black Start Available
- [x] EB #1 Unit Off Line
- [x] EB #2 Unit Off Line
- [x] EB #3 Unit Off Line
- [x] EB #4 Unit Off Line
- [x] EB #5 Unit Off Line
- [x] EB #6 Unit Off Line
- [x] EB Diesel Start Available
- [x] EB #7 Unit Off Line
- [x] EB #8 Unit Off Line
- [x] EB 401 Breaker Open
- [x] EB 402 Breaker Open
- [x] EB 403 Breaker Open
- [x] EB 404 Breaker Open
- [x] EB 405 Breaker Open
- [x] EB 406 Breaker Open
- [x] EB 407 Breaker Open
- [x] EB 408 Breaker Open
- [x] No Lockout Trip Alarms on NI Units
- [x] No Lockout Trip Alarms on EB Units
- [x] 600 Volt Station Service Available to NI
- [x] NI Diesel Assisted Start Available
- [ ] Manitoba Hydro Available

Figure 5.14 Case Description for Situation 4


Power System Restoration After a Blackout
File Drawing Scenario Action Check Profile Print Help
E.B.CAMPBELL
CODETTE
NIPAWIN
Restoration Operation Sequence:
OK
Start winrun Exploring - mayjun Power System ... Microsoft Word - ... 1:42 PM

600 volt station service is available:
1. Normal start a unit, generator breaker will remain open,
   Adjust unit Kv to 13.5KV (.98 p.u.).
2. Normal start the second unit,
   Adjust unit Kv to 13.5 Kv (0.98 p.u.).
   Place transformer tap changer on nominal tap #9.
   Close the unit breaker.
   Initial a " close" on generator breaker.
3. Make alive CE 917T on nominal tap.
   ----- This makes alive CE station service.
4. Adjust 72 KV voltage to 72 KV .

Figure 5.15 Restoration Proposal for Situation 4

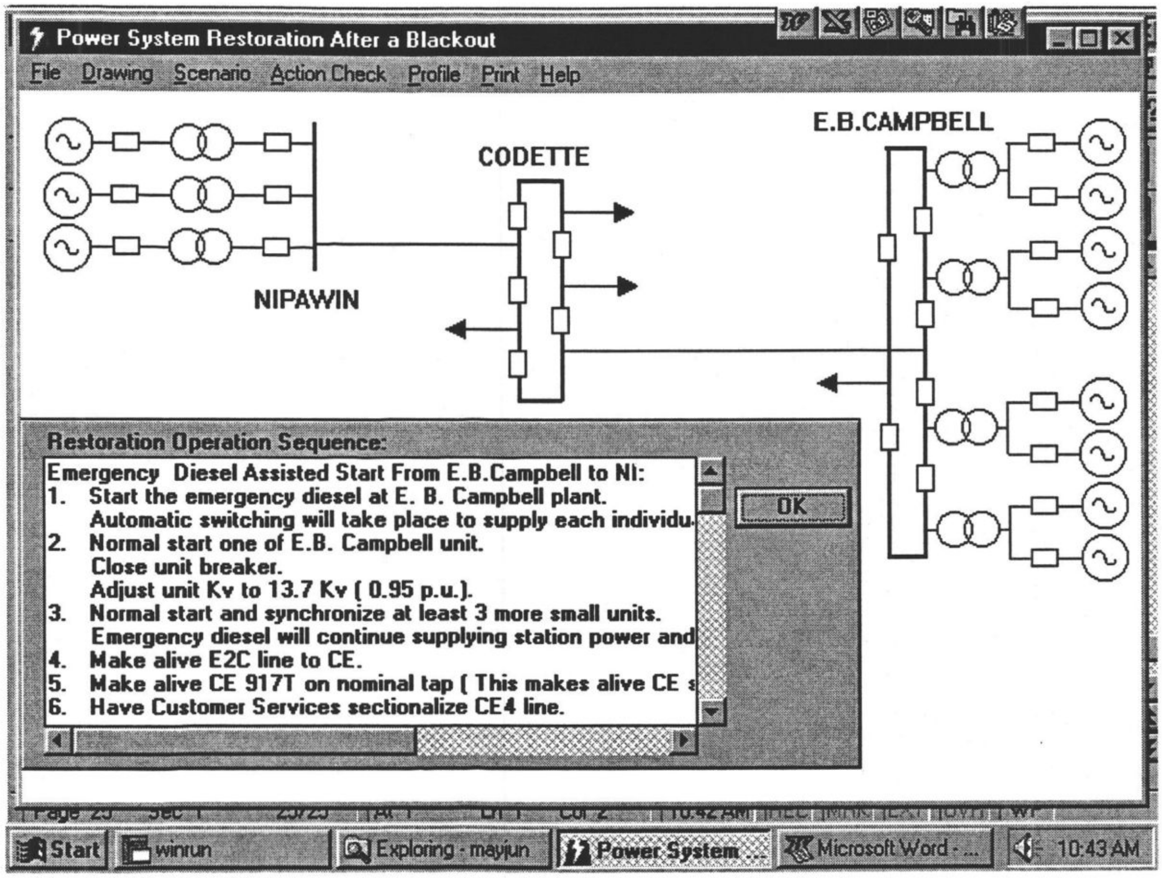

Power System Restoration After a Blackout
File Drawing Scenario Action Check Profile Print Help
Case Input
Case Retriever
Cancel
Start
winrun
Exploring - mayjun
Power System Restor...
10:40 AM

Subsystem which needs to be restored: Nipawin and E.B.Campbell Plants

System Initial State:

- [✓] NI #1 Unit Off Line
- [✓] NI #2 Unit Off Line
- [✓] NI #3 Unit Off Line
- [✓] NI 401 Breaker Open
- [✓] NI 402 Breaker Open
- [✓] NI 403 Breaker Open
- [✓] Codette 916 Breaker Open
- [✓] Codette 917 Breaker Open
- [ ] NI Black Start Available
- [✓] EB #1 Unit Off Line
- [✓] EB #2 Unit Off Line
- [✓] EB #3 Unit Off Line
- [✓] EB #4 Unit Off Line
- [✓] EB #5 Unit Off Line
- [✓] EB #6 Unit Off Line
- [✓] EB Diesel Start Available
- [✓] EB #7 Unit Off Line
- [✓] EB #8 Unit Off Line
- [✓] EB 401 Breaker Open
- [✓] EB 402 Breaker Open
- [✓] EB 403 Breaker Open
- [✓] EB 404 Breaker Open
- [✓] EB 405 Breaker Open
- [✓] EB 406 Breaker Open
- [✓] EB 407 Breaker Open
- [✓] EB 408 Breaker Open
- [✓] No Lockout Trip Alarms on NI Units
- [✓] No Lockout Trip Alarms on EB Units
- [ ] 600 Volt Station Service Available to NI
- [ ] NI Diesel Assisted Start Available
- [ ] Manitoba Hydro Available

Figure 5.16 Case Description for Situation 5



Restoration Operation Sequence:

Emergency Diesel Assisted Start From E.B.Campbell to NI:
1. Start the emergency diesel at E. B. Campbell plant.
   Automatic switching will take place to supply each individu
2. Normal start one of E.B. Campbell unit.
   Close unit breaker.
   Adjust unit Kv to 13.7 Kv (0.95 p.u.).
3. Normal start and synchronize at least 3 more small units.
   Emergency diesel will continue supplying station power and
4. Make alive E2C line to CE.
5. Make alive CE 917T on nominal tap (This makes alive CE s
6. Have Customer Services sectionalize CE4 line.

Figure 5.17 Solution Proposal for Situation 5

6. ACTION CHECKER

6.1 Introduction

The purpose of the action checker is to provide a user with the required information on impacts of restoration actions on the system in terms of power injection, voltage magnitude and phase angle of each bus, active and reactive power through each line. This function can be called through the graphical environment by selecting the "Action Checker" command on the menu bar of the ITA. The action checker consists of three components: topology processor, load flow and limits checker. The network topology processor is triggered to form network information which provides the required data for the load flow program and limits checker then examines the numerical solutions sent by the load flow program. The results can be viewed by selecting the appropriated menus. The block diagram of an action checker is shown in Figure 6.1.

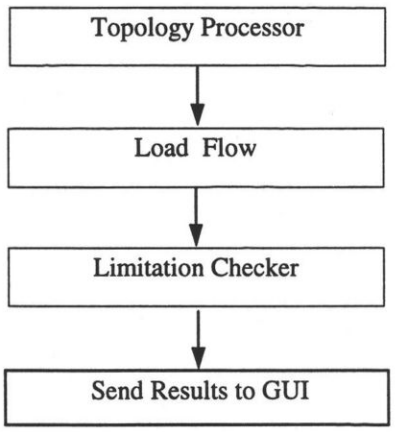
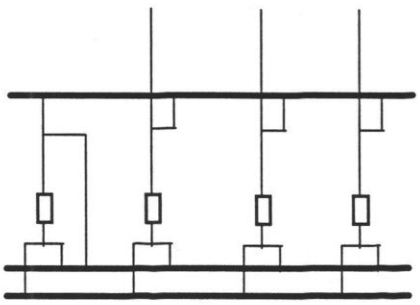
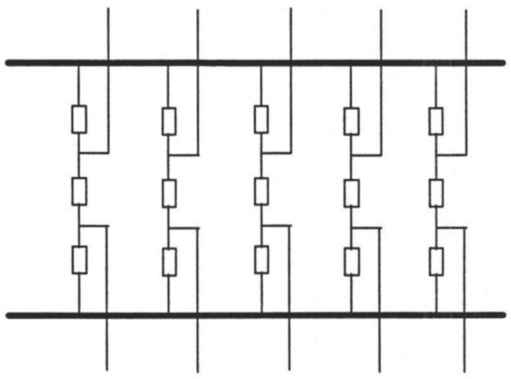


Figure 6.1 Configuration of the Action Checker

6.2 Topology Processor

The single-line diagrams that have been used so far in this research are based on bus/breaker representations. The three typical bus/breaker connections of a substation are shown in Figure 6.2. Most power system analysis calculation programs, however, utilize bus/branch representations as shown in Figure 6.3. The function of the topology processor is to map the bus/breaker representation which has been used to display single-line diagrams on the screen into bus/branch representations which will be used in load flow calculations. Basically, the mapping process of topology is to form logical buses based on the status of the breakers in the system.


(a) Double busbar with auxiliary busbar arrangement


(b) A breaker-and-a-half busbar arrangement

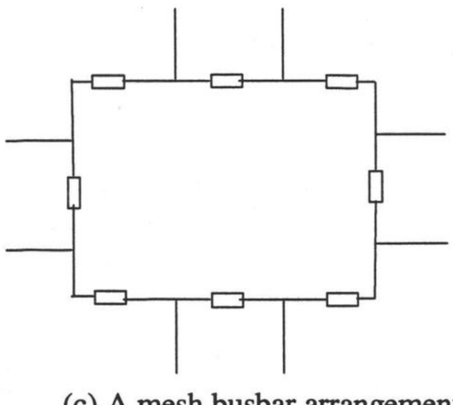

(c) A mesh busbar arrangement

Figure 6.2 Bus/Breaker Representation

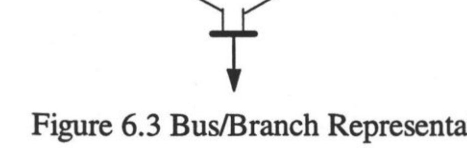


Figure 6.3 Bus/Branch Representation

Based on the typical bus/breaker connections of a substation which is shown in Figure 6.2, it can be deduced that a change in the state of a breaker can result in one of the following conditions:

1. There is no change in electrical connection, and the number of nodes for calculation remains the same.
2. Although some actions have taken place such as some load have been picked up or cut off, generators have been synchronized or cut off, the number of nodes remain the same.
3. Buses have been separated or combined and, as a result, the number of nodes will be increased or decreased.

Both a transmission line and a transformer have two ends which are connected to other system elements, and a generator and a load have one end. Therefore, a transmission line and a transformer are considered as two node elements, while a generator and a load are considered as one node elements in the research. Elements connected through closed circuit breaker are considered to be connected to the same bus which is a connection of nodes. Elements on opposite sides of an open circuit breaker are not connected to the same bus (unless another path between the two sides exists). This can be seen in Figure 6.4 by looking at the bus/breaker representation of a substation and the resulting bus/branch representation created by the topology processor.

