J. Indian Inst. Sci., Sept.–Oct. 2004, **84**, 141–154

Object-oriented power system analysis

M. P. SELVAN AND K. S. SWARUP
Department of Electrical Engineering, Indian Institute of Technology Madras, Chennai 600 036, India.
email: swarup@ee.iitm.ac.in; Phone: +91-44-22578404; Fax: +91-44-22570509.

Received on April 13, 2004; Revised on August 24, 2004.

Abstract

The software design of power system computation often requires modifications due to modernization of existing network and automation of the power system operation. Object-oriented design has gained widespread importance and acceptance in software development due to its advantages over other methodologies. An object-oriented design for power system analysis is proposed in this paper with greater concentration on power system equivalents and distribution system. The important contribution of this paper is the development of software objects for various transmission and distribution system components, which can be reused in most of the power system analysis programs.

Keywords: Power system computation, object-oriented design, REI equivalents, distribution system.

1. Introduction

Steady-state analysis and time-domain simulations of physical systems are carried out by digital computers to understand their behaviour completely. Behaviour of a system is represented mathematically by linear and nonlinear, algebraic and differential equations. Figure 1 shows the steps involved in the development of analysis and simulation software. Software modeling plays a significant role in representing the mathematical model of the system into the computer memory. In traditional procedural programming the behaviour of the system is completely decoupled from the characteristics or attributes of the system, which reduces the one-to-one matching between the physical system and the software model. So, the software model turns out to be very difficult to realize when the physical system becomes more complex. This is where the object-oriented modeling can provide better solution. Software objects in object-oriented modeling represent the physical components, which build the physical system. Object is an entity, which encapsulates both the characteristics and behaviour of the component. So, the physical system can be considered as a composition of such objects. These objects can interact with each other by message passing and thus can perform the task of a physical system. This paper deals with the object-oriented methodology and its applications to power system analysis.

2. Object-oriented methodology

Object-oriented technology is a new methodology, which is being extensively applied in the area of software development. This is a supreme technique to solve complex problems by tear-

*Author for correspondence.

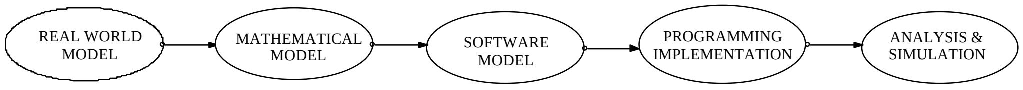

FIG. 1. Software development procedure.

ing the problem into sub-tasks. This methodology is well suited for developing not only the business application software but also engineering application software. In the object-oriented paradigm, the world is viewed as a collection of objects interacting with each other to achieve a meaningful behaviour. The basic principle of object-oriented technique is to represent each component by an element called 'object' and the relationship between objects is represented by links called 'messages' and 'responses'.

2.1 *Main features*

Object-oriented programming (OOP) generally leads to more flexible, modular and reusable code. Using OOP approach, programs can be written in a more general way. Software systems built on an object structure are more robust in the long run. One important concept in the object-oriented paradigm is *abstraction*, which captures the essential characteristics of the components of the application domain. Encapsulation gives strong coupling between the data and the operations performed on the data. In object-oriented approach the data and the functions are encapsulated into an entity called class so that the data is hidden from the user. Objects are instances of the class. The data or the internal structure of an object can be accessed only through the interfaces defined for that object. Another key concept in OOP is *inheritance*, which allows the classes to be organized into a hierarchy so that the subclasses of a super class inherit data and methods unidirectionally from the super class. This allows incremental modification to the existing class to create a new class. The new class may include new data and new methods or it can override the method of its super class. *Polymorphism* is the prime feature of object-oriented methodology, which allows different objects to receive same message but respond differently. *Operator overloading* and *function overloading* are two different kinds of polymorphism. These capabilities lead to an evolutionary and incremental approach for software development. In this approach, the software system can be designed with an open architecture, which allows long-term modifications and expansion [1].

2.2. *Phases of object-oriented methodology*

The strengths of object-oriented methodology are exploited well by the efforts involved during the analysis and design phases rather than from the actual implementation. This gives three different but interrelated phases to object-oriented methodology (Fig. 2) [2]. Object-oriented analysis (OOA) phase concentrates on finding objects in the problem do main. Object can be found as naturally occurring entities or items in the domain of interest. Domain analysis is an attempt to identify the objects. In the object-oriented design (OOD) phase, essential features of the objects are abstracted and relationships between the objects are well defined. The proposed design is implemented in the third phase, which is object-oriented programming (OOP). In OO terminology, a distinction will be made between two categories of classes. A *primitive class* is a basic class reusable in a variety of applications; a class intended to be used in a particular application is *application class*.

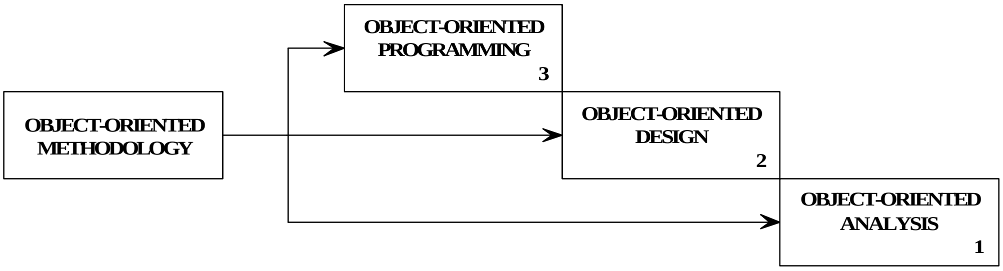
OBJECT-ORIENTED METHODOLOGY
OBJECT-ORIENTED PROGRAMMING 3
OBJECT-ORIENTED DESIGN 2
OBJECT-ORIENTED ANALYSIS 1

FIG. 2. Three phases of object-oriented methodology.

