Object-Oriented Software of Power System Analysis

Tarek Bouktir, Linda Slimani, Belkacem Mahded, Ahmed Gherbi
Department of Electrical Engineering, University of Oum El-Bouaghi, 04000 Algeria

Abstract— This paper describes an efficient software platform for power system analysis. Making use of the object-oriented methodology advantages, the power system is modeled as an aggregate of linked objects and structured in class hierarchy. Each class describes an object and encapsulates both its data structure and behavior (operating methods). Attributes of all electrical objects are defined using Power System Application Data Dictionary (PSADD). The objects interact and cooperate among themselves via ultimate message-passing paradigms. All parts of the software (numerical methods, database, GUI…) have been built in terms of objects. Borland C++Builder 5 was used to develop this compact, flexible and easy-to-maintain program. This software may be used in educational purpose or in modeling, planning and analyzing of the power system.

Key-words : Power System, Simulation, Object-Oriented Programming, PSADD.

1 Introduction

The planning, design and operation of electrical power system require simulation analyses to determine the performance and the reliability of the current and future system. The computer simulation requires that a power system should be represented by a set of data and functions structured in certain processing algorithms to produce simulation results. During the last three decades, many power system computer programs have been described in the open literature [1-3]. Some of these softwares were designed to provide various engineering analysis ranging from load flow to transient stability. Others were developed for planning and controlling power system in real-time [4,5]. However, these programs have proved their inadequacy and their limitation to specified power configuration. For different applications, the data structures are different depending on the programming languages. With the conventional programming languages (Fortran, C, Pascal), the data structures and the algorithm procedures are very strong. So that any change even minor may propagate through a whole developed modules of the program. This requires tremendous efforts to debug and time period is proportional to the size of the program source rather than the magnitude of the change. For a large-scale software system, this could conduct to a catastrophic consequence so that the system becomes unmanageable and requires to be redesigned. The limitations of the software in turn restrict the potential use of modern electronic equipment to help manage and protect the grid, because the software cannot manage the increased data such equipment can deliver. The New York Times of August 18, 2003 wrote (after the August 14, 2003 Northeast Blackout), regarding transmission lines in Ohio and the Midwest, "Problems on the lines were becoming more frequent, and a series of reports, by the industry's own quality-control offices and government agencies, described the risks and urged utilities to be more aware of the physical limits of the Midwest system. The complexity and magnitude of the power flows, one report by the Federal Energy Regulatory Commission said, could 'overwhelm the electronic and software tools used to model and manage power flows on the grid'". We think that is the same conclusion for the Feb,03,2003 Algerian blackout.

To overcome these disadvantages of traditional softwares, one promising approach to achieve software reliability, maintainability and extensibility is to use Object-Oriented Methodology (OOM) [6].

Object-Oriented Technology (OOT) is the newest methodology of software development, which has proved to be an effective tool for complex systems modeling. It allows expansions and modifications over a long period and supports a wide range of applications. With OOP, problems are modeled based on physical real-world concepts. Its greatest benefits come from helping developers express abstract concepts (objects) clearly and communicate them to each other. Object-oriented approaches use the concept of objects as the unit of the organization. Objects are simple entities which are specified *what* they can do rather *how* they are done. So, they can be viewed as 'black-boxes'.

During the few last years, the application of the OOP to electric power system has gained widespread acceptance. The object-oriented modeling scheme is not unique and affects directly the performance of the developed software. Many authors have developed various packages and proposed interesting object-oriented power system model using different OOP languages like C++, Smalltalk, Eiffel [7,8]. However, most of the developed packages are limited to specific subjects among the various fields of power systems engineering.

In a previous article [9], the authors have proposed an object model of power system which has been tested successfully for load flow application on Sonelgaz network. This paper describes an other version of software platform for power system modeling using OOM. The system has been modeled as set of linked objects (entities). These may be real objects (e.g. electrical devices) or software objects (e.g. database, numerical methods). Then, an object Optimal Power Flow application is described and integrated as a new module in our developed object-oriented software called Object Oriented Electrical Network Simulator (OOENS). This one has been implemented using Borland C++Builder 5 and run under

Windows XP environment on desk PC Pentium 4 (1.5 GHz, 128 Mo of main memory).