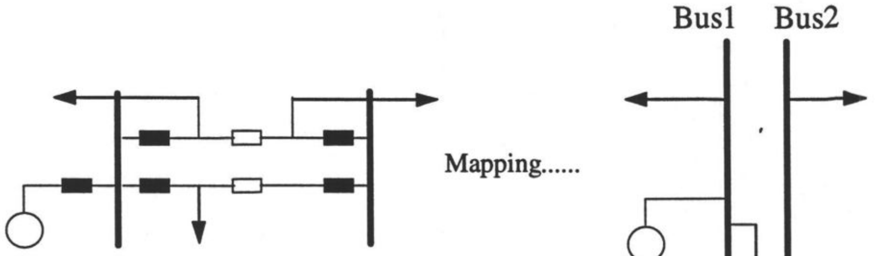


Figure 6.4 Topology Mapping Procedure

6.3 Load Flow

After the topology processor finishes the mapping process, the load flow program is triggered to calculate system states. It provides active and reactive power flows of each line, voltage magnitude and phase angle of each bus for the specified subsystem under operating conditions achieved by restorative actions. The formation of load flow equations and their solution techniques are illustrated in Appendix B.

6.4 Checking Limits

The results of a load flow calculation provide the system profile which include: voltage magnitude and phase angle at each bus; active and reactive power injections at each bus; active and reactive power flows through each line. Since a load flow calculation may converge at a state that violates some system operation limits, it is necessary to check if there are any violations on the restored system. The outcome of a restoration action should be transmitted to the ITA and the user. The limit checking component of the Action Checker compares the calculated values with the specified limits of the system components, and shows the information of any violation. The Action Checker consists of three kinds of checks as shown in Figure 6.5.

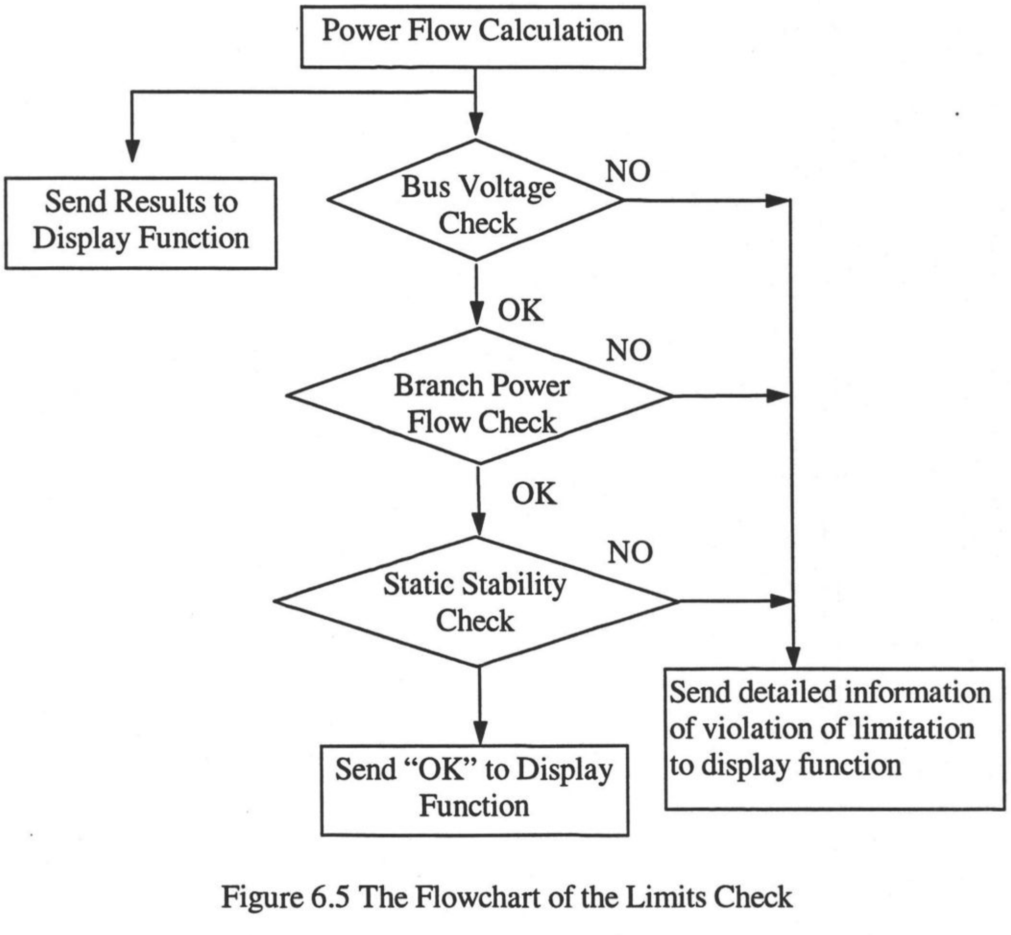

Power Flow Calculation
Send Results to Display Function
Bus Voltage Check
NO
OK
Branch Power Flow Check
NO
OK
Static Stability Check
NO
Send "OK" to Display Function
Send detailed information of violation of limitation to display function

Figure 6.5 The Flowchart of the Limits Check

6.4.1 Bus Voltage Magnitude Check

When a system is at a restorative state, one or more of the following conditions may exist: reduced generation capability, reduced transmission capability, and lightly loaded conditions. The lightly loaded condition can cause excessive voltage rise, particularly when energizing long overhead lines due to the line charging effects. The electric equipment of a power system are designed to function at the nominal voltage of the network to which they are connected. Certain voltage tolerance must be incorporated in the equipment so that small deviations from the nominal voltage will not cause breakdown or malfunction. Excessive high voltage, however, can cause damage to essentially all electric equipment including lines, transformers, generators, circuit breakers and utilization equipment and low voltage will mainly cause damage to motors, both on the utility system and in home application, such as refrigerators, air-conditioners, and freezers. During normal conditions, standards dictate that system voltage levels be maintained between plus or minus 5% of the equipment voltage rating. During emergency conditions, plus or minus 10% is allowed. It is, therefore, important to ensure that the bus voltage magnitudes $|V|$ is normally constrained by

$$|V_{i\,\min}| \leq |V_i| \leq |V_{i\,\max}| \tag{6.1}$$

Where $|\ V_{imin}\ |$ and $|\ V_{imax}\ |$ are the upper and lower limits of the bus voltage magnitudes which are allowed during restoration.

6.4.2 Branch Power Flow Check

Although a system mostly faces a lightly loaded condition during a restoration stage, it is still necessary to check the power flow through the transmission line against the maximum power capacity limit since the distribution of active power in the network may be uneven at the restorative state. The equation which is used for this purpose is:

$$P_{ij} \leq P_{ij\max} \tag{6.2}$$

Where P_{ijmax} is transmission line power flow limit which is related to the type of transmission line and the ambient temperature of the air surrounding the line.

6.4.3 Static Stability Check

The power transfer through a transmission line is constrained by the static stability limit of the transmission line. In this research, the maximum power transfer has not been used as a criterion for the static stability check of transmission lines since it is a function of the terminal voltages of the transmission line. Instead, a more simplified method has been used for the static stability check. That is, the difference between the phase angles of two terminal voltages of the line is considered as a criterion. This value is defined as power limit angle at which the maximum power transfer occurs. The power limit angle can be deduced in the following manner.

It can be noticed from Figure 6.6 that the complex power through a transmission line from *i*th bus to *j*th bus is equal to the product of the voltage at *i*th bus and the conjugate of the current through the line.

$$\overline{S}_{ij} = P_{ij} + jQ_{ij} = \overline{V_i}\,\overset{*}{\overline{I}}_{ij} = \overline{V_i}\left[\frac{\overline{V_i} - \overline{V_j}}{R_{ij} + jX_{ij}}\right]^*$$

$$= \frac{|V_i|^2\left(R_{ij} + jX_{ij}\right)}{R_{ij}^2 + X_{ij}^2} - \frac{|V_i||V_j|\left(Cos\theta_{ij} + jSin\theta_{ij}\right)}{R_{ij}^2 + X_{ij}^2}\left(R_{ij} + jX_{ij}\right) \tag{6.3}$$

The active power through the line is:

$$P_{ij} = \frac{|V_i|^2 R_{ij}}{R_{ij}^2 + X_{ij}^2} - \frac{|V_i||V_j|\left(R_{ij}Cos\theta_{ij} - X_{ij}Sin\theta_{ij}\right)}{R_{ij}^2 + X_{ij}^2} \tag{6.4}$$

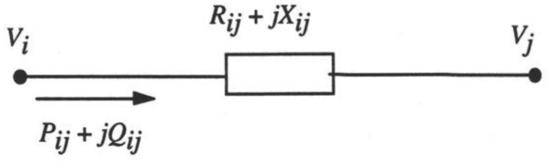


Figure 6.6 A Transmission Line through ith Bus to jth Bus

In order to find the power limit angle of the line, the derivative of the active power flow with respect to voltage phase angle difference should be set to zero as shown in Equation (6.5).

$$\frac{\partial P_{ij}}{\partial \theta_{ij}} = \frac{|V_i||V_j|\left(R_{ij} Sin\theta_{ij} + X_{ij} Cos\theta_{ij}\right)}{R_{ij}^2 + X_{ij}^2} = 0 \tag{6.5}$$

Therefore, the power limit angle can be obtained as follows:

$$\tan\theta_{ij} = \frac{-X_{ij}}{R_{ij}} \tag{6.6}$$

and

$$\theta_{ij\max} = \pi - \tan^{-1}\left(\frac{X_{ij}}{R_{ij}}\right) \tag{6.7}$$

Thus, the static stability of the line can be checked by using

$$\theta_i - \theta_j \le \theta_{ij\max} \tag{6.8}$$

Where θ_i and θ_j are the voltage phase angles of buses *i* and *j*, respectively, and $\theta_{ij\max}$ is the power limit angle of the transmission line which is constant and only related to the line parameters.

6.5 Simulation Results

The Action Checker has been applied to test a five-bus network. The system parameters and operation condition are shown as below.

1. Line Parameters:

From Bus	To Bus	R	X	Yc/K
1	2	0.04	0.25	0.25
1	3	0.1	0.35	0.0
2	3	0.08	0.30	0.25
2	4	0.0	0.015	1.05
3	5	0.0	0.03	1.05

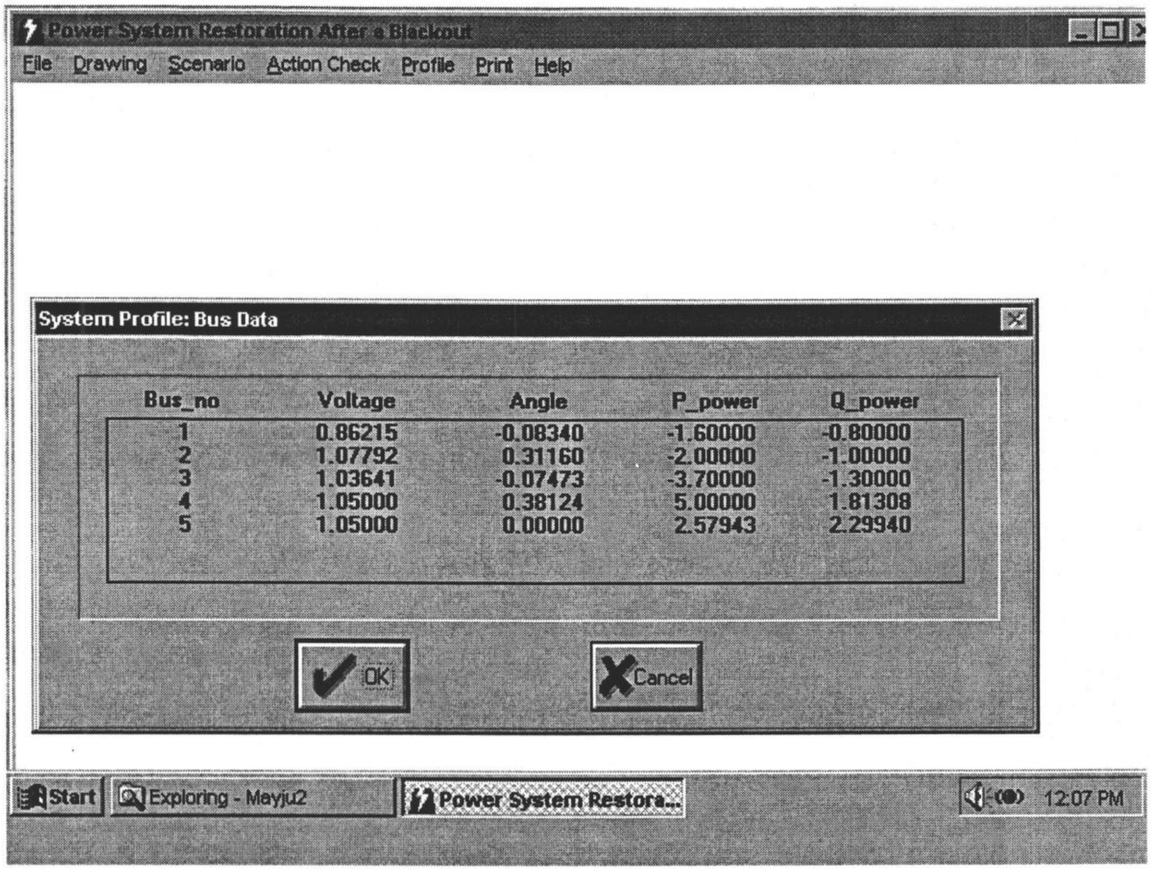
2. Operation Condition:

Bus Injection	P (p.u.)	Q (p.u.)	V (p.u.)	θ (p.u.)
1	-1.6	-0.8		
2	-2.0	-1.0		
3	-3.7	-1.3		
4	5.0		1.05	
5			1.05	0.0

A negative sign before an injection represents that injection is a load. The results of the Action Checker are shown in the Figures 6.7 to 6.11. The bus injection, magnitude and phase angles of bus voltages are shown in Figure 6.7. The real and reactive power through each line are shown in Figure 6.8. The results from branch active power check, line stability check and bus voltage magnitude check are shown in Figures 6.9 ~6.11.