2.3. *Application to power system analysis*

Application of object-orientation towards power-system analysis is a fertile area of research since 1990. Power-system components can be grouped into different classes according to their tasks and the relationships between them can be realized through inheritance. The amount of commonality existing in the power system components, which can be exploited using inheritance and virtual function, ensures the applicability of object-oriented programming in the area of power systems. Some structure-oriented programming also supports these techniques of programming but it takes exceptional effort or exceptional skill to write such programs. Application of OOP in power system has generally concentrated on mapping a traditional problem into data modeling and message-sending features. Neyer *et al*. exploited the local processing of object-oriented approach for Newton Raphson load flow, which is the first paper in this area [3]. Several papers have been published on power-system modeling focusing on object-oriented analysis and design rather than focusing on a particular application [4–7]. Object-oriented design for sparse matrix operations was developed and significant amendments are still in progress [8, 9]. Zhou has defined a network container base class, which contains classes for power-system apparatus. AC load flow was treated as a derived class, which has load flow as a method [10]. Fuerte *et al*. developed object-oriented load flow program incorporating FACTS-controlled branches [11]. Foley and Bose proposed object-oriented online network analysis [12]. Little attention is paid to object-oriented development for dynamic simulation, deregulated market and distribution system modeling [13–15]. This paper deals with object-oriented modeling of power system with more stress on power-system equivalents and distribution system modeling. The main contribution of this paper is expressing the *extensibility* and *modularity* of the object-oriented design through distribution system modeling and exploiting the *flexibility* in power-system equivalent computation with a hierarchy of equivalents using inheritance, which is the key issue of *reusability*. Figure 3 shows the physical and conceptual objects modeled and application programs developed using them. Circles are the modeled objects. Individual circles represent the modular nature of object-oriented programming. The intersection of circles indicates reusability and extensibility.

2.4. *Power system computation*

Power-system computation involves huge amount of data. Representations of data and results as well as storing the data in the computer memory as per the requirement of the

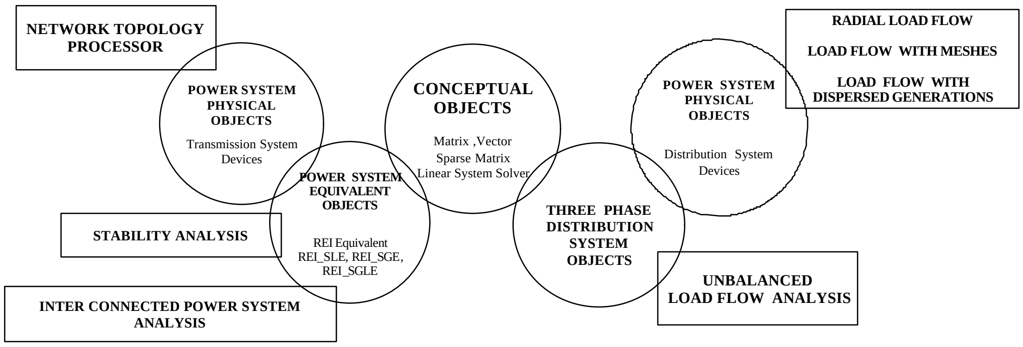
NETWORK TOPOLOGY PROCESSOR
POWER SYSTEM PHYSICAL OBJECTS
Transmission System Devices
CONCEPTUAL OBJECTS
Matrix ,Vector
Sparse Matrix
Linear System Solver
POWER SYSTEM PHYSICAL OBJECTS
Distribution System Devices
RADIAL LOAD FLOW
LOAD FLOW WITH MESHES
LOAD FLOW WITH DISPERSED GENERATIONS
POWER SYSTEM EQUIVALENT OBJECTS
REI Equivalent
REI_SLE, REI_SGE, REI_SGLE
STABILITY ANALYSIS
INTER CONNECTED POWER SYSTEM ANALYSIS
THREE PHASE DISTRIBUTION SYSTEM OBJECTS
UNBALANCED LOAD FLOW ANALYSIS

FIG. 3. Reusability and extensibility of the objects.

analysis software are the typical problems faced by power system engineers. Because of the nature of the power system network sparse matrix technique becomes familiar to represent the network matrices. In the past, programming languages used arrays to store matrix elements in Row Column Entry method. With the introduction of pointers, dynamic data structures are used for storing the matrix elements. Network equivalents play an important role in the computations involved in interconnected operation of power system. Equivalents are used in energy management system software, which requires better software modeling of the equivalents.

