Object-Oriented Development of Software Systems for Power System Simulations

Jun Zhu, Member, IEEE
Operation Technology, Inc.
Irvine, California 92715

David L. Lubkeman, Senior Member, IEEE
Department of Electrical and Computer Engineering
Clemson University
Clemson, South Carolina 29634-0915

Abstract— The software systems developed for power system simulations have evolved into such complicated systems that the applied design practices appear inadequate to support further enhancement and maintenance. The major obstacles stem directly or indirectly from the conventional function-oriented software development methodologies. To overcome the drawbacks of these methodologies, this paper proposes applying object-oriented technology for power system software development. As an illustration of this approach, the Object Modeling Technique (OMT), an object-oriented software development methodology, is used to develop a distribution analysis system. With this technique, a generic Distribution Circuit Object Model (DCOM) is designed based on the real-world concepts and implemented with the object-oriented programming techniques. The developed DCOM is capable of supporting a wide range of distribution applications and allowing modification and maintenance over a long period of time. The performed case study has demonstrated that object-oriented technology, when applied to each stage of software development, can help achieve complicated tasks in power system simulations.

I. INTRODUCTION

During the last three decades, a large number of software systems have been developed for power system simulations. Some of these systems were designed to provide various engineering analysis for power system design, planning and training, ranging from load flow study to transient stability analysis. Others were developed to monitor and control power systems in real time. Energy Management Systems (EMS) and Distribution Management Systems (DMS) are two examples of these real-time applications. With the introduction of new computer techniques, many of these software systems have been upgraded to meet the new requirements of power system operations. The consequence is that some of these systems have become so complicated that the applied software design practices are inadequate to support further enhancement and maintenance. As a result, replacement of the entire system becomes inevitable.

One of the major obstacles in upgrading and maintaining the existing software systems stems directly or indirectly from the applied function-oriented software development methodologies. In these methodologies, the primary emphasis is put on decomposing and describing system functionality. The designed software systems are usually composed of a variety of application modules. Each of them is capable of analyzing only one aspect of power system operations and may require data input in different formats. Integration of these application modules is usually achieved by developing interfaces among them. Such an approach might be a direct and efficient way of achieving the pre-specified goals at the beginning. However, as requirements change, a system designed in this approach may require massive reconstruction. This problem and others have largely increased the cost of the development and have made software maintenance quite expensive.

Another difficulty in the enhancement and maintenance of the existing power system software lies in the way the power system is modeled in the applications. Although structured programming techniques have been successfully applied in some of these applications, modeling of the studied power system remains application-dependent. In other words, how the power system is represented depends on what particular application is being developed. As requirements evolve, the designed data structure may be subject to frequent changes. Some of the fundamental changes may propagate to all of the developed modules and require tremendous efforts to debug. For a large-scale software system, this modeling method could result in a catastrophic consequence; the system becomes unmanageable and has to be redesigned.

To overcome these drawbacks in the current power system software design practices, various strategies are being investigated. One example is the development of a Common Information Model (CIM) for the EMS environment, an EPRI sponsored project [1]. The objective of this project is to provide a standard for all shared EMS data, therefore making it possible for different multi-vendor applications to work together in an EMS environment. In general, the objective of these investigations can be summarized as creating an integrated software development environment capable of supporting a wide range of power system applications and allowing expansion and modification over a long period of time. Object-oriented technology, characterized as a new way of thinking about problems based on real-world concepts, has proved to be an effective tool to achieve this goal.

Object-oriented technology is a new methodology of software development. Its greatest benefits come from helping developers express abstract concepts clearly and communicate them to each other. When describing the application domain based on the objects that exist in the real world, object-oriented technology stresses specifying what an object is, rather how it is used. As requirements evolve, the feature supplied by an object is much more stable the way it is used. Thus, the 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 supports a hierarchical description of the application domain. This capability leads 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 modification and expansion.

Object-oriented technology has been extensively applied in the area of software development. Recently, application of this
technology to power system software development has been addressed. The applications include database management [2-3], GUI design [4], power system modeling [5-7], power system simulations [8-10] and educational tool development [11]. These applications stress applying object-oriented technology to a certain aspect of power system software development and most of the efforts have been focused on using object-oriented programming techniques to develop power system applications and supporting tools. However, the essence of object-oriented technology is the identification and organization of application-domain concepts, rather than their final representations in an object-oriented programming language. In fact, object-oriented technology can be applied throughout the life cycle of power system software development, from analysis through design to implementation. When applied to each of these stages, object oriented technology can help solve many of the problems facing the power system software developers.

This paper explores the prospect of applying object-oriented technology throughout the development of the software systems for power system simulations. As an illustration of the object-oriented approach, a Distribution Circuit Object Model (DCOM) is first designed to support the applications. Then the object-oriented programming techniques are applied to implement the designed DCOM. Finally, the benefits of using this object-oriented distribution circuit model in the development of power system applications are investigated.

II. OVERVIEW OF THE OBJECT MODELING TECHNIQUE

The Object Modeling Technique (OMT), developed by Rumbaugh et al [12], is a new methodology for the object-oriented development of software systems. The methodology involves building a model of an application domain and then adding implementation details to it during the design of a system. It consists of the following stages:

1. Analysis: The purpose of this stage is to model the real-world system so that it can be understood. Based on the problem statement, the objects and their relationships are identified using application-domain concepts. When building the model, the object-oriented analysis stresses what must be done, rather how it is done.

2. Design: This includes system design and object design. The system design focuses on designing the overall system architecture, while the object design involves building a design model based on the analysis model. The design model is described with computer-domain concepts, such as data structures and algorithms.