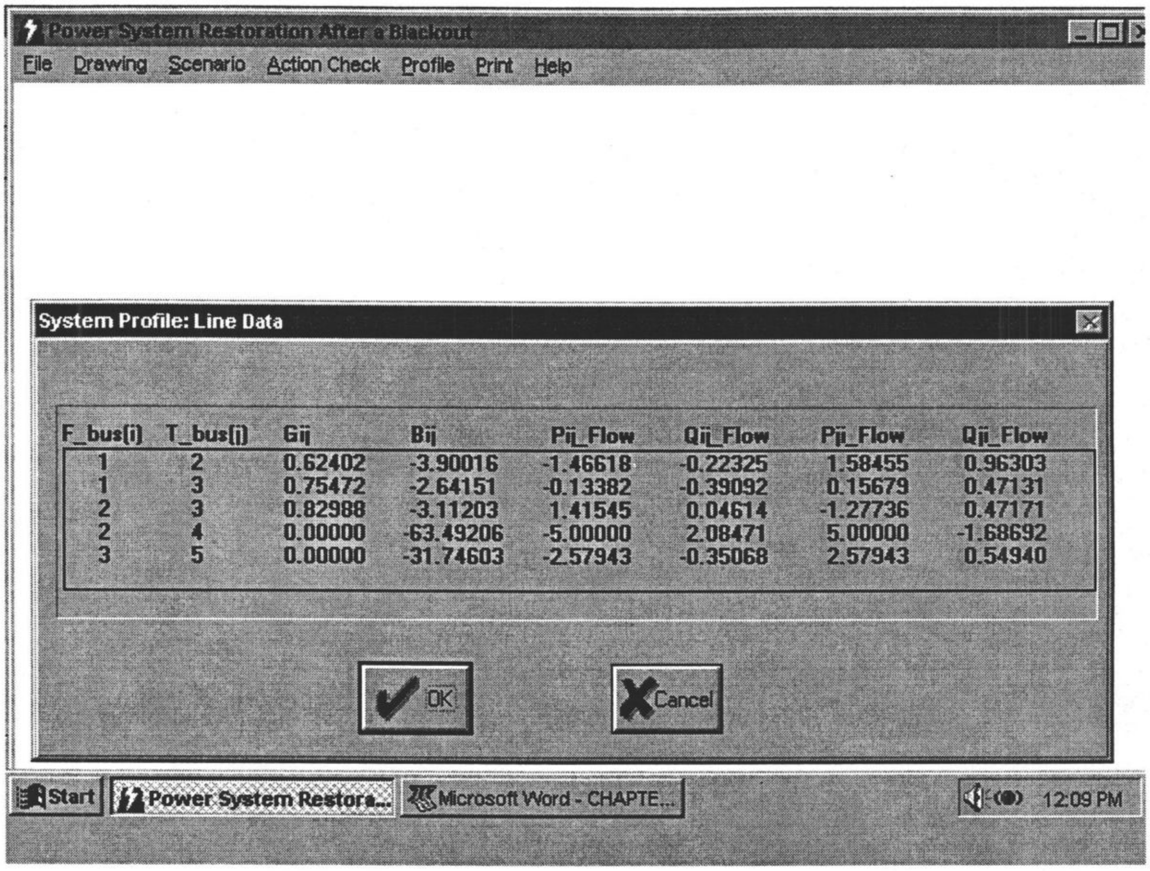
6.6 Summary

The functions of the action checker have been presented in this chapter. The action checker verifies restoration sequences proposed by the CBR. It evaluates the solution by applying analytical methods to calculate load flow and check the system profile. By providing the cause-effect to the user and the CBR, it helps the user to understand the restoration concept, in the sense of what action is good and what is wrong. It also provides necessary information to the CBR to make the CBR improve its learning ability.


Power System Restoration After a Blackout
File Drawing Scenario Action Check Profile Print Help
System Profile: Bus Data
OK
Cancel
Start
Exploring - Mayju2
Power System Restora...
12:07 PM

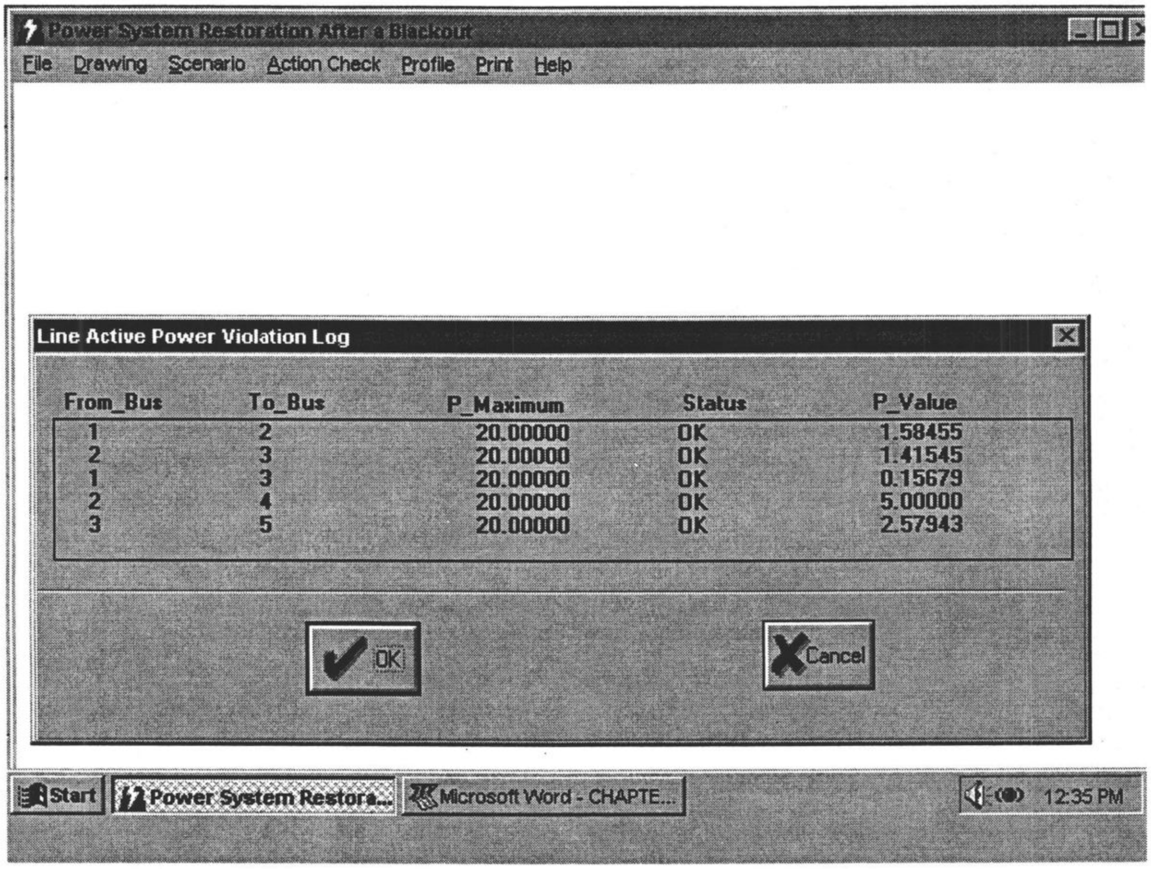
Bus_no	Voltage	Angle	P_power	Q_power
1	0.86215	-0.08340	-1.60000	-0.80000
2	1.07792	0.31160	-2.00000	-1.00000
3	1.03641	-0.07473	-3.70000	-1.30000
4	1.05000	0.38124	5.00000	1.81308
5	1.05000	0.00000	2.57943	2.29940

Figure 6.7 The System Profile


Power System Restoration After a Blackout
File Drawing Scenario Action Check Profile Print Help
System Profile: Line Data
OK
Cancel
Start
Power System Restora...
Microsoft Word - CHAPTE...
12:09 PM

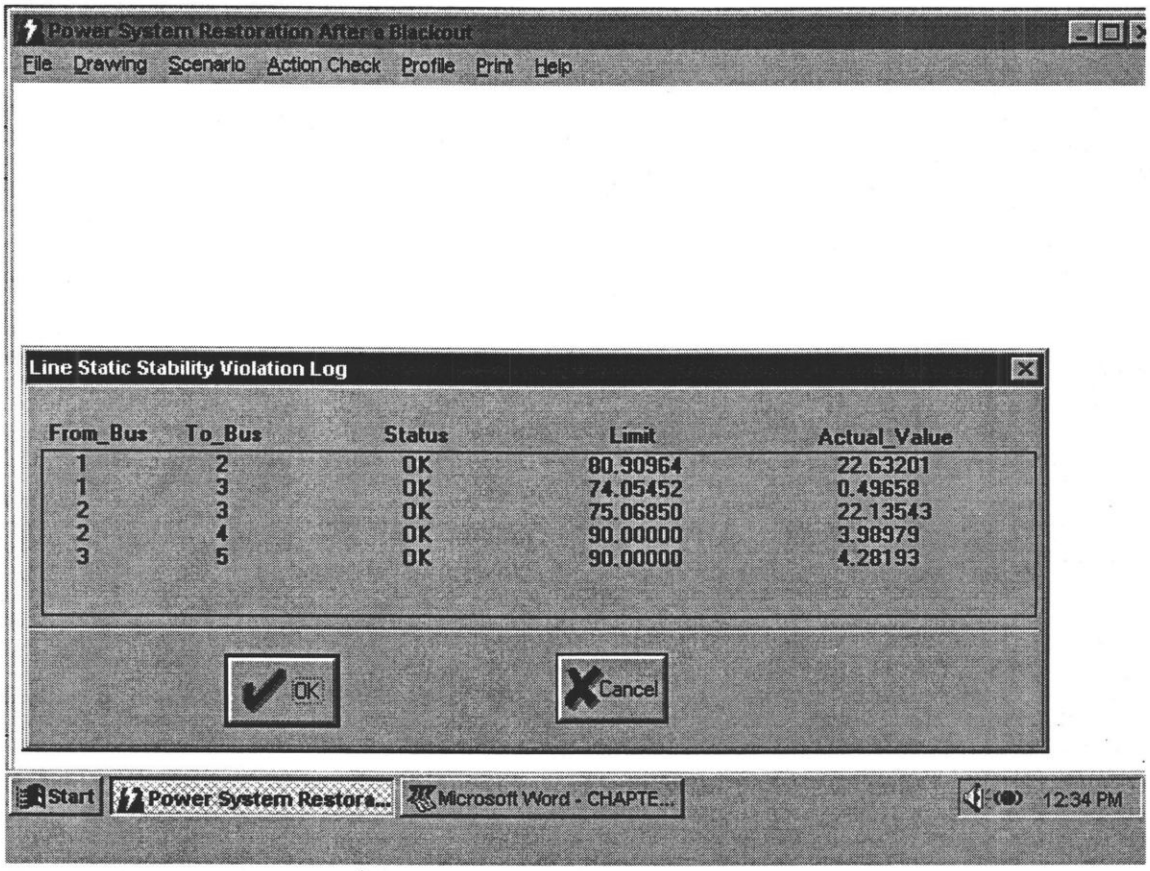
F_bus(i)	T_bus(j)	Gij	Bij	Pij_Flow	Qij_Flow	Pji_Flow	Qji_Flow
1	2	0.62402	-3.90016	-1.46618	-0.22325	1.58455	0.96303
1	3	0.75472	-2.64151	-0.13382	-0.39092	0.15679	0.47131
2	3	0.82988	-3.11203	1.41545	0.04614	-1.27736	0.47171
2	4	0.00000	-63.49206	-5.00000	2.08471	5.00000	-1.68692
3	5	0.00000	-31.74603	-2.57943	-0.35068	2.57943	0.54940