2 Overview Of The Object-Oriented Modeling

The central theme in OO modeling is the process of discovering classes, object, operations as well as the relationships among classes. In this section, we shall give a brief overview of these concepts.

2. 1. CLASSES AND OBJECTS

Somewhat predictably, the idea of an object is central to object-oriented programming. An object is defined as an item of data, very much like a variable (or constant) in a conventional programming language. Every object belongs to an object class, which is analogous to a data type in a conventional language. However, one of the most important things about object classes is that new classes can be defined by the programmer, based on existing classes. Every object is an instance of a class. A class defines the methods and attributes that each instance of the class will possess (intentional view). It can also be seen as defining the set of objects which are instances of the class (extensional view). Using the Rumbaugh's notations [11], a class in object models is represented as a rectangle with three parts as shown in Figure 1; at the Top is indicated the name of the class, the middle part contains all data attributes of the class and at the lower part the class functions (methods) are defined. The c*onstructor* "Line (){}" is a function which initializes all the class variables. The d*estructor* "~Line (){}" is a special function for freeing the allocated memory.

Line
Bus FromBus **Bus** ToBus **complex** Current ……………..

complex *Impedance*();
complex *PowerLoss*();
Line (){};
~Line (){};

Figure 1 Class representation

2. 2. PRINCIPLES OF OOM

Four major principles underlie object-oriented modeling [6,11,12]. These are:

i. **Abstraction** which denotes the capability to capture the essential properties of an entity without undue details. It consists of filtering of details not immediately needed.

ii. **Encapsulation** this defines the hiding of implementation. It requires the packaging of the entity into one impermeable unit.

iii. **Polymorphism** means that an entity can assume many forms.

iv. **Inheritance (generalization)** which allows new object classes to be defined in terms of existing object classes, inheriting both data structure and behavior (definition of methods) from the defining parent class or superclass. A class which inherits from another class is said to be a subclass of its parent. It is possible to define a method in a subclass with the same name as a method in its parent class, and this new method will **override** the method from the parent class. Complete class hierarchies can be built up through inheritance, perhaps representing hierarchies in the real-world.

2. 3. OBJECT-ORIENTED RELATIONSHIPS

The relationships are mathematical relations among classes and objects. They capture associations that may exist among objects. The following relationships are common:

i. **Classification**: it is used to describe the inheritance association between superclass and subclasses.

ii. **Aggregation**: it used to describe an aggregate class, which is formed from a number of component classes.

iii. **Instantiation**: this refers to the case where a class is instantiated to create a specific object.

iv. **Association** : it describes the physical or conceptual connection between classes.

2. 4. STAGES OF OBJECT MODELING

In Object Modeling **Technique** (OMT), building a model of an application domain is achieved following three major stages [11,13]:

1. Object-Oriented **Analysis** (OOA): during this stage, the real-world system is modeled as an aggregate of simple linked objects. Their relationships are identifying using the application-domain concepts.
2. Object-Oriented **Design** (OOD): During this phase, decisions are made about how the problem will be solved. This includes two design stages:
   i. System design: in this phase, the overall system architecture is decided.
   ii. Object design: which involves the building of the model by defining the classes and their associations used in the implementation and algorithms of the methods used to implement operations.
3. **Implementation and testing**: during this phase, the developed objects and classes are finally translated in a particular OOP language (e.g. C++Builder) to become complete software.

Object-oriented modeling uses the same conceptual model across analysis, design and implementation. Analysis and design stages are intertwined in some OO methodologies.

3 Anatomy Of The Developed Software

The main idea presented in this project software is based upon the object concept. So, all parts of the software are designed separately as objects using OOT in order to facilitate modifications or enhancements.

It consists of five major objects:

- A user-friendly Graphical User Interface (GUI)
- A central Object-Oriented Database (OODB)
- A number of power system applications
- An Object Mathematical Library.
- A network container.