3. *Object-oriented design for sparse matrix computations*

Power-system analysis involves unwieldy mathematical computations. Most of the computation problems in power systems are in solving the equation $Ax = B$, where A is a square matrix and B, a vector. This makes the computation techniques to move toward matrix algebra. The matrix A is a sparse matrix because of the nature of power-system network. To store the sparse matrix elements and to perform algebraic operations and factorization on those matrices, different techniques have been used by researchers for developing efficient power-system analysis software. In this work, sparse matrix is considered as a software object though it is not a real-world object but is a conceptual object. The abstraction of the sparse matrix is a collection of nonzero elements stored in contiguous memory locations. Doubly linked list data structure (Fig. 4) is used for storing the nonzero elements. Each

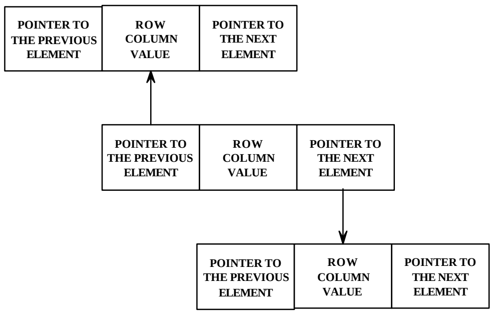
POINTER TO THE PREVIOUS ELEMENT
ROW COLUMN VALUE
POINTER TO THE NEXT ELEMENT
POINTER TO THE PREVIOUS ELEMENT
ROW COLUMN VALUE
POINTER TO THE NEXT ELEMENT
POINTER TO THE PREVIOUS ELEMENT
ROW COLUMN VALUE
POINTER TO THE NEXT ELEMENT

FIG. 4. Doubly linked list data structure.

```
Class spmatrix
 {
   Private :
      Dimension
      Number of non zero elements
      Location pointers: first, current,last

   Public :
      Spmatrix ( ) ;
      Addelement( );
      Operator +( );
      Operator (r,c);
      .
      .
};
```

FIG. 5. Pseudo-code for sparse matrix.

element in the doubly linked list data structure has two pointers to address the previous and next elements. Each element of the sparse matrix has been modeled as a structure, which includes information about its location in the matrix and its value. Figure 5 shows the object-oriented design of sparse matrix. The sparse matrix object holds three pointers. Two pointers address the first and last locations of the sparse matrix object. Another pointer, named 'current', moves along the data structure to locate a particular element. With object-oriented design it is easy to perform algebraic operations on the matrices by overloading the basic arithmetic operators. Individual elements, for instance A (2, 3), can be accessed by overloading the operator ( ), which makes user-friendly environment. The vector B in equation $Ax = B$ is also treated as an object. A method called *matvectprod* ( ) has been implemented in the sparse matrix class to obtain the product of a sparse matrix and a vector. Actually, this method has to deal with the private data of two different objects. This situation can be easily handled using the *friend function* technique of object-oriented programming.

3.1 *LU factorization*

The solution of $Ax = B$ involves factorization of matrix A. Usually, linear system solvers use LU factorization for factorizing matrix A and forward, backward substitutions to obtain the solution vector x. Figure 6 shows the basic operations involved in linear system solver. In the object-oriented design, linear system solver is designed as a conceptual object since it is a black box which takes matrix A and vector B as inputs, and performs some mathematical calculations which are not necessarily shown to the outside world and gives the vector x

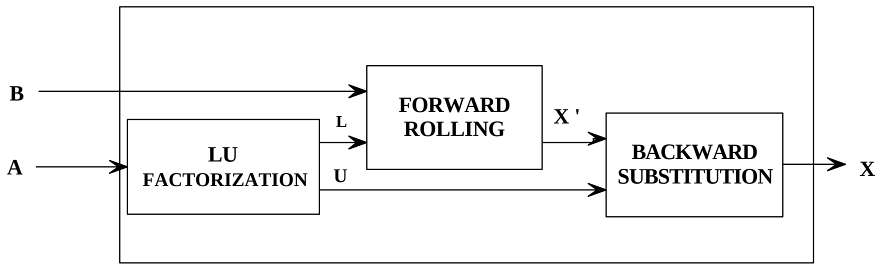

FIG. 6. Linear system solver.

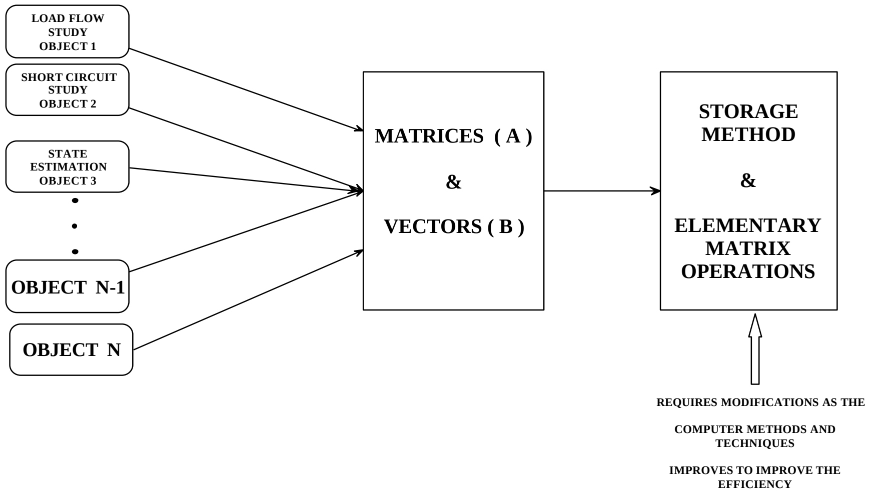
LOAD FLOW STUDY OBJECT 1
SHORT CIRCUIT STUDY OBJECT 2
STATE ESTIMATION OBJECT 3
OBJECT N-1
OBJECT N
MATRICES (A) & VECTORS (B)
STORAGE METHOD & ELEMENTARY MATRIX OPERATIONS
REQUIRES MODIFICATIONS AS THE COMPUTER METHODS AND TECHNIQUES IMPROVES TO IMPROVE THE EFFICIENCY