Figure 6.8 Line Flow


Power System Restoration After a Blackout
File Drawing Scenario Action Check Profile Print Help
Line Active Power Violation Log
OK
Cancel
Start
Power System Restora...
Microsoft Word - CHAPTE...
12:35 PM

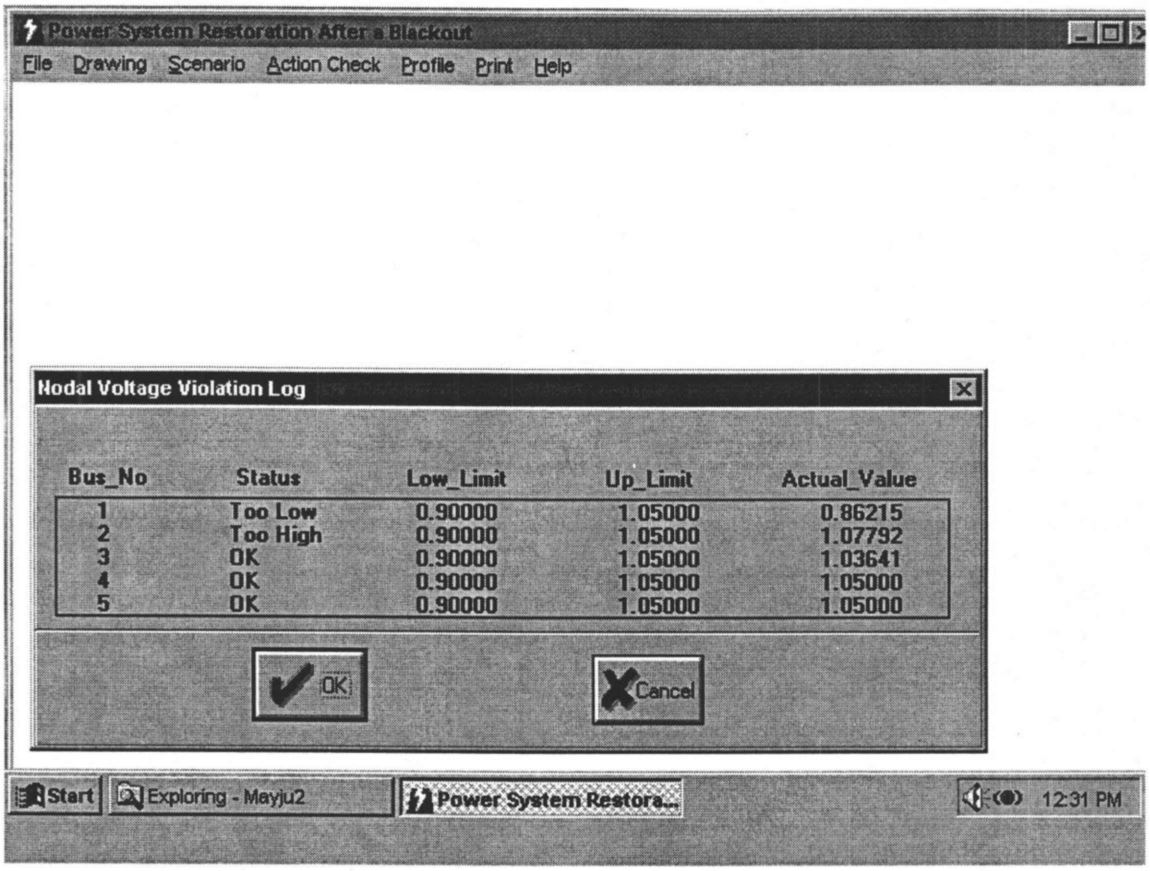
From_Bus	To_Bus	P_Maximum	Status	P_Value
1	2	20.00000	OK	1.58455
2	3	20.00000	OK	1.41545
1	3	20.00000	OK	0.15679
2	4	20.00000	OK	5.00000
3	5	20.00000	OK	2.57943

Figure 6.9 Line Active Power Violation Log


Power System Restoration After a Blackout
File Drawing Scenario Action Check Profile Print Help
Line Static Stability Violation Log
OK
Cancel
Start
Power System Restora...
Microsoft Word - CHAPTE...
12:34 PM

From_Bus	To_Bus	Status	Limit	Actual_Value
1	2	OK	80.90964	22.63201
1	3	OK	74.05452	0.49658
2	3	OK	75.06850	22.13543
2	4	OK	90.00000	3.98979
3	5	OK	90.00000	4.28193

Figure 6.10 Line Static Stability Violation Log


Power System Restoration After a Blackout
File Drawing Scenario Action Check Profile Print Help
Nodal Voltage Violation Log
OK
Cancel
Start
Exploring - Mayju2
Power System Restora...
12:31 PM

Bus_No	Status	Low_Limit	Up_Limit	Actual_Value
1	Too Low	0.90000	1.05000	0.86215
2	Too High	0.90000	1.05000	1.07792
3	OK	0.90000	1.05000	1.03641
4	OK	0.90000	1.05000	1.05000
5	OK	0.90000	1.05000	1.05000

Figure 6.11 Bus Voltage Violation Log

7. SUMMARY AND CONCLUSIONS

The objectives of PSR training are to let system operators become familiar with restoration steps, grasp fundamental principles and improve their skills of handling activities related to the restoration process. An ITA has been developed in this research to meet these objectives. The ITA consists of three components: the GUI, the CBR and the action checker. The GUI provides the essential foundation for building a restoration training program. It displays the restoration plan and the single-line diagrams in the same interface, and therefore, the common reference for easier understanding and communication is established. The CBR mechanism is applied to draw the restoration plan based on the postdisturbance condition and the experiences that have been accumulated in the case library. The action checker verifies the restoration plan and provides a set of results from analytical calculations. By viewing the feedback from the action checker, a trainee can have a better feeling about cause and effect and his/her learning ability can be significantly enhanced. At the same time, the feedback to the CBR allows the CBR to improve its reasoning ability over time. As well, the ITA provides the training methods which are student-paced and scheduled. There is no inherent need to coordinate and schedule a class with an instructor and other students. Since system operators typically work on rotating shifts, the ability to allow a student to schedule a training session at odd hours or when opportunities arise is invaluable.

In Chapter 1, the overall area of PSR training is introduced and different techniques which have been used so far in the area are summarized. The advantages and potentials of a new approach as used in this research work are presented and the architecture of the ITA is discussed.

Chapter 2 introduces the Fusion method, one of the Object-Oriented approaches. In the Fusion, the software development is divided into three stages: OOA, OOD, OOI. The concepts, notations and methodologies related to each stages are introduced. The application of the Fusion to develop the GUI is presented in Chapter 3.

The implementation of the GUI is discussed in a step by step manner. Inheritance, encapsulation and polymorphism techniques of object-oriented approach have been utilized to characterize power system information. Power system elements are categorized into inheritance graphs of class relationships. The GUI provides the foundation for a user to interact with the ITA, without which, the basic interaction would be impossible. The pertinent data are often more easily understood when the information is presented by graphics as opposed to pure numerical presentations in tabular forms. An object-oriented approach has proved to be a suitable technique for developing the GUI. The natural modularity of the class structure makes it easier to contain the effects of changes, and the use of inheritance reduces the number of disparate concepts needed to understand the code. The program tends to be easier to extend. Inheritance enables new classes to be built from old ones while still participating in all the original relationships. The classes form a loosely coupled structure that is easier to modify.

The CBR methodology is introduced in Chapter 4 and each functionality of the CBR, which includes the construction of the case base, retrieval, evaluation, adaptation/modification of the cases as well as the presentation of domain knowledge is discussed in detail. Chapter 5 illustrates how the restoration knowledge is represented in case by using case language. The greatest match algorithm has been used to make comparisons between the cases in the case library and the problem at hand. The solution from the best matched case is applied to solve the problem after suitable adaptation. The CBR has been embedded in the ITA to generate restoration plans. The simulations of many cases had been carried out using the SaskPower system. The restoration plan of a whole network has been decomposed into a set of subplans referred to subsystem restoration. This decomposition has reduced the retrieval time of the CBR. The major observation as illustrated in Chapter 5 is that a restoration plan can be developed with the help of a CBR system with little difficulty. A CBR system overcomes the inherent difficulty of modelling the steps of power system restoration by using analytical techniques and rule-based ES systems. A concrete guide to the development of a case library in which cases should be registered has been obtained.

Chapter 6 illustrates how the action checker of the ITA has been implemented to verify restoration sequence which is proposed by the CBR. The decoupled NR load flow algorithm has been used to calculate the system profile and the limits checking algorithms are used to compare the postrestoration state of the system with certain operating criteria. By displaying the results of load flows and operating limits check, action checker provides benefits to both trainees and the CBR in following aspects:

I. Since case-based reasoning integrates reasoning and learning, it is not enough for a case-based reasoner to stop reasoning after it derives a solution. Rather, the feedback must be sent to the CBR. By evaluating the feedback, a case-based reasoning system becomes more reliable and effective.

II. The learning ability of a user is significantly enhanced by knowing the impact of restoration actions on the system.

This research provides insight into some of the issues involved in development of computer-based power system restoration training simulator, and some conclusions deduced from the research work in regard to these issues are presented as follows.

The ITA has the potential for use as a training tool for teaching power system operators to handle system restoration. By providing visualization of the restoration process and the way to interact with a user, the ITA offers a more effective and flexible training method compared with other training tools.

The CBR approach is applicable to generate a power system restoration plan. It provides solutions quickly by avoiding reasoning from scratch. It is difficult to model the power system restoration domain using analytic methods, the CBR approach has shown its merit to propose solutions for restoration. This application area for the CBR approach demonstrates the versatility of the paradigm for use in various domains.

The object-oriented approach has proven to be a useful methodology in pursuing the uniformity throughout the GUI by the use of a mouse-driven manipulation mode. The GUI conforms with the current GUI development trends by providing a way to translate mouse actions into display effects.

The research also provides directions for future research in several areas, including adaptation of retrieved cases for use in solution proposal, and illustration of system dynamic response of restoration actions.

8. REFERENCES

1. North American Electric Reliability Council Operating Committee Restoration Reference Document.
2. D.Coleman, P.Arnold, S.Bodoff, C.Dollin, H.Gilchrist, F.Hayes, P.Jeremes, "**Object-Oriented Development**: The **Fusion method**", Prentice Hall, 1994.
3. T.Sakaguchi, K.Matsumoto, **"Development of a Knowledge Based System for Power System Restoration",** IEEE Transactions on Power Apparatus and Systems, Vol. Pas-102, No.2, pp. 320-329, February 1983.
4. C.C. Liu, S.J. Lee, S.S. Venkata, **"An Expert System Operational Aid for Restoration and Loss Reduction of Distribution Systems",** IEEE Transactions on Power Systems, Vol. 3, No. 2, pp. 619-626, May 1988.
5. Z.Z. Zhang, G.S. Hope, O.P.Malik, **"A Knowledge-based Approach to Optimize Switching in Substations",** IEEE Transactions on Power Delivery, Vol. 5, No. 1, January 1990.
6. T.Koima et al. **"Restoration Guidance System for Trunk Line System",** IEEE Transaction on Power Systems, Vol. 4, No. 3, pp. 1228-1235, August 1989.
7. I. Takeyasu et al, **" An Expert System for Fault Analysis and Restoration of Trunk Line Power System",** in Proceedings of the First Symposium on Expert Systems Applications to Power Systems, Stockholm-Helsinki, August 22-26, 1988, pp.8-24 to 8-31.
8. L.R. Blessing, C.K. Bush, S.J. Yak, **"Automated Power System Restoration Incorporating Expert System Techniques",** in Proceedings of the Second Symposium on Expert Systems Applications to Power Systems, Seattle, July 17-20, 1989, pp. 133-139.
9. S. Kambhampati, **"Representational requirements for plan reuse",** in Proceedings of the Second Case-Based Reasoning Workshop, pages 20-23, Pensacola Beach, FL.