The electrical network is modeled using two main object concepts: the classification which groups similar objects into

classes and the specialization which refines classes into subclasses.
According to this approach, an object-oriented system is viewed as a collection of classes and instances ordered by two relations: instantiation and inheritance. The inheritance is a powerful abstraction for sharing simulates among classes while preserving their differences. It is the relationship between a class and one or more refined versions of it. The facilities of the base class are automatically available to all subclasses. The electrical network model is structured in the same way as the physical network. In the present case, the plant components that make up the power system for the static analysis consist of elements such as generators, transformers, transmission lines, loads and capacitor banks.

Figure 2 illustrates the implementation of the class *Device* that has been defined as the base class for all the electrical network classes. The class *Device* has a private part consisting of parameters that are normally used for describing the electrical behavior of the component and a public part consisting of data accessing methods used to coordinate the class with the other classes and to communicate with the external environment of the class.

```
class Device   // class name
{
private: // Private part of the class
AnsiString FNames; // device name
int FNum;          // device number
AnsiString GetName();  // getting the
name
void SetName(AnsiString); // setting
name
int GetNum();// getting number
void SetNum(const int); // setting
number
public: // Public part
__property AnsiString Names=
{read=GetName, write=SetName};
__property int Num = {read=GetNum,
write=SetNum};
Device(){} // Constructor
~Device(){} // Destructor } // {Device}
```

Figure 2. The base class Device of the electrical network

Figure 3 shows the global design of the base classes and their subclasses. Four base classes are identified and derived from the base class *Device*.
The class *Bus* represents a main base class because all the electrical components are connected to one or two buses. *DeviceWithTwoConnections* represents the electrical components which are linked to two buses whereas those linked to a single bus are grouped in *DeviceWithOneConnection class*. The fourth class is grouping all protection devices. Using inheritance, new classes representing the remaining network elements are derived from these classes. For example, the class *DeviceWithTwoConnections* is the base class of the network lines and transformers. Attributes and operations attached to the base class *DeviceWithTwoConnections* such as the sending bus and the receiving bus numbers, and also the resistance, reactance and susceptance are inherited by the line and transformer subclasses.

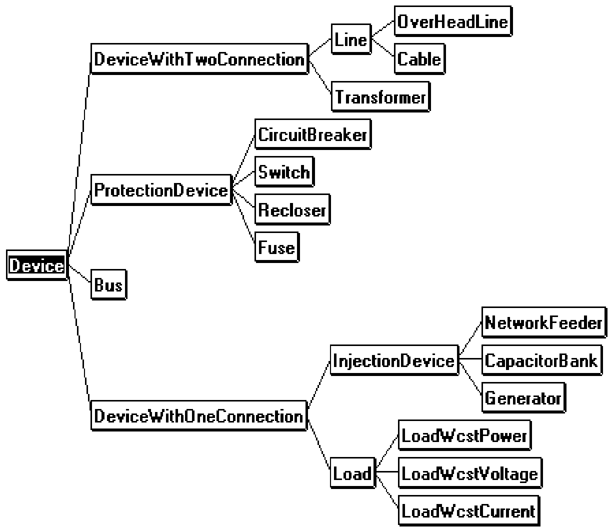


Figure 3. Class diagram of the electrical network object model

4 Network Container Class

The network container is a dynamic vector instance inclosing the different components of the network model. The topology of the electrical system is defined by the admittance matrix which is declared in the public part. This can be used by other applications having an instance of the network container. Figure 4 illustrates the network container class with its different parameters and necessary methods providing access to manage the component objects.

```
class Network: public Device{
private:
vector<Bus> Bus;
vector<Generator> Generator;
vector<Line> Line;
vector<Load> Load;
vector<Transformer> Transformer;
void Admittance();
public:
__property nit NB =
{read=GetTotalBusesNumber,
write=SetTotalBusesNumber};
__property int NL =
{read=GetTotalBranchesNumber,
write=SetTotalBranchesNumber};
__property int NSLACK  ={read=GetBusBar,
write=SetBusBar};
__property double BASE ={read=GetBasePU,
write=SetBasePU};
.................................
Network(){};// Constructor
~Network(){};// Destructor};
```

Figure 4. Network container class with its attributes and Operations

Figure 5 shows the IEEE 30-bus system that is drawn by our software.