FIG. 7. Interfaces between objects.

as output solution. Linear system solver object has methods to calculate LU factorization, forward rolling and backward substitution. These methods can be invoked from external to the object. As discussed above, object-oriented design treats each object as a black box to other objects. They can communicate themselves to get the work done. In power-system analysis the algorithms such as load flow, fault analysis and state estimation can be modeled as different conceptual objects. These algorithm objects can communicate with the sparse matrix object (Fig. 7). The ‘Load flow’ algorithm object, while building the Y-bus matrix, gives a message *addelement* (*r*, *c*, *value*) to the object ‘sparse matrix’, where *r* and *c* represent the row and column numbers, respectively. Similarly, the overloaded *operator* (*r*, *c*) gives the value stored in the location (*r*, *c*) of the matrix to the load flow object. The load flow object does not need to know how the elements are stored in the sparse matrix object. If the data structure of the matrix object is changed, it will not affect the load-flow algorithm since load flow does not have the knowledge of internal structure of the matrix. It can only communicate with the object. But in the function-oriented methodology of software development, if the data structure is changed, then each and every algorithm needs amendments accordingly, which is a tedious process. Moreover, in object-oriented design, if some modifications are done in the algorithm objects, data structure of the matrix object does not require any changes, since for matrix object the algorithm object is a black box and it can only communicate with it.

4. Object-oriented design for REI equivalent

4.1. *REI (Radial Equivalent and Independent) equivalent*

Network equivalents are used in power-system computation to reduce the huge computational burden. REI equivalent is one such equivalent proposed by Paul Dimo, which leads to a reduced standardized net, which is radial (*R*) and equivalent (E) to the rest of the system for the node under consideration and independent (I) of any other circumstances provided the real voltages applied to the terminals of the radial branches are known [16]. Recent publications on REI equivalents show the potential application of REI–Dimo equivalent for security analysis, steady-state stability analysis and fast maximum transfer capability prediction for open access transmission in an energy management system [17]. When REI equivalents are used in energy-management system, its software modeling should be flexible and extensible which can be met by object-oriented design. This section gives the details of object-oriented modeling of various REI equivalents.

4.2. *Different types of REI equivalents*

It is possible to make three different REI equivalents for a system depending on the requirements of the applications. (1) Single load equivalent (SLE), (2) Single generator equivalent (SGE) and (3) Single generator and single load equivalent (SGSLE). Even though these equivalents differ in the number of generators and the number of load nodes they include, they are radial in nature. For the 6-bus system [4], shown in Fig. 8(a), which has three generator nodes and three load nodes, the SLE, SLG and SGSLE are shown in Fig. 8(b), (c) and (d), respectively.

4.3. *Object model for power system*

In object-oriented design, the power system is considered as an object, which aggregates the objects *node* and *branch*. With this basic abstraction, the system equivalent classes can

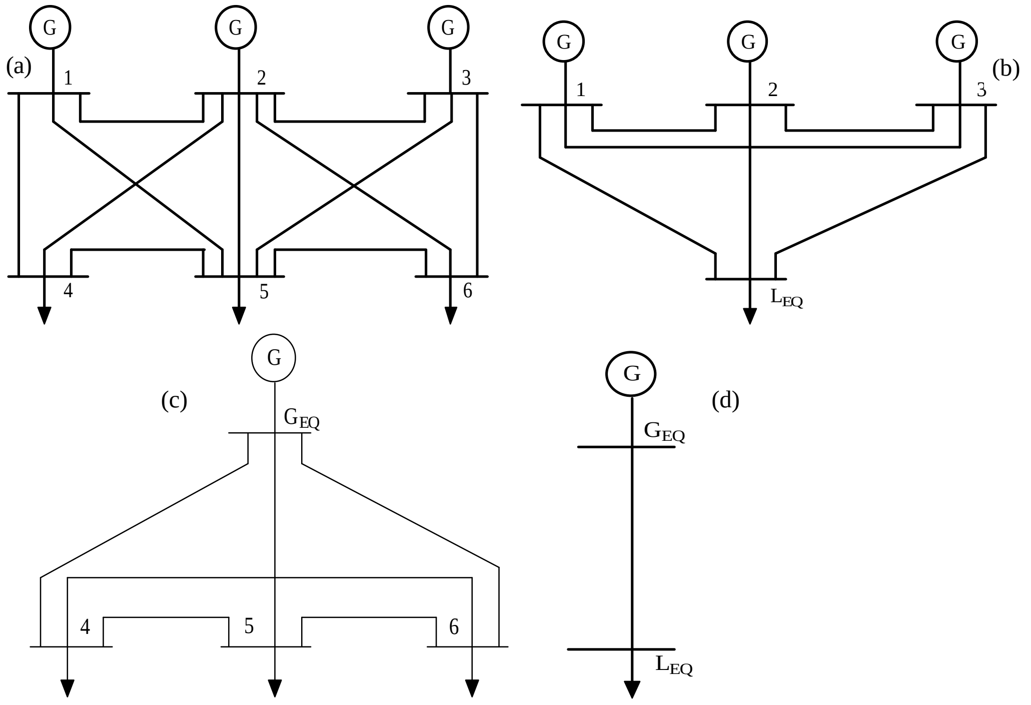

FIG. 8. (a) 6-bus system. (b) 6-bus system SLE, (c) SGE, and (d) SGSLE.

