A sound design method is based upon a sound theoretical foundation, yet it offers degrees of freedom for artistic innovation. In the object-oriented paradigm, the world is viewed as a collection of objects interacting with each other to achieve a meaningful behavior.

The design perspective provided in this article can be used for many applications, such as power system state estimation and optimal power flow (OPF), and the sparse matrix class can be developed further to include eigenvalue analysis. As such, the architecture presented in this article is scalable.

**Mathematical Modeling Tools**

Digital simulation of power system applications (PSA) has been a focus of research for the last 5 decades. Considerable literature has dwelt upon development of fast and numerically stable algorithms, applicable to large systems for a variety of applications.

The domain of applications can be classified into steady-state analysis, dynamic analysis, and real-time decision making/automation. Steady-state analysis problems arise in system planning as well as operation of a power system. Such software is used either for offline analysis or online application in energy control centers. Some typical applications are load-flow analysis, short-circuit analysis, relay coordination, network topology processing, observability analysis, power system state estimation, OPF, and contingency analysis. The other category of problems involving dynamics include transient stability and small signal stability.

Mathematical modeling tools used in the two domains are distinct. Problems in steady-state analysis essentially require “good quality” large sparse linear system solvers (LSS), sparse matrix optimization, and graph theoretic computational tools. Problems involving dynamics require eigenvalue analysis, time domain simulations of differential equations, etc. This article focuses on applications belonging to the first category.

**Why Object-Oriented Programming?**

Software used presently (whether for PSA or banking systems) is inherently complex, with complexity often exceeding human intellectual capacity. For example, complexity in PSA arises because of:

- Large-scale systems to be modeled (computations)
- Volume of data to be handled (database application)
- Visualization requirements (GUI).

The task of the software development team is to engineer the illusion of simplicity. Traditionally, programs were written using structured languages (algorithmic decomposition approach), such as FORTRAN/C. Such programs are difficult to decipher. This classical design approach has the limitation of scalability. In other words, the life cycle of the program is small and is not amenable to iterative and incremental development.

The complexity of the software and the fundamental limitation on human cognition calls for an entirely different approach to software design and programming. Object-oriented programming (OOP) is appealing because human perception is largely object oriented.

Objects can be physical entities, such as a transmission line, or abstract, such as an educational program of an institution. They can be characterized by attributes, such as resistance or inductance (R or L) of a transmission line or degree programs offered by the institution. Of course, abstraction depends on the particular behavioral pattern under investigation. For example, while studying steady-state behavior of a power system, time constants of an apparatus are irrelevant and need not be modeled. The behavior of the system can be captured through methods that process the attributes (e.g., R and L) of an object (e.g., line) and derive relevant information (e.g., power flow across the line).

A class is a programming abstraction that captures the important attributes and behavior for a set of similar objects. An object is an instance of a class.

Software development is an evolutionary process. Over a period of time, specifications change. This calls
for a software development strategy that is modular, resilient to change, and provides an easy interface to the external world.

Object-oriented languages, such as C++, make it possible to achieve these objectives by development of hierarchical system through inheritance, aggregation, polymorphism, and friend relationships. It helps develop a stable intermediate platform for evolution. The ripple of change in a class can be restricted by encapsulation of data through public, protected, and private interface.

Identification and Design of Classes

First, the requirements are examined from the perspective of the classes and objects that form the vocabulary of the problem domain (analysis). After that, one invents abstraction and mechanisms that provide the desired functionality (design). Analysis is an intensive activity in classification of the system into components that can be modeled as classes.

The analysis approach presented in this article could probably be classified as domain analysis. Domain analysis can be defined as an attempt to identify an object’s operations and relationships that domain experts perceive to be important about the domain. In the present context, interpretation of the definition is somewhat blurred, because the software development team (in this case, the authors) themselves are domain experts. Therefore, to some extent, there has been an overlap in paper design, documentation, and implementation activities. The domain experts have derived expertise from available literature as well as programming experience in structured languages like FORTRAN and C.

The identification of classes is an iterative process. Identification is achieved primarily by analyzing key activities in application domain. The key activities that capture commonality are modeled by respective classes. Some of the key activities in designing PSA software can be classified into:

- Modeling of power apparatus
- Computation to capture system behavior.

Modeling of Power Apparatus

Modeling of apparatus has been well researched in literature. A rule of thumb is to design a class for each apparatus that features in the vocabulary of the problem. It is easy to design such a class because of the close correlation with the physical object.