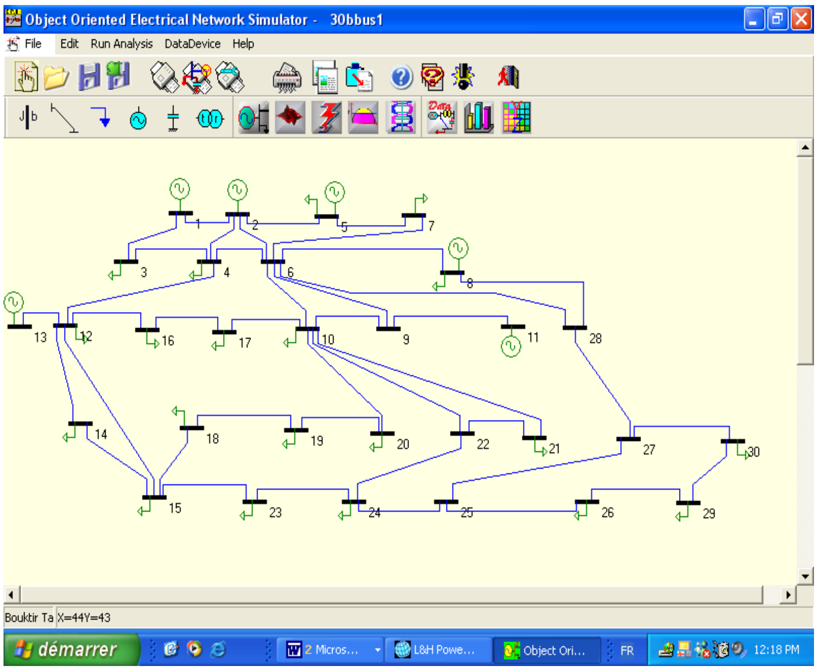


Figure 5. The main form of the simulator

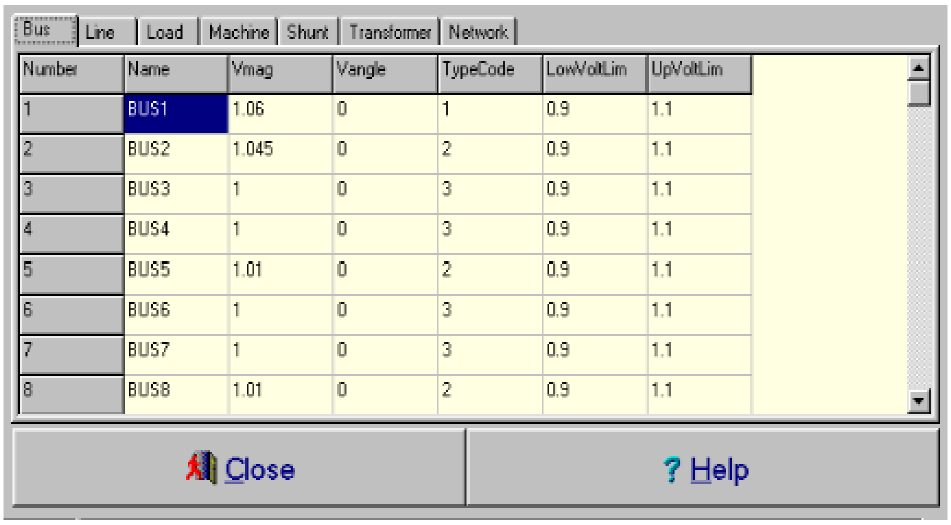
5 Database Module and Data Dictionary

5.1. Database Module

The data associated with any electrical device can be stored using windows-based database system, and can be invoked by any analysis application. The advantages of database structure are: Flexibility of data transfer between the environment and other applications and data source. Data can be imported and exported easily via the SQL (structured Query Language) Security of data from loss or corruption is correctly ensured. Extension to the database can be made without difficulties.