10. P. Koton, "**Reasoning about evidence in causal explanations**", in Proceedings of the Case-Based Reasoning Workshop, pages 260-270, Clearwater Beach, FL.
11. J.L. Kolodner, R.L. Simpson, "**The MEDIATOR: analysis of an early case-based problem solver**", Cognitive Science, 13(4):507-549.
12. K.J. Hammond, "**Case-Based Planning: Viewing Planning as a Memory Task**", Boston, MA: Academic Press.
13. R. Barletta, W.Mark, "**Explanation-based indexing of cases**", in Proceedings of the Case-Based Reasoning Workshop, pages 50-60, Clearwater Beach, FL, 1988.
14. K.D. Ashley, "**Indexing and analytic models**", in Proceedings of the Second Case-Based Reasoning Workshop, pages 197-202, Pensacola Beach, FL, 1989.
15. C. Martin, "**Complex indices: A metaphorical example**", in Proceedings of the Second Case-Based Reasoning Workshop, pages 295-299, Pensacola Beach, FL, 1989.
16. C. Martin, "**Indexing using complex features**", in Proceedings of the Second Case-Based Reasoning Workshop, pages 26-30, Pensacola Beach, FL, 1989.
17. C.M. Seifert, "**Goals in reminding**", in Proceedings of the Case-Based Reasoning Workshop, pages 357-369, Clearwater Beach, FL, 1988.
18. M.J. Pazzani, "**Indexing strategies for goal-specific retrieval of cases**", in Proceedings of the Second Case-Based Reasoning Workshop, pages 31-35, Pensacola Beach, FL, 1989.
19. L.J. Kolodner, "**Retrieving events from a case memory: A parallel implementation**", in Proceedings of the Case-Based Reasoning Workshop, pages 233-249, Clearwater Beach, FL, 1988.
20. E. Domeshek, "**Parallelism for index generation and reminding**", in Proceedings of the Second Case-Based Reasoning Workshop, pages 244-247, Pensacola Beach, FL, 1989.
21. K.D. Ashley, "**Assessing similarities among cases: A position paper**", in Proceedings of the Second Case-Based Reasoning Workshop, pages 72-76, Pensacola Beach, FL, 1989.

22. R. Alterman, "**A concept space for reasoning about cases involving event structure**", in Proceedings of the Second Case-Based Reasoning Workshop, pages 16-19, Pensacola Beach, FL, 1989.

23. R.C. Schank, "**Dynamic Memory: A Theory of Learning in Computers and People**", New York: Cambridge University Press, 1982.

24. D. Gentner, "**Finding the needle: Accessing and reasoning from prior cases**", in Proceedings of the Second Case-Based Reasoning Workshop, pages 137-143, Pensacola Beach, FL, 1989.

25. L. Whitaker, S. Wiggins, G. Klein, "**Using qualitative or multi-attribute similarity to retrieve useful cases from a case base**", in Proceedings of the Second Case-Based Reasoning Workshop, pages 345-347, Pensacola Beach, FL, 1989.

26. T.R. Hinrichs, "**Towards an architecture for open world problem solving**", in Proceedings of the Case-Based Reasoning Workshop, pages 182-189, Clearwater Beach, FL, 1988.

27. R.M. Turner, "**Modifying previously-used plans to fit new situations**", in Proceedings of the Ninth Annual Conference of the Cognitive Science Society, pages 1-10, Seattle, WA, 1987.

28. J.A. Hendler, S. Kambhampati, "**Refitting plans for case-based reasoning**", in Proceedings of the Case-Based Reasoning Workshop, pages 179-181, Clearwater Beach, FL, 1988.

29. R. Barletta, D. Hennessy, "**Case adaptation in autoclave layout design**", in Proceedings of the Second Case-Based Reasoning Workshop, pages 203-207, Pensacola Beach, FL, 1989.

30. K. Sycara, "**Using case-based reasoning for plan adaptation and repair**", in Proceedings of the Case-Based Reasoning Workshop, pages 425-434, Clearwater Beach, FL, 1988.

31. G. Collins, "**Plan adaptation: A transformational approach**", in Proceedings of the Second Case-Based Reasoning Workshop, pages 90-93, Pensacola Beach, FL, 1989.

32. R. Alterman, "**An adaptive planner**", in Proceedings of the Fifth National Conference on Artificial Intelligence (AAAI-86), pages 65-69, Philadelphia, PA, American Association for Artificial Intelligence, 1986.
33. E.L. Rissland, K.D. Ashley, "**Credit assignment and the problem of competing factors in case-based reasoning**", in Proceedings of the Case-Based Reasoning Workshop, pages 327-344, Clearwater Beach, FL, 1988.
34. D.Chapman "**Planning for Conjunctive Goals**", Artificial Intelligence pp. 333-377.
35. J.L. Kolodner, "**Capitalizing on failure through case-based inference**", in Proceedings of the Ninth Annual Conference of the Cognitive Science Society, pages 715-726, Seattle, WA, 1987.
36. J.L. Kolodner "**Case-Based Reasoning**", Morgan Kaufmann Publisher, Inc., 1994.
37. Brian Stott "**Review of Load-Flow Calculation Methods**", Proc. of the IEEE, Vol. 62, No.7, July 1974.
38. Glenn W.Stagg "**Computer Methods in Power System Analysis**", McGraw-Hill Inc. 1968.
39. Ward, J.B., and H.W.Hale "**Digital Computer Solution of Power-Flow Problems**", Trans. AIEE, vol.75, pt. III, pp. 398-404,1956.
40. K.Walden, J.M.Nerson, "**Seamless Object-Oriented Software Architecture-Analysis and Design of Reliable Systems**", Prentice Hall, 1994.
41. G.Booch, "**Object-Oriented Analysis and Design with Applications**", 2nd edition, Benjamin cummings, Redwood City, 1994.
42. J.Rumbaugh, M.Blaha, W.Premerlani, F.Eddy, W.Lornsen, "**Object-Oriented Modelling and Design**", Prentice Hall International Editions, 1994.
43. S.W.Ambler, "**CRC Modelling: Bridging the Communication Gap between Developers and Users**", Ambysoft Inc., 1994.
44. B.Meyer, "**Object-Oriented Software Construction**", Prentice Hall, 1989.

9. APPENDIX A

A-1 Introduction

This appendix summarizes the notations of Fusion method which have been used in this research. Each notation is discussed in approximately the order in which it is encountered in the process.

In the analysis stage, two model notations are used: the Object Model which serves to describe classes of objects that exist in a system and relationships between those classes; the Interface Model which demonstrates the behaviour of a system and defines the boundary between the system and its environment.

In the design stage, four model notations are used. The Object Interaction Graph describes how objects interact at run-time to support the functionality specified in the operation model. The Visibility Graph is used to specify the object communication paths which show how the objects have access one to another. The Class Description serves to establish the methods, data attributes and object-valued attributes of each class. Lastly, the Inheritance Graph describes the class/subclass inheritance structures.

In the implementation stage, the Class Description for each class is mapped to object-oriented language (in this research , C++ has been used). There are no additional notations besides those that are used in the language. More detailed descriptions of the above mentioned notations are given below:

A-2 Object Model

The object model defines classes in the system and relationships between those classes. Attributes of classes, aggregation, specialization/generalization of classes are produced in the object model. Object model syntax is a diagram whose components are class boxes and relationship diamonds connected by subclass arcs and role arcs.

Classes

A class box is a rectangle containing the name of a class and its attributes. The class name is separated from its attributes by a horizontal line drawn across the box. The class name appears above the line. The class syntax is shown in Figure A-1.

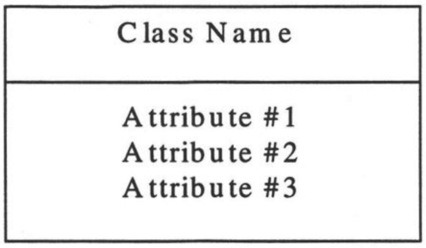


Figure A-1 Class Representation

Aggregation

Aggregation is a mechanism for structuring an aggregate class from the other component classes and models the "part of" or "has a" relationship. The syntax of the aggregation is shown in Figure A-2.

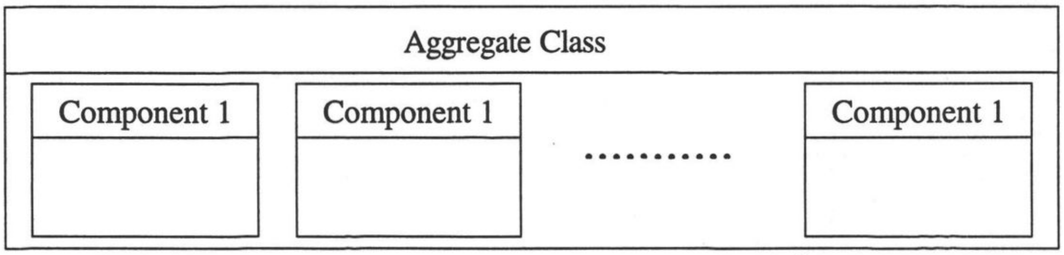


Figure A-2 Aggregation Representation

Generalization

Generalization allows a superclass to be formed by factoring out the common properties of several subclasses. It models the "kind of" or "is a" relationship. In a generalization, the attributes and the relationships of the superclasses are inherited by all the subclasses.

Specialization

Specialization is the converse case of generalization. Multiple specialization allows a new subclass to be defined as a specialization of more than one immediate superclass. This permits information to be mixed from more than one source. The subclass inherits the attributes and relationships of all its superclasses.

The syntax of generalization and specialization is shown in Figure A-3.

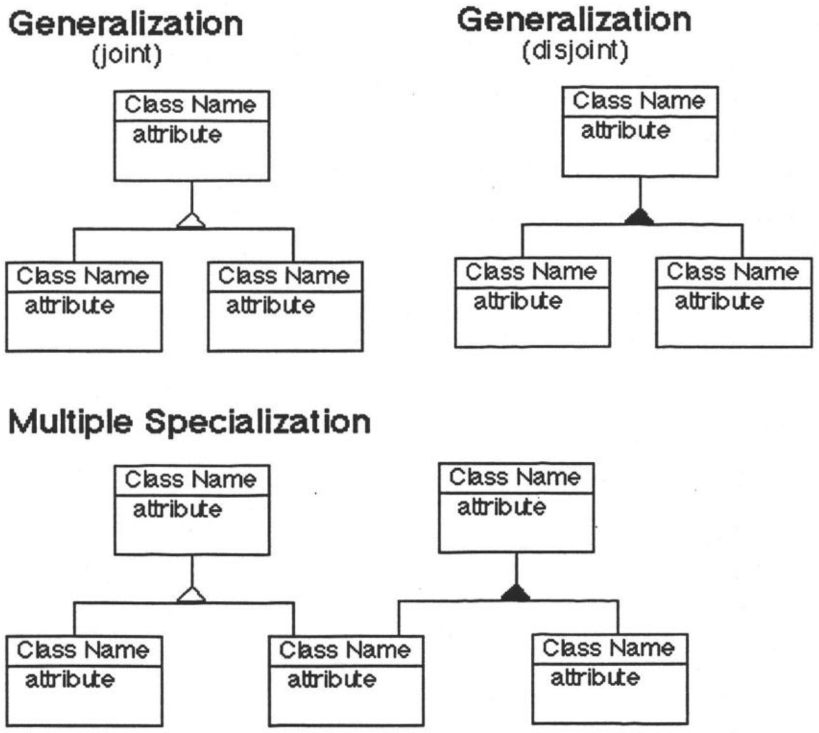


Figure A-3 Generalization & Specialization Syntax

Relationship

Relationship models indicate association or correspondence between the objects belonging to classes. The following points must be considered when a relationship is described.

•Cardinality Constraints

A cardinality is the number of classes that may be associated with each other in a relationship.

•Ternary and Higher Relationships

Ternary relationships relate three separate objects. Relations involving more than three objects are referred to as n-ary relationships. Relationship syntax is shown in Figure A-4.