Modeling is an application-dependant activity. The model has two types of attributes: primary and secondary. Primary attributes considered for a line are resistance (R), inductance (L), capacitance (C) per unit, length, and connectivity details, such as from-node-name, to-node-name in a substation, etc. Secondary attributes are modeled because they either feature significantly in the vocabulary of the problem domain or provide simplicity in implementation. For example, the pi model of a line is sufficient to model steady-state behavior at terminals, whereas a distributed model is required when considering applications that have to capture fast transients. At this stage, one realizes the need for a method to compute the pi model from the primary attributes of the line. Secondary attributes (like pi model R, X, B) must honor constraints that state this derivation. To read the parameters of an apparatus, the input operator can be overloaded and made intelligent enough to identify the nature of the data.

All apparatus classes form one class category called class apparatus.

Computational Activity

Typically PSA applications are computationally intensive. Computational activity can be classified into:

- Sparse matrix computation
- Graph theoretic computation.

Sparse Matrix Computation

Transmission and distribution networks have a very low degree of connectivity. As such, the resulting matrixes (like admittance and Jacobian) are sparse. Quite often, the resulting matrixes are symmetric. Considering the importance of sparse matrix computations in PSA, and symmetric sparse matrixes in particular, separate classes (sparse_mat and sym_sp_mat, respectively) have been designed to model them. It is desired that coupling between classes be minimal so that changes in one do not affect others. While the two classes appear conceptually close, the way computations are performed to optimize space and time semantics have little in common. In fact, if sym-sp-mat is derived from sparse-mat, it will not only lead to a waste of memory, but also to a suboptimal implementation in terms of time. The class sym-sp-mat has been designed using a static linked list data structure spmat (Figures 1 and 2). Only the upper triangular matrix is stored. This storage scheme differs considerably from general sparse matrix representation based on a linked list. Fortunately, in OOP, all of these implementation-specific details are encapsulated. Hence, a feeling of simplicity is engineered.

In addition to coupling, other metrics used to gauge the quality of abstraction are cohesion, sufficiency, completeness, and primitiveness. The most desirable form of cohesion is functional cohesion, in which the elements of a class provide well-bounded behavior. Figure 2 indicates the cohesive behavior of class sym-sp-mat. Suffi-
ciency implies a "good enough" interface, while completeness refers to generality for use by any client. Since completeness can be overdone, it has been suggested that a class be primitive. Access to underlying representation of the abstraction is essential for efficient implementation of primitive operations. A useful guideline while designing a class is that it should be sufficient, complete, and primitive. The interface of class sym-sp-mat is shown in Figure 2.

Operations listed in public interface (Figure 2) require knowledge and access to the data structure and hence are primitive. However, computing an inverse of sparse matrix can be performed efficiently by repeated calls (for loop) to the forward/backward substitution function (a friend relationship to the class sparse matrix). As such, it is not a primitive operation and, therefore, not listed. Methods like forward backward substitution need access to the internal representation of a class vector. Implementation using a friend relationship allows this access. It should be noted that, in order to make the class sym-sp-mat a generic class that can handle integer, real, and complex matrixes, the template facility in C++ has been used extensively.

Graph Theoretic Computation

The computational activity discussed so far is numerically intensive. Unlike this, the other kind of computation required is graph theoretic in nature. Specific applications that require such computations include network topology processing, observability analysis, and relay coordination. The structure of a symmetric sparse matrix contains embedded information of a graph. In fact, sparse matrix is one of the ways of representing a graph. However, the behavior of a graph is distinctly different from that of a sparse matrix. While considering a graph, the vocabulary of the problem domain revolves around phrases like identifying links, finding a tree (forest), or finding components of the graph. This behavioral pattern does not have functional cohesiveness with sparse matrix, even though the raw data is similar. The commonality of class graph with class sym-sp-mat is best captured by making the former a subclass of sym-sp-mat. However, the methods for the class graph give it an entirely different feel.

Another suitable scheme of representation for a sparse graph is by an adjacency list. Each node u is associated with a set S[u]. The members of set S[u] are adjacent vertices to u. A set by itself is an important mathematical tool as well as a data structure. Sets are associated with operations like union, intersection, and difference. Therefore, a class set has been designed to model set theoretic operations by overloading operators like +, -, and % for set union, difference, and intersection, respectively. Set operations essentially form the vocabulary of graph theoretic computations. In the present context, class set is a supportive class to class graph. It satisfies the important measures used to gauge the quality of an abstraction.

Ordering methods like the minimum degree algorithm used in sparse linear system solvers and QR decomposition of a sparse matrix are elegantly described by graph theoretic notations. They truly complement sparse matrix methods. As such, they are designed as friends to the class sparse matrix.

All PSA data is based on geographical locations. Application engineers are more comfortable using these names rather than integer numbers required for modeling matrixes and graphs. The class associative array maps such names to integer numbers. Use of an overloaded operator [ ] permits addressing matrixes/graphs through name arguments, making programming user friendly. At this stage, we have discussed the individual components that form the nuts and bolts of a power system application function. The class category sparse matrix and graph feature as a reusable class library. We are now poised to look at the big picture.