5.2. The Data Dictionary

Attributes of all electrical objects are defined using Power System Application Data Dictionary (PSADD).



Number	Name	Vmag	Vangle	TypeCode	LowVoltLim	UpVoltLim
1	BUS1	1.06	0	1	0.9	1.1
2	BUS2	1.045	0	2	0.9	1.1
3	BUS3	1	0	3	0.9	1.1
4	BUS4	1	0	3	0.9	1.1
5	BUS5	1.01	0	2	0.9	1.1
6	BUS6	1	0	3	0.9	1.1
7	BUS7	1	0	3	0.9	1.1
8	BUS8	1.01	0	2	0.9	1.1

Figure 6 attributes of electrical devices stored using windows-based database system

Its can be stored using windows-based database system in order to build a flexible general-purpose power flow environment that have great expendability and flexibility (Figure 6).

The PSADD is an attempt by IEEE to extend and renovate the old standby of the power industry, the IEEE Common Format, which served the industry well for many years. The Data Dictionary is meant primarily as a definition of terms used for analytical applications within power system models. It is organized around the notion of objects. In this paper, we us this Data dictionary to define attributes of all electrical objects. The PSADD integrates two viewpoints: a bus-oriented viewpoint common to analytical studies, and an equipment-oriented viewpoint common to measured values and system operation. The communication between the graphics, the database, network classes and power system applications are presented in the figure 7.

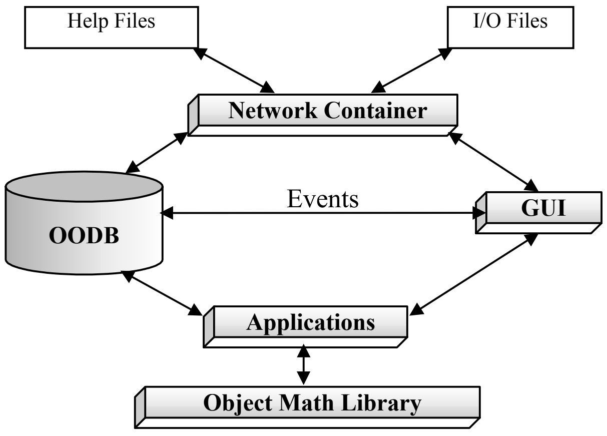


Figure 7. Architecture of the OOP environment.