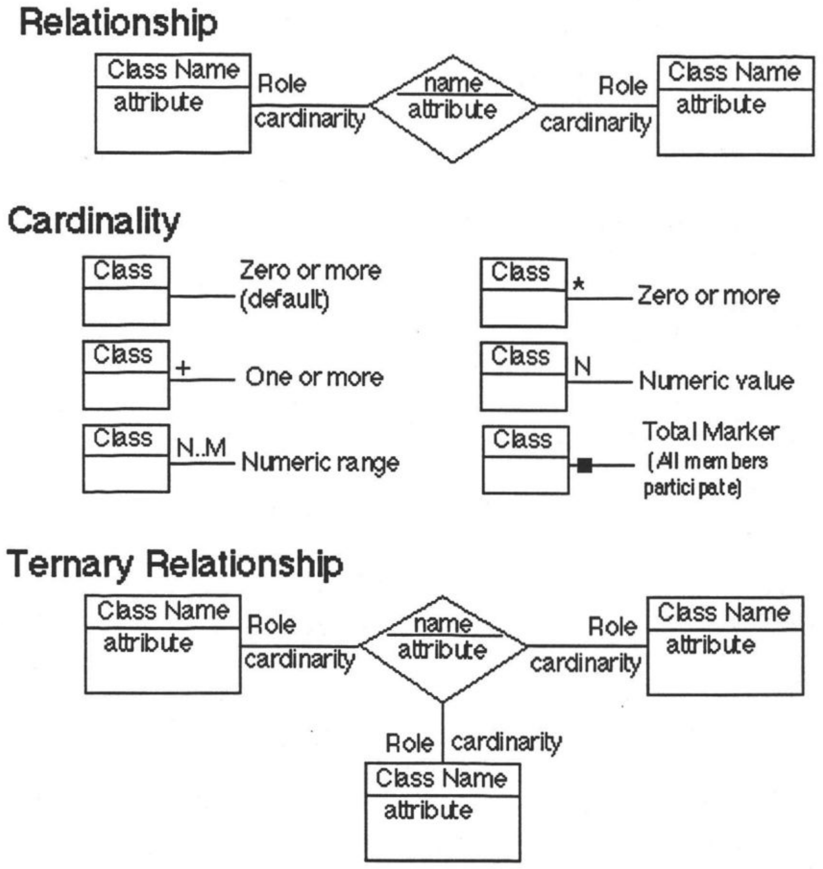
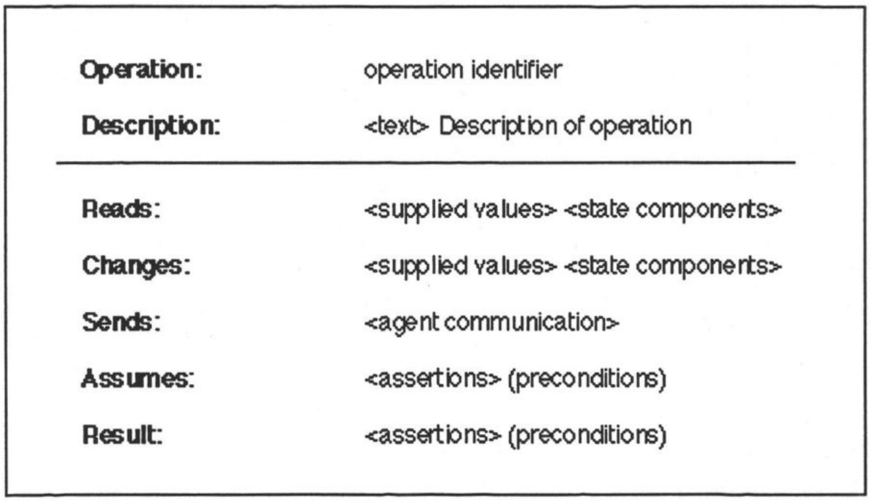


Figure A-4 Relationship Syntax

A-3 Interface Model

The interface model describes the behavior of a system; i.e. it defines the input and output communication of the system. It consists of a collection of system operations. A system operation is an input event and its effect on the system. It may create a new instance of a class, or change the value of an attribute of an existing object, or send an event to the environment.

Each system operation is described by a schema. The syntax of the schema is shown in Figure A-5.



Operation:	operation identifier
Description:	<text> Description of operation
Reads:	<supplied values> <state components>
Changes:	<supplied values> <state components>
Sends:	<agent communication>
Assumes:	<assertions> (preconditions)
Result:	<assertions> (preconditions)

Figure A-5 Operation Schema Syntax

The syntax of a schema consists of several clauses to describe a system operation. The Operation clause is the name of the system operation being described. The Description clause gives an informal summary of the meaning of the schema. The Reads and Changes clauses list objects and attributes. The Sends clause records what events are sent to the environment. The Assumes and Result clauses list conditions.

A-4 Object Interaction Graph

An object interaction graph is a collection of boxes that are linked by arrows. The boxes represent objects, and the arrows represent message passing. There are two kinds of boxes: controller and collaborators.

Controller

Controller has an arrow entering it, the arrow does not come from any other box in the graph. The arrow is labeled with the name of the system operation.

Collaborators

The boxes other than the controller are called collaborators. Every arrow other than the system operation arrow goes from one box to another within the graph.

Message Passing & Sequencing

Message passing is a directed point-to-point communication. The notation of message passing is a directed arrow with labels. The direction of an arrow is from sender(client) to receiver(server).

If a sequence of message passing is important, the order of sequences is shown by introducing sequence labels in parentheses above the message name. For example, (1) (1.1) (1.1.1)...(1.2)...(2) (2.1) and so forth. A sequence label with a prime (i.e., " 1' ") indicates additional constraints on invocations. For example, given the sequence labels (1) and (1'), the message with label (1') will not be invoked if message (1) is invoked and vice versa.

Dynamic Object Creation

The keyword **new** indicates that an object is created as part of the execution of an object interaction graph.

Object interaction graphs are built for each system operation. This means object interaction graphs are developed for each schema in the operation model. The syntax of object interaction graphs is shown in Figure A-6.

A-5 Visibility Graph

The purpose of the visibility graph is to define the reference structure of classes in the system. The notation of a visibility graph is a diagram and its components are client boxes, server boxes, and visibility arrows.

Client Box

Client box represents the class requiring the access. A client box is a rectangle containing the name of a class.

Server Box

Server box represents the object being accessed. A server box is a rectangle containing a server label, which gives the names of the object and the class.

Visibility Arrow

Visibility arrow represents the fact that the client has access to an instance of the server via an access path.

There are four design decisions to be made when considering the reference structure of classes.

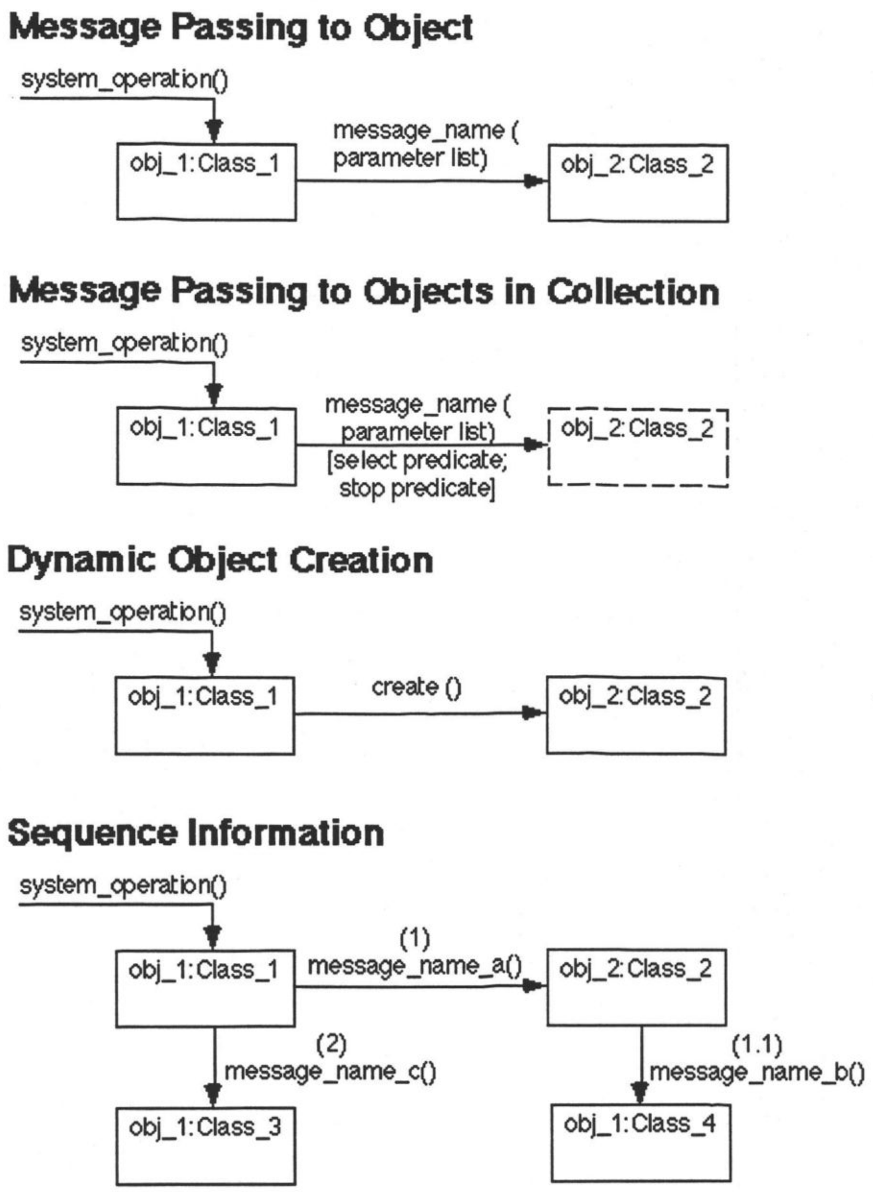


Figure A-6 Message Passing

1. Reference Lifetime

When a client only needs to message a server in the context of a single method invocation, access can be given through a parameter by a local variable of the method. In this situation a dynamic reference is specified. It is denoted by a dashed arrow.

When an object needs the same reference in many contexts, a permanent reference is used. It is denoted by a solid arrow.

2. Server Visibility

A non-shared server object receives a message from a client exclusively. This guarantees that, at the time of any method invocation, only one client has a reference to the server. The server object box has a double border. The dashed outline is used for servers which are a collection of server objects.

A shared server object receives a message from multiple clients. The server object box has a single border.

3. Server Binding

If a server object will be deleted when its client is deleted, it is said that the lifetime of the server is bound to its client. In this case, the box of the bound server is put inside the client box.

When the lifetime of a server object is not bound to the lifetime of a client, the server box is shown outside the client box.

4. Reference Mutability

When a reference has constant mutability, it is not assignable after initialization. The keyword **constant** is prefixed to the server name.

If a reference is reassignable, the mutability is variable.

The summary of notations in Visibility Graph is shown in Figure A-7.

A-6 Class Descriptions

After developing visibility graphs for all classes, the next step is to collate information from the system object model, object interaction graphs, and visibility graphs in class descriptions, one for each class. At this stage, the methods, some data attributes and object- valued attributes for each class are established. The process of developing class description is shown in Figure A-8.

The syntax of a class description is shown in Figure A-9. It consists of a class name, immediate superclass(es), attributes of the class, and the methods provided by the class. The visibility information of attribute qualifiers is documented by appropriate keywords. Mutability of the attribute is given as constant and variable as appropriate.

Mutability defaults to variable when not indicated in the class description. Sharing information can be indicated as shared or exclusive. The default is shared. Binding information is either bound or unbound. The default is unbound.