Architecture

Object-oriented design recognizes that the big picture of a design affects its details, and the details often affect the big picture. By zooming in on the various parts of the big picture, some strategic design decisions are taken. These decisions have resulted in formation of a class category apparatus, a class category sparse matrix, and a class graph. At this stage, applications can be discussed in the context of the logical and physical structure of a system, forged by strategic and tactical design decisions taken during development. These aspects are best captured by considering class diagrams and object diagrams. While a class diagram shows the existence of classes and their relationships in the logical design of a system, an object diagram represents a snapshot in time that captures how objects collaborate in the logical design. Undirected arcs connecting two objects are used to establish the visibility among multiple objects.

Load-Flow and Short-Circuit Analysis

Figure 3 shows the class diagram for load-flow analysis (LFA) and short-circuit analysis (SCA). The class diagram

---

**Figure 2. Class declaration for sparse matrix**

```cpp
template<class T>
class sym-sp-mat {
protected:
  spmat<T> *A;
  int no_of_rows, no_of_cols, no_of_nonzeros;
public:
  void analyze(); // symbolic factorization
  void factor(); // cholesky factorization
friend void solver(sym-sp-mat &A, const vector<T> &v, int *old_order); // forward/backward substitution
};
```

---

October 2000 45
reflects one important tactical decision, which is the creation of a class network. This is a container class that aggregates apparatus (e.g., lines and transformers) by physical containment. It also contains an instance of sparse matrix invoked by a complex data type, to create an admittance (Ybus) matrix. Methods like add_lines(int) and create_ybus( ) are defined to create sequence admittance matrixes. Subclasses LFA and SCA have been derived from the class network. Additional methods defined in the subclass implement the higher level functionality. The LSS developed in an OOP frame is versatile enough to handle both real and complex matrixes. This design highlights the commonality of algorithms and data structures used for real and complex matrixes. The design captures the commonality of algorithms and data structures used for real and complex matrixes.

The network topology processor (NTP) is an application that relies solely on graph theoretic computation. It is interesting to study how easily an NTP can be built from the existing class library. Figure 4 shows the class diagram of an NTP. It shows that class NTP is a physical aggregation of class category apparatus, class substation, and class graph. Class substation models a object substation. In addition to attributes like substation name, it contains a class topology, which is a collection of structure spmat<int> to model the circuit-breaker (CB) data and its status (on or off). All possible substation topologies can be captured by this simple abstraction. Class graph has a constructor to instantiate a graph from an array of spmat. By restricting the array spmat to one with closed CB, a graph is created that reflects the electrical connectivity of a substation. Method find_component finds the number of components of this graph and hence identifies electrical nodes of a substation. An electrical node in a substation is a set of distinct physical nodes connected through zero-impedance branches. Having established this mapping, the from and to nodes of a circuit element are mapped onto the corresponding electrical nodes. On similar lines, the graph of the network is now created. Method find_component computes the number of islands of a network.

Figure 5 shows the object diagram for NTP. Directed arcs show the methods that a client invokes on the supplier object. The number associated with the message shows the sequence of the activity. Directed arc establishes a relationship between client and supplier.

**Relay Coordination**

This is yet another interesting application that relies primarily on graph theoretic computation. Relay coordination of a meshed system requires identification of all primary backup relay pairs. If the graph of a network contains loops, then the relaying procedure becomes iterative. It is recommended that one finds a set of minimum break-point relays that opens all loops in the network. This, in turn, calls for identification of a tree and
links of the graph. This activity is common to many graph theoretic applications. Methods of class graph capture this functionality. However, methods to compute all the loops, break-point relays, and the sequential pairs of primary and backup relays are a specific activity associated with relay coordination. Hence, class relay coordination has been derived from class graph. A class relay has been defined to model a generic relay. Over-current and distance relays that have to be coordinated are derived from the class relay. Class relay coordination is the aggregation of instances of class apparatus and class relay. Over-current relay coordination and distance relay coordination problems differ in the computation of relay settings. The methodology up to computation of sequential pairs is identical for both of the coordination problems; hence, the corresponding coordination classes have been derived from class relay coordination.

For Further Reading


Biographies

Shubha Pandit is an assistant professor at S.P. College of Engineering, Mumbai, India. She is working for her PhD at Indian Institute of Technology (IIT) Bombay, India. Her research interests include OOP, software engineering, sparse matrix computations, and power system analysis.

S.A. Soman is an assistant professor at IIT Bombay. He received his PhD in 1996. His research interest include power system analysis, OOP, and computer aided protection and automation of power system.

S.A. Khaparde is a professor at IIT Bombay. He received his PhD in 1978. He is an IEEE senior member. His research interests include power system analysis, neural networks, and CORBA application to power system.