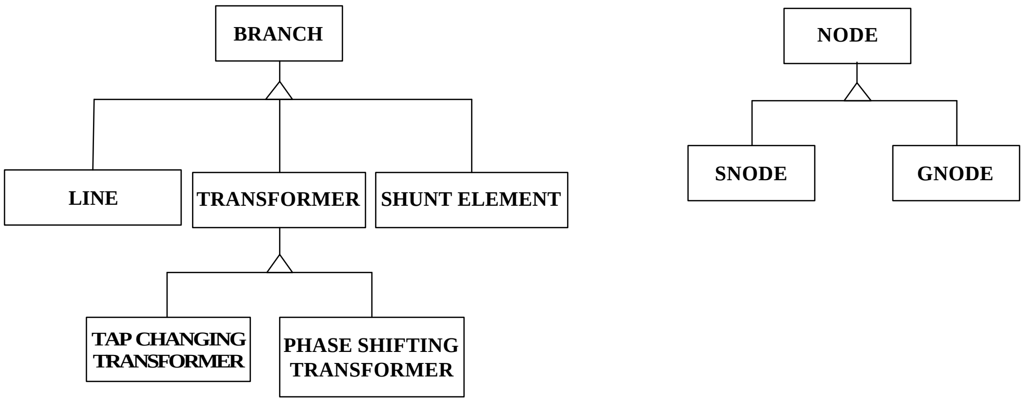

FIG. 9. Class hierarchy of power system components.

be derived. Object *system* has dynamically created arrays for nodes and branches as their attributes. The system admittance matrix is stored in a sparse matrix object. The developed *sparse matrix* class is reused here without any modifications, since our new object can communicate with the object *sparse matrix*. Thus the reusability of the object-oriented design is validated. Class *Node* and class *Branch* are the base classes. Several specialized classes have been derived from these base classes. The hierarchy of the physical objects in power system is shown in Fig. 9. *Line* and *Transformers* are the specialized classes of class *Branch* which is connected between two nodes. *Shunt element* is a special class of *Branch*, which is connected between a node and ground. Classes *SNode* and *GNode* represent the load nodes and generator nodes, respectively. Bus admittance matrix can be built by communicating with each branch object. Specialized objects *tap changing transformer* and *phase shifting transformer* respond to the same message in a different manner since the 'Ybus' of these specialized objects is not symmetrical. *Dynamic binding* and *polymorphism* have been exploited to implement the different responses of objects to the same message. Every branch object can respond to four different messages to give the four elements (Y_{ff}, Y_{ft}, Y_{tf}, Y_{tt}) of the 'Ybus' matrix.

4.4. *Object model for REI equivalent*

We are interested in node voltages and the system admittance matrix, when we form the REI equivalent. As mentioned earlier, three equivalents differ in the number of generator and load nodes they have. So it is attractive to make an abstract class of REI equivalent called 'REI equivalent', which has a dynamic array of object of class *Node* and an object of class *Sparse matrix* as its attributes. Different types of REI equivalents, which will be used in different applications, can be derived from this abstract class. Class *REI_SLE*, *REI_SGE* and *REI_SGSLE* are such specialized classes. The hierarchy of the REI equivalent is shown in Fig. 10. The object *system* has three methods called *singleloadrei* ( ), *singlegenrei* ( ) and *singleloadgenrei* ( ) to form these equivalents, when it is communicated to do so. To eliminate the nodes with zero current injections they use the method *kronreduction* ( ) of object "*sparse matrix*".

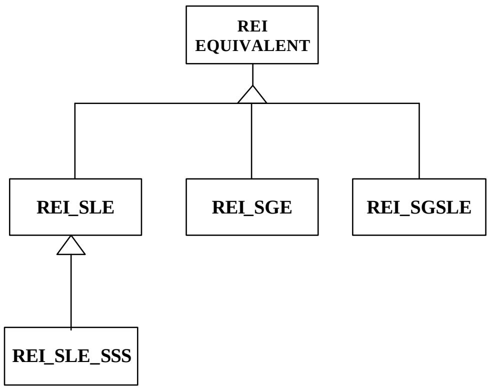

FIG. 10. Class hierarchy of REI equivalent.

Class *REI_SLE_SSS* is a specialized class of the base class *REI_SLE*. It includes the internal reactance and internal voltage of each generator. This equivalent is used for the steady-state stability analysis. In future, more classes can be derived from the base classes *REI_SGE* and *REI_SGSLE* to perform studies on security analysis and interconnected operation of power system. Figures 11(a)–(d) show the IEEE 14-bus system and its REI equivalents. Table I shows the node voltage magnitude and angle of IEEE 14-bus system for the reference state and its equivalents.