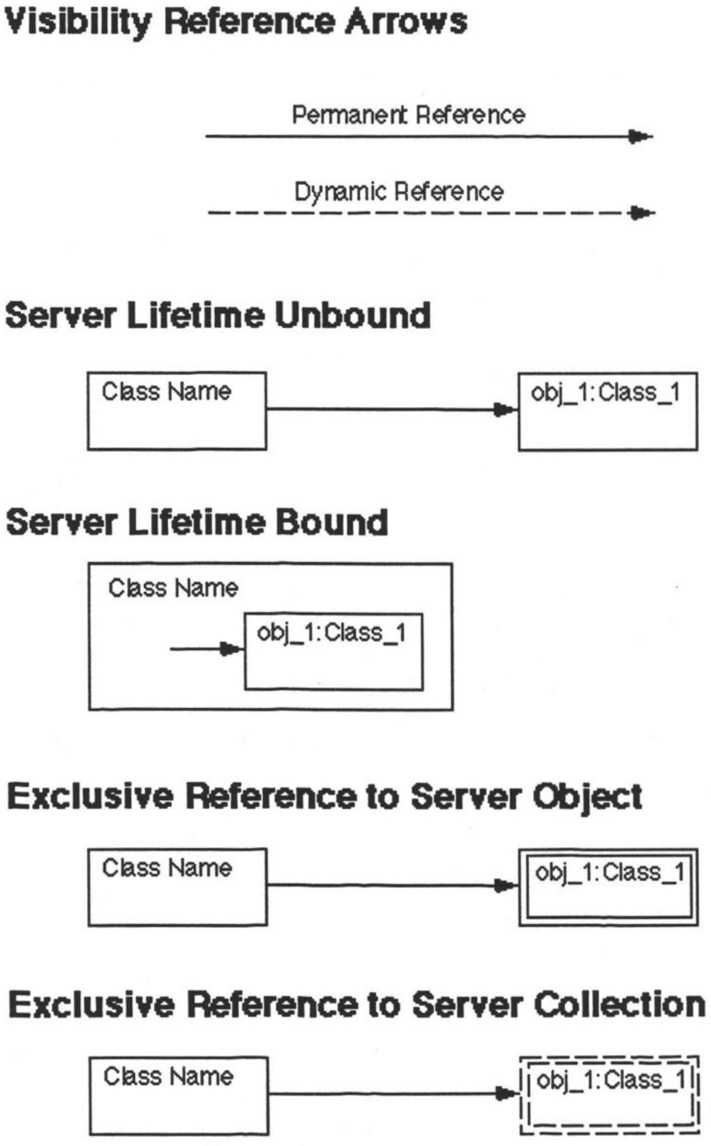


Figure A-7 Visibility Syntax

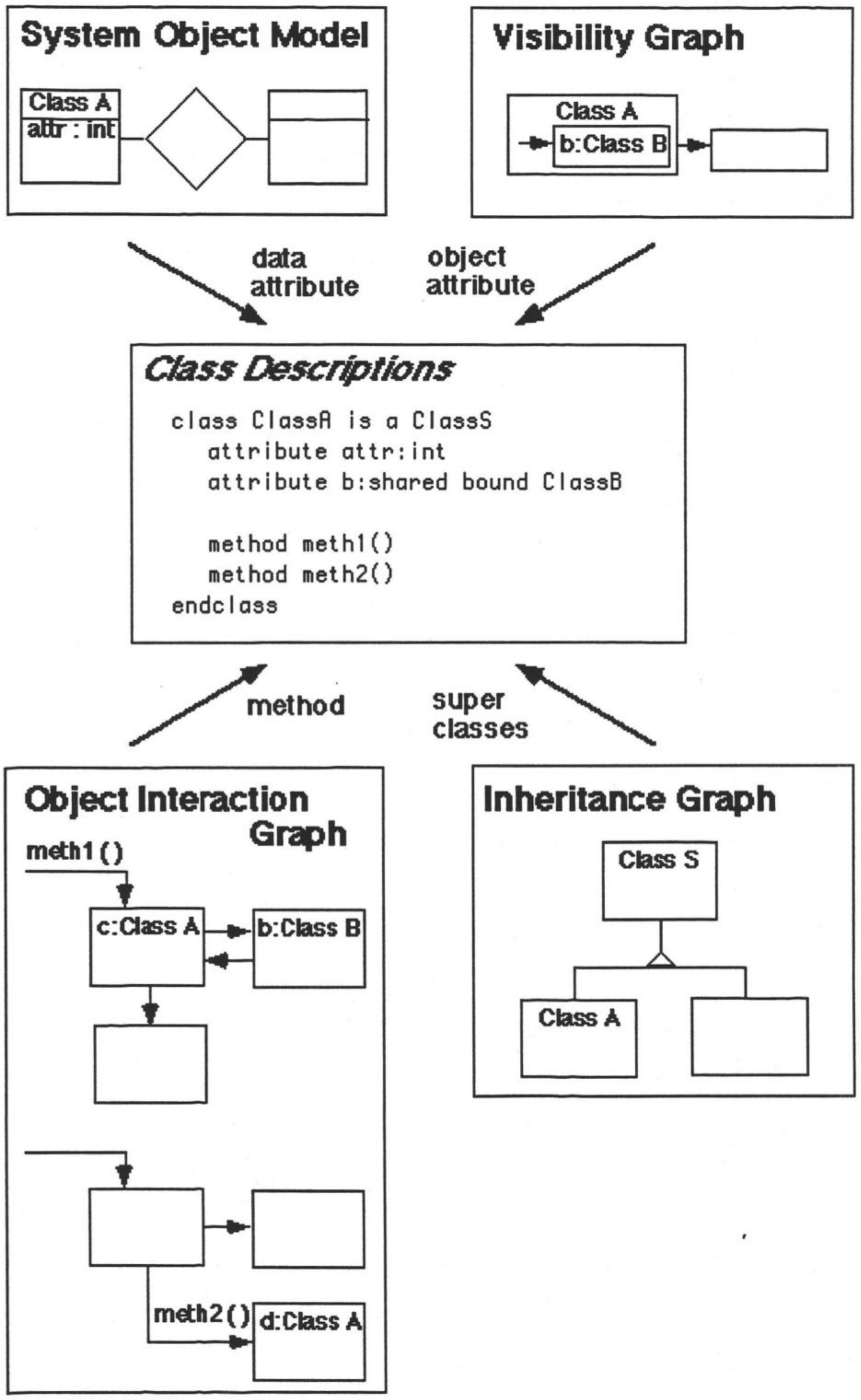


Figure A-8 The Process of Developing Class Description

Each method is introduced by the keyword **method** followed by the name of the method and its signature. Methods are derived from object interaction graphs. All object interaction graphs in which the objects of a class participate should be considered together. Messages sent to the objects (i.e., the incoming arrows) are collected together to form the interface of the class.

```
class <ClassName> [isa <SuperClassNames>]
        // for each attribute
        ...
[attribute] [Mutability]<a_name>:[Sharing][Binding]<Type>
        // for each method
        ...
[method] <m_name><arglist>[:<type>]
endclass
```

Figure A-9 Class Description Syntax

A-7 Inheritance Graphs

When the initial class descriptions have been produced, the inheritance structures required in a system are designed to position the classes in the class/subclass inheritance structures.

10. APPENDIX B

B-1 Load Flow Equations

A load flow problem [42,43] can be modelled as a set of nonlinear algebraic equations, the solutions are based on linear or iterative techniques. The solutions must satisfy Kirchhoff's law, i.e., the algebraic sum of all flows at a bus must equal zero, and the algebraic sum of all voltages in a loop must equal to zero. Equations (6.1) and (6.2) provide basic equations for load flow calculation.

$$\overline{S} = \overline{V}\overset{*}{\overline{I}} \tag{6.1}$$

$$\overline{I} = Y\overline{V} \tag{6.2}$$

Let

$$\overline{S} = P + jQ$$

$$Y = G + jB$$

$$\overline{V_i} = |V_i|e^{j\theta_i}$$

Equation (6.1) can be written as:

$$\begin{aligned} P_i + jQ_i &= \overline{V_i} * \overset{*}{\overline{I_i}} \\ &= |V_i|e^{j\theta_i}\sum_{j\in i}(G_{ij} - jB_{ij})V_j|e^{-j\theta_j} \qquad (i=1,2,....,n) \end{aligned} \tag{6.3}$$

Equation (6.3) can be extended for each node *i=1,2,....,n:*

$$\left.\begin{aligned} P_i + jQ_i &= |V_i|\sum_{j\in i}|V_j|(G_{ij} - jB_{ij})(\cos\theta_{ij} + j\sin\theta_{ij}) \\ P_i &= |V_i|\sum_{j\in i}|V_j|(G_{ij}\cos\theta_{ij} + B_{ij}\sin\theta_{ij}) \\ Q_i &= |V_i|\sum_{j\in i}|V_j|(G_{ij}\sin\theta_{ij} - B_{ij}\cos\theta_{ij}) \end{aligned}\right\} \tag{6.4}$$

The difference between the specified power and the calculated power can be expressed for each bus as:

$$\left.\begin{aligned} \Delta P_i &= P_{is} - |V_i| \sum_{j \in i} |V_j| (G_{ij} \cos\theta_{ij} + B_{ij} \sin\theta_{ij}) \\ \Delta Q_i &= Q_{is} - |V_i| \sum_{j \in i} |V_j| (G_{ij} \sin\theta_{ij} - B_{ij} \cos\theta_{ij}) \end{aligned}\right\} \qquad (6.5)$$

It is noticed from Equation (6.5) that associated with each bus of the network there are four variables: the real and reactive power, the voltage magnitude and phase angle. The objective of the load flow calculation is to find a set of $V_i, \theta_i \, (i = 1,2,\ldots,n)$ to satisfy a predefined set of values $\Delta P_i, \Delta Q_i \, (i = 1,2,\ldots.,n)$ for a given set of values $P_{is}, Q_{is} \, (i = 1,2,\ldots\ldots,n)$.

There are three types of buses in a load flow calculation and at each bus two of the four variables are specified. It is necessary to select one bus, called the slack bus, to provide the additional real and reactive power to supply the transmission losses. Since the real and reactive power are unknown at the slack bus until the final solution is obtained, the voltage magnitude and phase angle are specified. The remaining buses of the system are designated either as load (PQ) buses at which the real and reactive powers are specified or as generation (PV) buses at which the real power and voltage magnitude are specified. If the total number of PV buses is r, there are (2n-r-1) equations that have to be solved by the load flow program.

Applying the Newton-Raphson method to load flow studies, a set of linear equations (Equation 6.6) can be formed expressing the relationship between the changes in real and reactive powers and in the magnitude and phase angle of bus voltages:

$$\begin{bmatrix} \Delta P_1 \\ \Delta P_2 \\ M \\ \Delta P_{n-1} \\ \Delta Q_1 \\ \Delta Q_2 \\ M \\ \Delta Q_{n-1} \end{bmatrix} = \begin{bmatrix} H_{1,1} & H_{1,2} \Lambda & H_{1,n-1} & N_{1,1} & N_{1,2} \Lambda & N_{1,n-1} \\ H_{2,1} & H_{2,2} \Lambda & H_{2,n-1} & N_{2,1} & N_{2,2} \Lambda & N_{2,n-1} \\ & & \Lambda \Lambda & & & \\ H_{n-1,1} & H_{n-1,2} \Lambda & H_{n-1,n-1} & N_{n-1,1} & N_{n-1,2} & N_{n-1,n-1} \\ J_{1,1} & J_{1,2} \Lambda & J_{1,n-1} & L_{1,1} & L_{1,2} \Lambda & L_{1,n-1} \\ J_{2,1} & J_{2,2} \Lambda & J_{2,n-1} & L_{2,1} & L_{2,2} \Lambda & L_{2,n-1} \\ & & \Lambda \Lambda & & & \\ J_{n-1,1} & J_{n-1,2} \Lambda & J_{n-1,n-1} & L_{n-1,1} & L_{n-1,2} \Lambda & L_{n-1,n-1} \end{bmatrix} \times \begin{bmatrix} \Delta\theta_1 \\ \Delta\theta_2 \\ M \\ \Delta\theta_{n-1} \\ \Delta V_1 / |V_1| \\ \Delta V_2 / |V_2| \\ M \\ \Delta V_{n-1} / |V_{n-1}| \end{bmatrix} \qquad (6.6)$$

Equation (6.6) is called the correction equation. Each element of Equation (6.6) can be calculated in the following manner.

$$H_{ij} = \frac{\partial \Delta P_i}{\partial \theta_j} = -|V_i||V_j|(G_{ij}\sin\theta_{ij} - B_{ij}\cos\theta_{ij}) \quad j \neq i \tag{6.7}$$

$$\left.\begin{aligned} H_{ii} &= \frac{\partial \Delta P_i}{\partial \theta_i} = |V_i|\sum_{\substack{j\in i\\ j\neq i}} |V_j|(G_{ij}\sin\theta_{ij} - B_{ij}\cos\theta_{ij}) \\ H_{ii} &= |V_i|^2 B_{ii} + Q_i \end{aligned}\right\} \tag{6.8}$$

$$N_{ij} = \frac{\partial \Delta P_i}{\partial V_j}|V_j| = -|V_i||V_j|(G_{ij}\cos\theta_{ij} + B_{ij}\sin\theta_{ij}) \quad j \neq i \tag{6.9}$$

$$\left.\begin{aligned} N_{ii} &= \frac{\partial \Delta P_i}{\partial V_i}|V_i| = -|V_i|\sum_{\substack{j\in i\\ j\neq i}} |V_j|(G_{ij}\cos\theta_{ij} + B_{ij}\sin\theta_{ij}) - 2|V_i|^2 G_{ii} \\ N_{ii} &= -|V_i|^2 G_{ii} - P_i \end{aligned}\right\} \tag{6.10}$$

$$J_{ij} = \frac{\partial \Delta Q_i}{\partial \theta_j} = |V_i||V_j|(G_{ij}\cos\theta_{ij} + B_{ij}\sin\theta_{ij}) \quad j \neq i \tag{6.11}$$

$$\left.\begin{aligned} J_{ii} &= \frac{\partial \Delta Q_i}{\partial \theta_i} = -|V_i|\sum_{\substack{j\in i\\ j\neq i}} |V_j|(G_{ij}\cos\theta_{ij} + B_{ij}\sin\theta_{ij}) \\ J_{ii} &= |V_i|^2 G_{ii} - P_i \end{aligned}\right\} \tag{6.12}$$

$$L_{ij} = \frac{\partial \Delta Q_i}{\partial V_j}|V_j| = -|V_i||V_j|(G_{ij}\sin\theta_{ij} - B_{ij}\cos\theta_{ij}) \quad j \neq i \tag{6.13}$$

$$\left.\begin{aligned} L_{ii} &= \frac{\partial \Delta Q_i}{\partial V_i}|V_i| = -|V_i|\sum_{\substack{j\in i\\ j\neq i}} |V_j|(G_{ij}\sin\theta_{ij} - B_{ij}\cos\theta_{ij}) + 2|V_i|^2 B_{ii} \\ L_{ii} &= |V_i|^2 B_{ii} - Q_i \end{aligned}\right\} \tag{6.14}$$

Equation (6.6) can be written as:

$$\begin{bmatrix} \Delta \mathrm{P} \\ \Delta \mathrm{Q} \end{bmatrix} = \begin{bmatrix} \mathrm{H} & \mathrm{N} \\ \mathrm{J} & \mathrm{L} \end{bmatrix} \begin{bmatrix} \Delta\theta \\ \Delta V/|V| \end{bmatrix} \tag{6.15}$$

Where the coefficient matrix is called a Jacobian matrix.