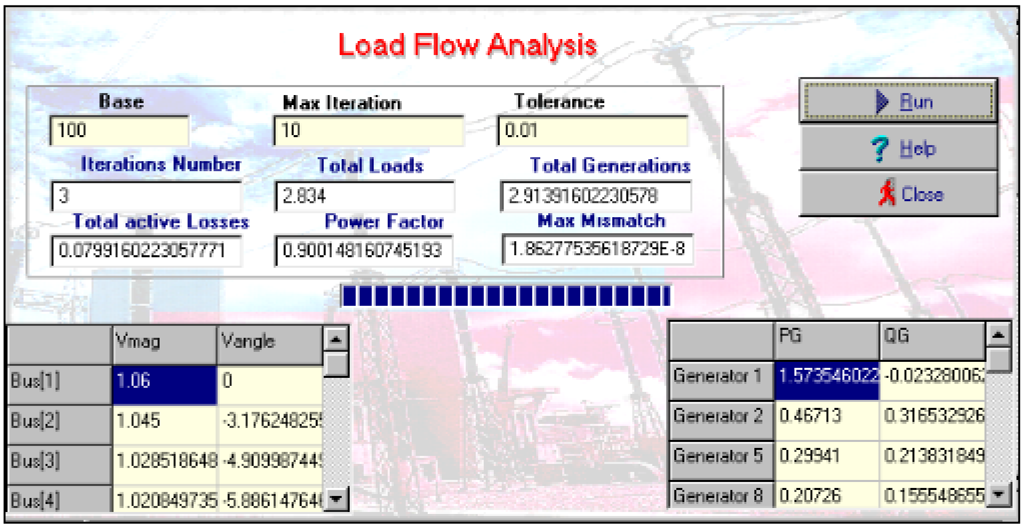
6 Object Math Library Module

The power system analysis requires powerful numerical methods to obtain accurate results. Using OOP concepts and operators overloading inline functions, a flexible mathematical library which includes the commonly used numerical methods has been implemented. The main idea is to include numerical algorithms and methods as conceptual objects. The heart of this module is arrays objects (vector and matrix) which are called in solving iterative and non-iterative equations. These arrays have been implemented as templates objects in order to support all types of data declaration (integer, complex and float). The memory is dynamically managed using the ***new*** and ***delete*** C++ keywords.

7 Power System Applications

7.1 Load flow calculation

Performing a load flow solution in a distribution network is required after any change in loads. This will provide updated voltages, angles and transformer taps and points out generators having exceeded reactive limits. to determining all active and reactive power of all generators and to determine power that it should be given by the slack generator after any change in load. The load flow is also necessary to perform other studies such as fault analysis, transient stability, and optimal power flow. All these require a fast and robust load flow program with best convergence properties. The developed load flow module is based upon the full Newton-Raphson algorithm.
By clicking the appropriate push button on the main window, an interactive and graphic interface for the Load Flow Application is invoked in the screen (figure 8).



Base	Max Iteration	Tolerance
100	10	0.01

Iterations Number	Total Loads	Total Generations
3	2.834	2.91391602230578

Total active Losses	Power Factor	Max Mismatch
0.0799160223057771	0.900148160745193	1.86277535618729E-8

	Vmag	Vangle
Bus[1]	1.06	0
Bus[2]	1.045	-3.17624825
Bus[3]	1.028518648	-4.90998744
Bus[4]	1.020849735	-5.88614764

	PG	QG
Generator 1	1.57354602	-0.0232800
Generator 2	0.46713	0.316532926
Generator 5	0.29941	0.213831849
Generator 8	0.20726	0.155548655

Figure 8. Power Flow Window

To perform the PF calculation, the user clicks the run button then the load flow program is executed, and the results are displayed on the load flow window.

7.2 Object Oriented Optimisation library

Using Object Oriented Programming, the optimisation methods used to compute the optimal power flow are designed in a hiearchical structure. In this structure, low-level objects are relatively abstract or general, while higher-level ones are more problem-specific. We will devide the methods into three classes:

- Local optimisation methods class
- Global optimisation methods class
- Newton type methods class.

In the local optimisation methods, the current iterates in the optimisation process are derived from the update of previous iterates that lie in a nearby neighbourhood. For example the nonlinear conjugate gradient method, the Nelder-Mead simplex method and the Quasi-Newton methods fall into the local optimisation methods class.