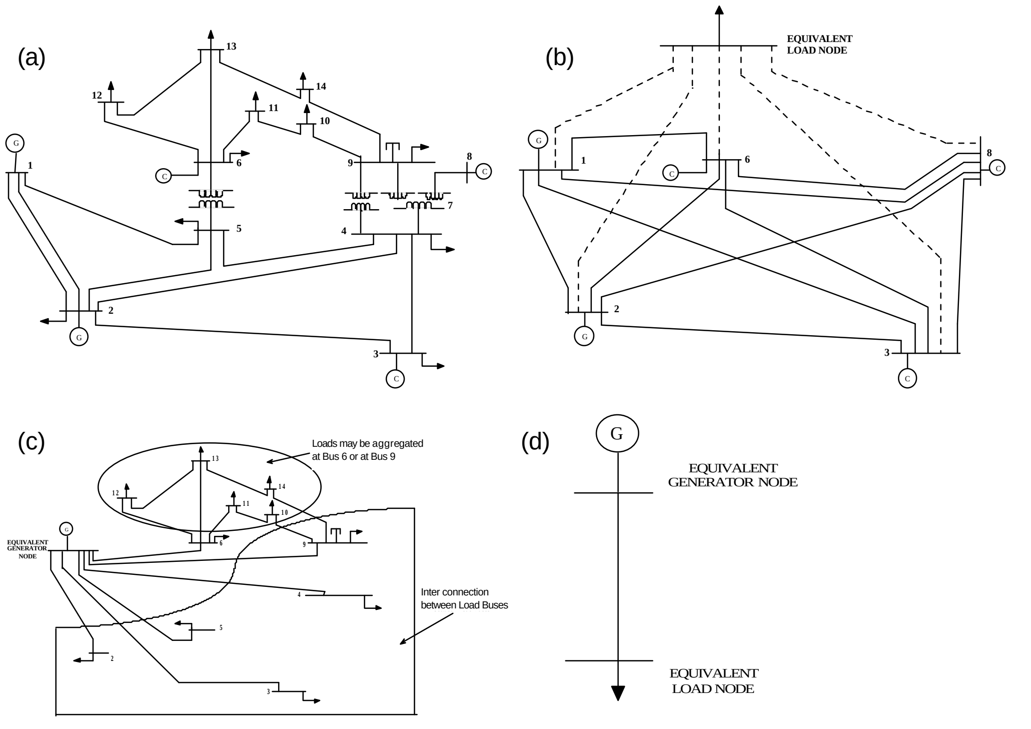
(a)
(b)
(c)
(d)
EQUIVALENT LOAD NODE
Loads may be aggregated at Bus 6 or at Bus 9
EQUIVALENT GENERATOR NODE
Inter connection between Load Buses
EQUIVALENT GENERATOR NODE
EQUIVALENT LOAD NODE

FIG. 11(a). IEEE 14-bus system, (b) IEEE 14-bus system SLE, (c) SGE, and (d) SGSLE.

Table I
Node voltage magnitude and angle (in degrees) of IEEE 14 bus system

Sl. no.	Node no.	Reference state		Single load equivalent (SLE)		Single gen equivalent (SGE)		Single gen single load equivalent (SGSLE)	
		V (Mag)	V (Ang)	V (Mag)	V (Ang)	V (Mag)	V (Ang)	V (Mag)	V (Ang)
1	**G_{EQ}**	–	–	–	–	**1.141**	**–1.7922**	**1.141**	**–1.7922**
2	1	1.060	0.000	1.060	0.000				
3	2	1.045	–4.9530	1.045	–4.9530	1.045	–4.9530		
4	3	1.010	–12.626	1.010	–12.626	1.01	–12.626		
5	6	1.070	–14.377	1.070	–14.377	1.07	–14.377		
6	8	1.090	–13.379	1.090	–13.379				
7	**L_{EQ}**			**1.029**	**–12.081**			**1.029**	**–12.081**
8	4	1.027	–10.374			1.027	–10.374		
9	5	1.034	–8.946			1.034	–8.946		
10	7	1.054	–13.379						
11	9	1.048	–14.963			1.048	–14.963		
12	10	1.044	–15.144			1.044	–15.144		
13	11	1.053	–14.886			1.053	–14.886		
14	12	1.054	–15.227			1.054	–15.227		
15	13	1.049	–15.287			1.049	–15.287		
16	14	1.029	–16.091			1.029	–16.091		

Note: Bold numbers correspond to the equivalent nodes in Fig. 11.

4.5. *Power distribution system*

A power distribution system consists of several components which are generally radial in nature [18]. Figure 12 shows the single line diagram of a typical radial distribution system. Radial distribution system is fed at a single node called root node, marked as ‘R’ in Fig. 12. Single main and several lateral feeders are there in a distribution system. Feeders comprise branches, which may be either a transformer or a transmission line section. Usually, a bus is connected with two branches and it sends end bus for one branch and receives one for another branch. Fork node is a bus to which more than two branches are connected (marked as

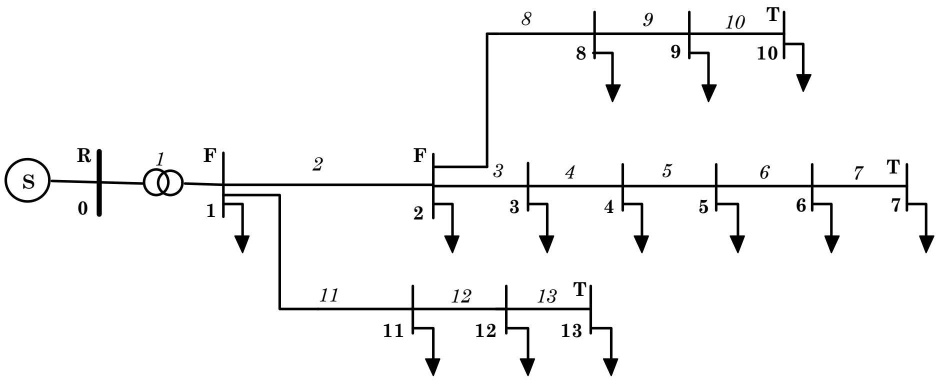

FIG. 12. Single line diagram of a typical radial distribution system.