Equation (6.15) can be expanded as:

$$\left.\begin{aligned}\Delta P &= H\Delta\theta + N\,\Delta V/|V| \\ \Delta Q &= J\Delta\theta + L\,\Delta V/|V|\end{aligned}\right\} \tag{6.16}$$

B-2 Decoupled Newton-Raphson Method

The close relationship between the active power and bus voltage phase angles is evident[44] in systems with modest loads and generally low R/X ratios in the transmission system. Similarly the reactive power are highly dependent on bus voltage magnitude. These observations are reinforced by examining the $\partial P/\partial\theta$ and $\partial Q/\partial V$ submatrices of Jacobian matrix. These submatrices are dominant in the Jacobian matrix. This dominance suggests the approximation in which the $\partial P/\partial V$ and $\partial Q/\partial\theta$ submatrices are assumed small enough to be ignored.

Thus, ΔP and $\Delta\theta$ variables can be decoupled from ΔQ and ΔV variables:

$$\left.\begin{aligned}\Delta P &= H\Delta\theta \\ \Delta Q &= L\,\Delta V/|V|\end{aligned}\right\} \tag{6.17}$$

In this way, a set of 2n order linear equations has been decoupled into two sets of n order linear equations. H and L coefficient matrices are changed during each iteration and these matrices are not symmetric. The second simplification is to consider that the phase angle between two ends of the line is small (usually not larger than 10°~20°) and, therefore, the following equations can be assumed as true:

$$\left.\begin{aligned}\cos\theta_{ij} &\approx 1 \\ G_{ij}\sin\theta_{ij} &<< B_{ij} \\ Q_i &<< |V_i|^2 B_{ii}\end{aligned}\right\} \tag{6.18}$$

The elements of **H**, **L** can be deduced as:

$$\left.\begin{aligned}H_{ii} &= |V_i|^2 B_{ii} \\ H_{ij} &= |V_i||V_j|B_{ij}\end{aligned}\right\} \tag{6.19}$$

$$\left.\begin{aligned}L_{ii} &= |V_i|^2 B_{ii} \\ L_{ij} &= |V_i||V_j|B_{ij}\end{aligned}\right\} \tag{6.20}$$

Therefore, **H** and **L** can be represented as:

$$H = L = \begin{bmatrix} |V_1| & & & \\ & |V_2| & & \\ & & \mathrm{O} & \\ & & & |V_n| \end{bmatrix} \begin{bmatrix} B_{11} & B_{12} & \Lambda & B_{1n} \\ B_{21} & B_{22} & \Lambda & B_{2n} \\ & \Lambda & & \\ B_{n1} & B_{n2} & \Lambda & B_{nn} \end{bmatrix} \begin{bmatrix} |V_1| & & & \\ & |V_2| & & \\ & & \mathrm{O} & \\ & & & |V_n| \end{bmatrix} \tag{6.21}$$

The correction equations for Δ **P**, Δ **Q** can be rewritten as:

$$\begin{bmatrix} \Delta P_1 \\ \Delta P_2 \\ \mathrm{M} \\ \Delta P_n \end{bmatrix} = \begin{bmatrix} |V_1| & & & \\ & |V_2| & & \\ & & \mathrm{O} & \\ & & & |V_n| \end{bmatrix} \begin{bmatrix} B_{11} & B_{12} & \Lambda & B_{1n} \\ B_{21} & B_{22} & \Lambda & B_{2n} \\ & \Lambda & & \\ B_{n1} & B_{n2} & \Lambda & B_{nn} \end{bmatrix} \begin{bmatrix} |V_1| * \Delta\theta_1 \\ |V_2| * \Delta\theta_2 \\ \mathrm{M} \\ |V_n| * \Delta\theta_n \end{bmatrix} \tag{6.22}$$

$$\begin{bmatrix} \Delta Q_1 \\ \Delta Q_2 \\ \mathrm{M} \\ \Delta Q_n \end{bmatrix} = \begin{bmatrix} |V_1| & & & \\ & |V_2| & & \\ & & \mathrm{O} & \\ & & & |V_n| \end{bmatrix} \begin{bmatrix} B_{11} & B_{12} & \Lambda & B_{1n} \\ B_{21} & B_{22} & \Lambda & B_{2n} \\ & \Lambda & & \\ B_{n1} & B_{n2} & \Lambda & B_{nn} \end{bmatrix} \begin{bmatrix} \Delta V_1 \\ \Delta V_2 \\ \mathrm{M} \\ \Delta V_n \end{bmatrix} \tag{6.23}$$

Multiply Equations (6.22) and (6.23) with the following equation:

$$\begin{bmatrix} |V_1| & & & \\ & |V_2| & & \\ & & \mathrm{O} & \\ & & & |V_n| \end{bmatrix}^{-1} = \begin{bmatrix} 1/|V_1| & & & \\ & 1/|V_2| & & \\ & & \mathrm{O} & \\ & & & 1/|V_n| \end{bmatrix}$$

The correction equation finally becomes:

$$\begin{bmatrix} \Delta P_1/|V_1| \\ \Delta P_2/|V_2| \\ \mathrm{M} \\ \Delta P_n/|V_n| \end{bmatrix} = \begin{bmatrix} B_{11} & B_{12} & \Lambda & B_{1n} \\ B_{21} & B_{22} & \Lambda & B_{2n} \\ & \Lambda & & \\ B_{n1} & B_{n2} & \Lambda & B_{nn} \end{bmatrix} \begin{bmatrix} |V_1| * \Delta\theta_1 \\ |V_2| * \Delta\theta_2 \\ \mathrm{M} \\ |V_n| * \Delta\theta_n \end{bmatrix} \tag{6.24}$$

$$\begin{bmatrix} \Delta Q_1/|V_1| \\ \Delta Q_2/|V_2| \\ \mathrm{M} \\ \Delta Q_n/|V_n| \end{bmatrix} = \begin{bmatrix} B_{11} & B_{12} & \Lambda & B_{1n} \\ B_{21} & B_{22} & \Lambda & B_{2n} \\ & \Lambda & & \\ B_{n1} & B_{n2} & \Lambda & B_{nn} \end{bmatrix} \begin{bmatrix} \Delta V_1 \\ \Delta V_2 \\ \mathrm{M} \\ \Delta V_n \end{bmatrix} \tag{6.25}$$

The correction Equations (6.24) and (6.25) together with mismatch Equation (6.5) form the basic equations for the decoupled Newton Raphson method. Compared to the Newton Raphson method, the decoupled Newton Raphson method simplifies calculations in the following aspects:

1. For a system with n nodes, the correction equations are two sets of n ordered equations instead of one set of 2n ordered equations.
2. The coefficient matrix of correction equation becomes the imaginary part of the admittance.
3. The coefficient matrix in Equations 6.24 and 6.25 remains symmetric and constant during each iteration.

The process of solving above equations begins with assigning initial status of the system, which includes voltage magnitude and phase angle for all buses, the real and reactive powers for PQ buses, the real power and voltage magnitude for PV buses. Then Equation (6.5) is used to calculate the difference between the specified power and calculated power under given conditions. These differences are used to calculate the adjusted value of voltage magnitude and phase angle for each bus which will be used for the next iteration. The process is repeated until the difference between the specified and calculated real and reactive power in all buses are within a specified tolerance.

The flow diagram of the load-flow study using the DNR decoupled method is shown in Figure B-1:

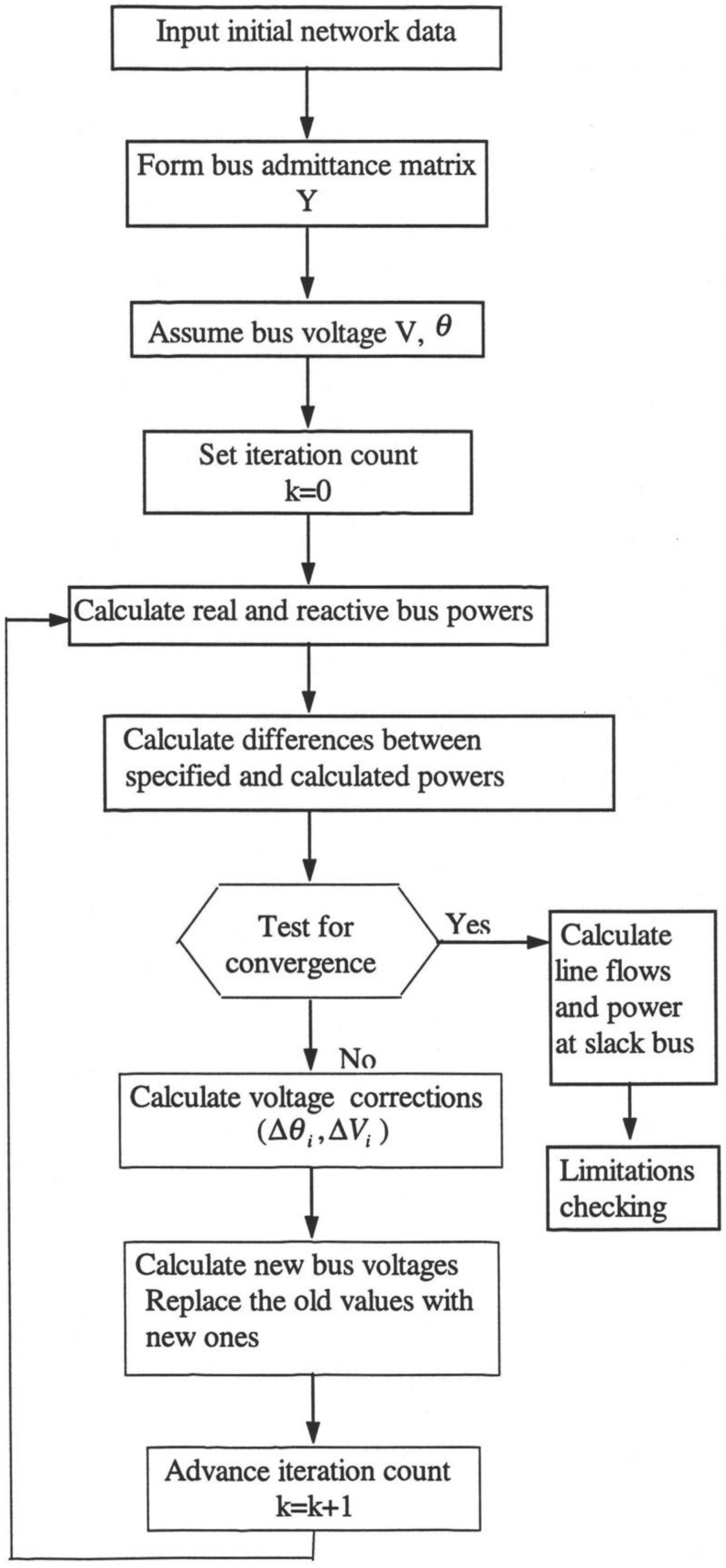

Input initial network data
Form bus admittance matrix Y
Assume bus voltage V, θ
Set iteration count k=0
Calculate real and reactive bus powers
Calculate differences between specified and calculated powers
Test for convergence
Yes
Calculate line flows and power at slack bus
Limitations checking
No
Calculate voltage corrections ($\Delta\theta_i, \Delta V_i$)
Calculate new bus voltages Replace the old values with new ones
Advance iteration count k=k+1

Figure B-1 The Flowchart of the DNP