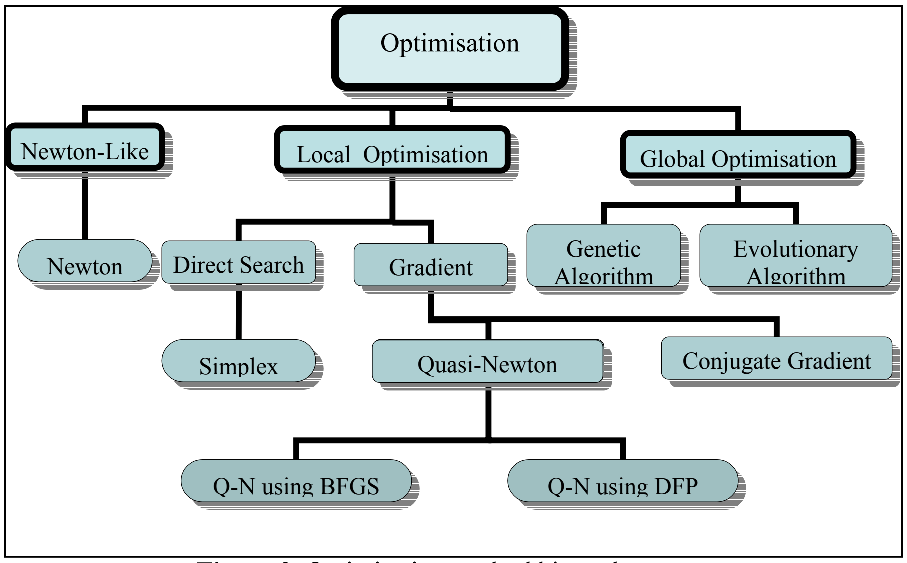


Figure 9. Optimisation method hierarchy

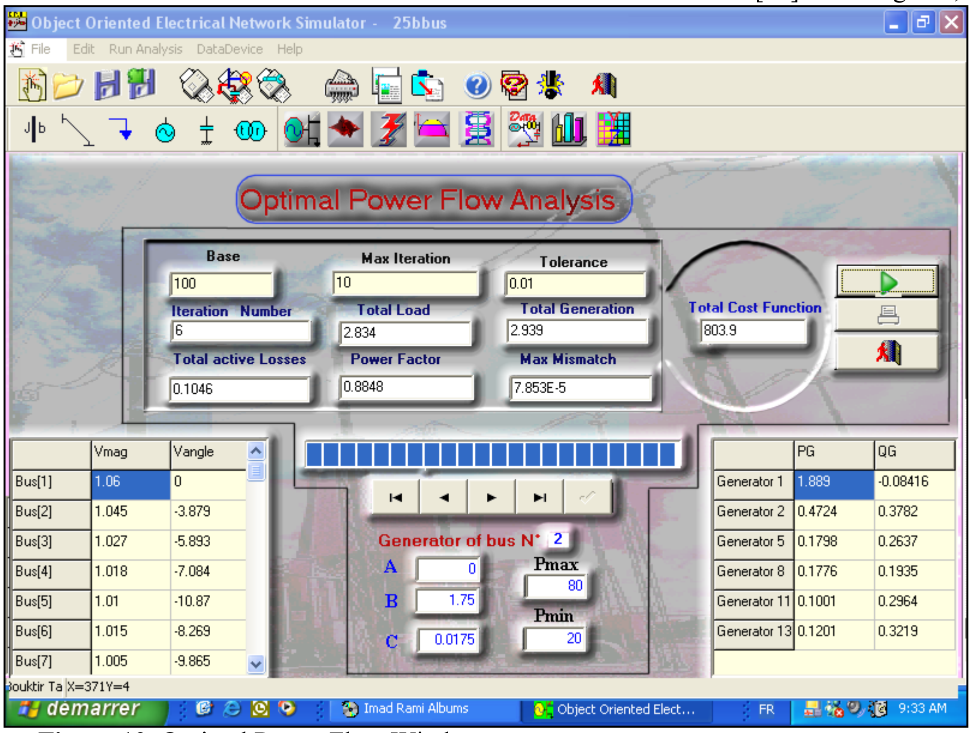
If the objective function has a high degree of complexity, then optimisation might require the use of stochastic algorithms. The genetic algorithm and evolutionary algorithm fall into this global optimisation methods class.
Due to their evolutionary nature, genetic algorithms will search for solutions without regard for the specific inner structure of the problem. GAs can handle any kind of objective functions and any kind of constraints, linear or nonlinear, defined on discrete, continuous, or mixed search spaces. In The Newton type methods, we need the availability of the objective function and analytic first and the second derivatives. In Figure 9, we present the optimisation method hierarchy.

7.3 Optimal Power Flow Class

The Optimal Power Flow (OPF) problem is modelled as a class multi-inherited from a Load Flow (LF) class and a Genetic Algorithm (GA) class and has a direct access to a main power system object class. The necessary data for performing the optimization are the upper and lower constraint vectors and the coefficient vector of the cost function. Several computing methods are available in this class such as the cost function method, the optimal line search of the minimum, the integration of the inequality constraints and the method for updating the design vectors (Figure 10).