'F' in Fig. 12). A bus, which is connected to only one branch is called terminal node and marked 'T'. The bus numbers are marked with bold letters and the branch numbers are marked with *italic letters*. A feeder section may start either from the root node or from a fork node and terminates either at a fork or at a terminal node.

5. Object-oriented modeling of distribution system

By analysing the distribution system we can come up with the physically existing components such as feeder, bus, branch, source, load and shunt devices (shunt capacitor), etc. to be modeled as software objects. The distribution system itself can be modeled as an abstract object since it is a composition of other physically existing objects. The classes *Root Node*, *Fork Node* and *Terminal Node* are specialized classes derived from the class *Bus*. They inherit all the prime attributes from the base class. *Root node* has the special property of having the source connected to it. *Fork node* is a bus with more than two branches connected to it. *Terminal node* is a bus with only one branch connected to it.

When the modeled objects are extended to accommodate the loops, a few new specialized classes have been derived. They are *Ties* and *Dummy buses*. *Tie* is an object which models the tie line that makes a loop. In the proposed object-oriented design the *Tie* is modeled as specialized class of feeder kind consisting of only one branch. It is derived from the class *Feeder*.

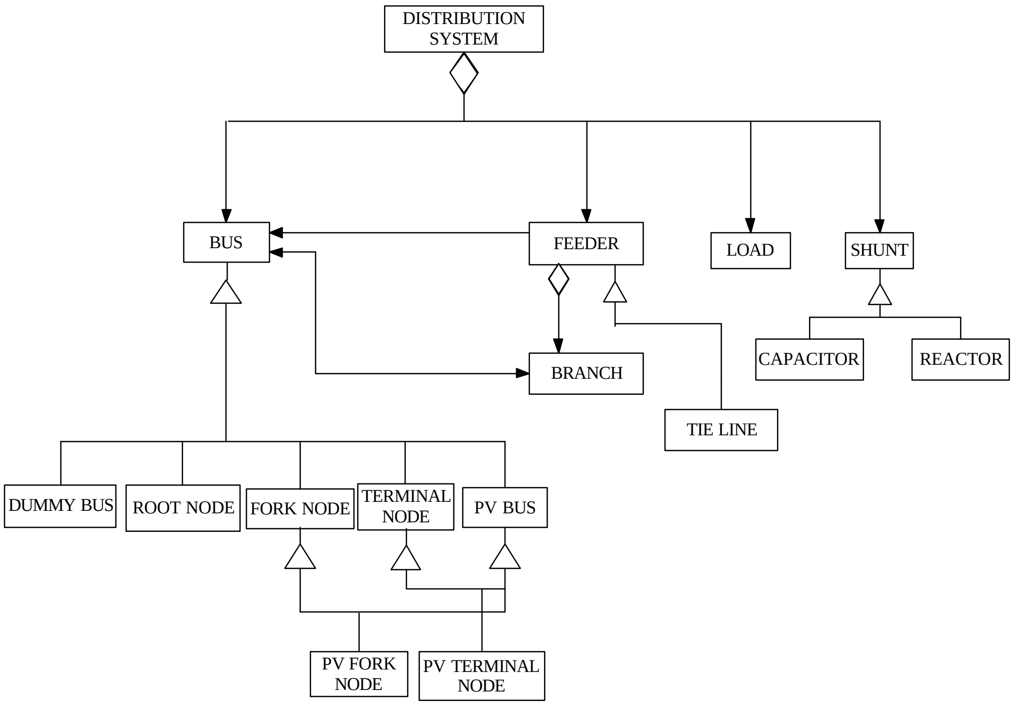

FIG. 13. Class diagram of the distribution system with dispersed generation.

Table II
Details of dispersed generation

PV bus no.	Generated power (p.u.)	
	Real power	Reactive power
9	0.35	0.17
15	0.60	0.30
22	0.80	0.40
30	0.80	0.40
35	0.40	0.20

Recent distribution networks are associated with dispersed generators driven by renewable energy such as wind, geothermal, etc. So it is essential to incorporate the dispersed generators into the radial load flow. These generators are modeled as PV buses. Compensation-based load flow algorithm gives an efficient technique to incorporate the PV buses and to maintain their voltage constant, provided the required reactive power supply is available. Object-oriented technique gives an elegant design to include PV buses in the existing distribution system model. Figure 13 shows the class diagram of the radial and weakly meshed distribution system with dispersed generation. A new class called *PV bus*, which is specialized from the class *Bu*s, is introduced. This specialization has been done because *PV bus* is a *Bus* having a generator connected to it. Moreover, in practical distribution system the local generators may be located at the fork or terminal node or at any one bus along the feeder. So a fork or a terminal node may be a PV bus. To handle this condition, the class *FPVNode* (Fork PV node), which is a fork

Table III
Bus voltage magnitudes of the 37-bus system with dispersed generation at five nodes

Bus no.	Voltage magnitude (p.u.)		Bus no.	Voltage magnitude (p.u.)	
	Without dispersed generation	With dispersed generation		Without dispersed generation	With dispersed generation
0	1.0000	1.0000	19	0.9819	0.9843
1	0.9871	0.9887	20	0.9778	0.9810
2	0.9821	0.9845	21	0.9777	0.9809
3	0.9780	0.9811	22	0.9776	0.9809
4	0.9765	0.9799	23	0.9775	0.9808
5	0.9761	0.9795	24	0.9759	0.9794
6	0.9761	0.9795	25	0.9754	0.9790
7	0.9816	0.9841	26	0.9753	0.9789
8	0.9809	0.9836	27	0.9748	0.9785
9	0.9802	0.9830	28	0.9739	0.9778
10	0.9809	0.9835	29	0.9732	0.9773
11	0.9808	0.9834	30	0.9729	0.9771
12	0.9801	0.9830	31	0.9728	0.9769
13	0.9801	0.9829	32	0.9727	0.9768
14	0.9798	0.9828	33	0.9737	0.9777
15	0.9797	0.9828	34	0.9736	0.9775
16	0.9797	0.9827	35	0.9737	0.9777
17	0.9819	0.9843	36	0.9727	0.9769
18	0.9818	0.9843			

node and is a PV bus, and the class *TPVNode* (Terminal PV node), which is a terminal node and is a PV bus, have been derived using multiple inheritance technique of object-oriented methodology. *FPVNode* is derived from *Fork Node* and *PVBus*. *TPVNode* is derived from *Terminal Node* and *PVBus*. Using the developed object-oriented model of the distribution system, load-flow analysis program has been implemented and tested on various standard test systems. Table III shows the results obtained for the IEEE 37-bus system with and without dispersed generations (details are given in Table II). These results are in agreement with the results given in Shehidehpour and Wang [19].

6. Conclusion

Flexible and extendable object-oriented design for power system has been discussed in detail. The proposed design has been used for deriving REI equivalent of a system, which can be used for different analyses in interconnected power system. Object-oriented modeling for radial and weakly meshed distribution system with dispersed generation has been discussed and it is found that the object-oriented approach is better at modeling the physical system than procedural programming.

References

1. S. W. Ambler, *The object primer–The application developer's guide to object orientation and the UML*, Second edition, Cambridge University Press (2001).
2. G. Booch, *Object-oriented analysis and design with applications*, Second edition, Addison-Wesley (1994).
3. A. F. Neyer, F. F. Wu and K. Imhof, Object-oriented programming for flexible software: Example of a load flow, *IEEE Trans. Power Systems*, **5**, 689–696 (1990).
4. T. S. Dillon and E. Chang, Solution of power system problems through the use of the object-oriented paradigm, *Electl Power Energy Systems*, **16**, 157–165 (1994).
5. J. Zhu and D. L. Lubkeman, Object-oriented development of software systems for power system simulations, *IEEE Trans. Power Systems*, **12**, 1002–1007 (1997).
6. S. Pandit, S. A. Soman, S. A. Khaparde, Object-oriented design for power system applications, *IEEE Computer Applic. Power*, 43–47 (2000).
7. S. J. Pollinger, C. C. Liu and M. J. Damborg, Design guidelines for object-oriented software with and EMS man-machine interface application, *Electl Power Energy Systems*, **14**, 1227–1230 (1992).
8. B. Hakavik and A. T. Holen, Power system modeling and sparse matrix operations using object-oriented programming, *IEEE Trans. Power Systems*, **9**, 1045–1051 (1994).
9. S. Pandit, S. A. Soman and S. A. Khaparde, Design of generic direct sparse linear system solver in C++ for power system analysis, *IEEE Trans. Power Systems*, **16**, 647–652 (2001).
10. E. Z. Zhou, Object-oriented programming, C^{++} and power system simulation, *IEEE Trans. Power Systems*, **11**, 206–215 (1996).
11. C. R. Fuerte, E. Acha, S. G. Tan and J. J. Rico, Efficient object oriented power systems software for the analysis of large scale networks containing FACTS-controlled branches, *IEEE Trans. Power Systems*, **13**, 464–472 (1998).
12. M. Foley and A. Bose, Object-oriented on-line network analysis, *IEEE Trans. Power Systems*, **10**, 125–132 (1995).
13. A. Manzoni, A. S. de Silva and I. C. Decker, Power systems, dynamics simulation using object-oriented programming, *IEEE Trans. Power Systems*, **14**, 249–255 (1999).
14. E. Handschin, M. Heine, D. Konig, T. Nikodem, T. Seibt and R. Palma, Object-oriented software engineering for transmission planning in open access schemes, *IEEE Trans. Power Systems*, **13**, 94–100 (1998).

15. A. Losi and M. Russo, Object-oriented load flow for radial and weakly meshed distribution networks, *IEEE Trans. Power Systems*, **18**, 1265–1274 (2003).

16. D. Paul, *Nodal analysis of power systems*, Abacus Press, Kent, England (1975).

17. S. C. Savulescu, Solving open access transmission and security analysis problems with the short-circuit currents method, *Latin America Power 2002 Conf.*, Controlling and Automating Energy Session, August 27, 2002, Monterrey, Mexico, pp. 1–7 (2002).

18. D. Shirmohammadi, H. W. Hong, A. Semlyen and G. X. Luo, A compensation based power flow method for weakly meshed distribution and transmission networks, *IEEE Trans. Power Systems*, **3**, 753–762 (1988).

19. M. Shahidehpour and Y. Wang, *Communication and control in electric power systems–Applications of parallel and distributed processing*, IEEE Press, Wiley (2003).