1. Conclusion

The complexity of the electric power system has been resulted in the need for a simulation and visualization tool for power system modeling, control and operation. This paper presents an efficient object-oriented software platform for power system analysis. It has been shown that the object-oriented technology can be successfully applied in the development of large-scale software system. This software satisfies the requirements of flexibility, extensibility, maintainability and data integrity. The developed power system model may be used to develop a wide range of power applications (load flow, OPF, transient stability…). A very flexible and reusable module for numerical methods for power system analysis has been implemented. The software manages its memory requirement dynamically by itself using the features and benefits of C++Builder 5.



Base	Max Iteration	Tolerance
100	10	0.01

Iteration Number	Total Load	Total Generation	Total Cost Function
6	2.834	2.939	803.9

Total active Losses	Power Factor	Max Mismatch
0.1046	0.8848	7.853E-5

	Vmag	Vangle
Bus[1]	1.06	0
Bus[2]	1.045	-3.879
Bus[3]	1.027	-5.893
Bus[4]	1.018	-7.084
Bus[5]	1.01	-10.87
Bus[6]	1.015	-8.269
Bus[7]	1.005	-9.865

Generator of bus N° 2

A	0	Pmax	80
B	1.75	Pmin	20
C	0.0175		

	PG	QG
Generator 1	1.889	-0.08416
Generator 2	0.4724	0.3782
Generator 5	0.1798	0.2637
Generator 8	0.1776	0.1935
Generator 11	0.1001	0.2964
Generator 13	0.1201	0.3219

Bouktir Ta X=371Y=4

Figure 10. Optimal Power Flow Window

References

[1] Adielson T.,"SIMPOW- A Digital Program System For Static And Dynamic Simulation Of Power Systems" OEPSI conference, Bankok, Nov. 1982.

[2]Neyer A.F., Wu F.F., Imhof K., "Object-Oriented Programming For Flexible Software; Example Of A Load Flow", IEEE Trans. on Power Systems, Vol. 5, No.3, Aug. 1990.

[3]Rochefort M. , De Guise N., Gingras L. "Development Of A Graphical User Interface For A Real-Time Power System Simulator", Electric Power Systems Research 36, 1996, pp.203-210.

[4]Foley M., Bose A., "Object-Orientated On-Line Network Analysis", IEEE, Trans. on Power Systems, Vol. 10, pp.125-132, 1995.

[5]Hasan K., Ramsay B., Moyes I."Object Oriented Expert System For Real-Time Power System Alarm Processing-Part 1. Selection Of A Toolkit" Electric Power System Research 30, 1994, pp. 77-82

[6] Belkhouche B., "Object-Oriented Modeling Tools", http://www.eecs.tulane.edu/Belkhouche.

[7]Zhou E.Z., "Object-Oriented Programming, C++ And Power System Simulation", IEEE Trans. on Power Systems, Vol. 11, No. 1, pp. 206-214, Feb. 1996.

[8]Eldridge G. "Introducing Eiffel : An Object-Technology Approach to Power System Analysis" http://www.progsoc.uts.edu.au/~geldridg/psa-objects/ ppr_9601.zip

[9] Bouktir T., Gherbi A., Belfarhi L., Belkacemi M. "An Efficient Object-Oriented Load Flow Applied To A Large-Scale Power System : Application To Sonelgaz Network", ICEL2000, Oran, Algeria.

[10] Rumbaugh J., Baha M., Premerlani W., Eddy F., Lorensen W., "Object-Oriented Modeling And Design", Prentice Hall, Englewood Cliffs, New Jersey, USA, 1991.

[11] Reisdorph K., "Borland C++Builder 3", Simon & Schuster Macmillan, France, 1998.

[12] Booch G., "Object-Oriented Design With Applications", 2nd Ed. Benjamin Cummings, Coleman Arnold, 1994.

[13] Zhu J., Lubkeman D. L., "Object-Oriented Development Of Software Systems", IEEE Trans. on Power systems, Vol. 12, No. 2, pp. 1002-1007, May 1997.

[14] Stagg G.W., El-Abiad A.H.,"Computer Methods In Power System Analysis", McGraw Hill, 1968.