3. Implementation: The object classes and relationships developed during the object design are finally translated into a particular programming language. An object-oriented language is usually used to implement the designed system.

During each stage of this development, the OMT uses three kinds of models to describe a system: the object model, the dynamic model and the functional model. The object model describes the static structure of the object in a system and their relationships. The dynamic model describes the aspect of a system that changes over time. And the functional model describes the data transformation of the system. In other words, the functional model specifies what happens, the dynamic model specifies when it happens and the object model specifies what it happens to. Each model acquires details as development progresses from analysis to implementation.

Although a complete description of a system requires all three models, only the object model concept is addressed here since it captures the essential features of the technique.

The purpose of object modeling is to describe objects, which combine both data structure and behavior in a single entity. The objects with similar properties, common operations and relationships to other objects are grouped into a class. A graphic notation of the BranchDevice class is shown in Figure 1. From the viewpoint of object modeling, there are three types of relationships among the objects, that is Generalization, Association and Aggregation.

![Figure 1. Class with Attributes and Operations](image1)

Generalization or Inheritance is a powerful abstraction for sharing similarities among classes while preserving their differences. It is the relationship between a class and one or more refined versions of it. The class being refined is called the base class and each refined class is called a subclass. Attributes and operations attached to a base class are inherited by each subclass. For example, BranchDevice is the base class of Line and Transformer, and Line is the base class of OverheadWire and UndergroundCable, as illustrated in Figure 2. Attributes like fromBus, toBus, current and operations like impedance, PowerLoss, defined in BranchDevice, are shared by subclasses, Line and Transformer. Each subclass only describes its special feature. Generalization is usually called the "type-of" relationship.

![Figure 2. Generalization (Inheritance)](image2)

Association describes the physical or conceptual connection between classes. This relationship is the exact one represented in a relational database. For example, a BranchDevice is protected by a ProtectiveDevice, as illustrated in Figure 3. Associations often appear as verbs in a problem statement and contain the features of multiplicity. The multiplicity specifies how many instances of one class may relate to a single instance of an associated class. The graphic notations of multiplicity are presented in Figure 4.

![Figure 3. Association](image3)
The OMT is a popular software engineering approach. It is especially suitable for development of large software systems. Since its introduction, it has been successfully applied in the software development industry. Various commercial tools are available to support this software development approach.

III. DEVELOPMENT OF A DISTRIBUTION ANALYSIS SYSTEM USING THE OBJECT MODELING TECHNIQUE

The OMT has been applied to develop a distribution analysis system for distribution power system operations. The distribution analysis system was designed to provide a variety of distribution analysis for distribution dispatchers. Since distribution automation is an evolutionary process in most utilities, it is important to the development of the distribution analysis system using an approach that supports future amendment and expansion. The OMT was selected to achieve the goal of this incremental development.

A. A Distributed System Architecture

In order to meet the open system requirements, a distributed software architecture was proposed to configure the system. As shown in Figure 6, each component is an independent module and interacts with others through message passing.

The kernel of the system is the Distribution Circuit Object Model (DCOM), which is implemented as in-memory objects. The DCOM is built to support all of the associated distribution applications. Obviously, a unified representation of the distribution circuit is essential for achieving the high-performance system integration. The two types of data needed to build the DCOM are circuit description data and system operating data. The circuit description data, derived from the information residing in the utility distribution databases, is used to build a static distribution circuit model. The system operating data, either accessed from a SCADA real-time database or specified by users for simulations, is used to describe the operating conditions of the distribution circuits. Therefore, in the case of engineering analysis, the DCOM represents the study scenarios created by the users. When applied in on-line applications, it collects real-time information from the SCADA system and reflects the current conditions of the monitored distribution circuits.

Around the DCOM is an Application Programming Interface (API) which generates the specific data for the applications and sends the analysis results back to the DCOM. The applications can not access the DCOM directly, but only through the API. Moreover, there are no direct connections among the applications. All of the communications are through the DCOM. Obviously, this distributed configuration makes the future system expansion and maintenance convenient.

B. Object-Oriented Modeling of Distribution Circuits

As described in the previous section, modeling of the distribution circuit is of central importance to the development of the distribution analysis system. In order to support a wide range of distribution applications, distribution circuits should be modeled at a basic or low level as well as at various levels of abstractions.

By applying the OMT, a hierarchical object model can be built to describe the static structure of a distribution circuit. This object model was identified during the analysis stage and calibrated throughout the design stage. The class diagram of this object model is shown in Figure 7. As shown in this figure, the proposed object model is mainly based on the physical objects that exist in a real distribution circuit, that is, electric distribution devices. Each type of device is represented as a class, which defines both the attributes and the procedures associated with this particular type of device.

Through data abstraction, a hierarchy of the object model can be built to describe the distribution circuits at different levels. At the top level, an abstract class, Device, is identified to represent a generic distribution device. It carries only the common attributes of all types of distribution devices, such as device ID, phase configuration, etc. Each Devic object is associated with a GraphicElement object and a PropertyRecord object. The former defines the graphic representation of this device on the GUI, while the latter represents the device property record stored in a relational distribution database. Through these associations, the related device parameters can be accessed from the database and the analysis results associated with this device can be displayed in the GUI. For the on-line applications, dynamic links between the Device objects and the SCADA real-time database are also required.

The second-level object modeling is based on the physical configuration of distribution circuits. Four classes are identified and derived from the base class Device. BranchDevice represents all types of distribution devices that link two Buses, while ShuntDevice represents all types of generation and consumption devices that connect to a Bus. Distribution circuits are usually equipped with the...
protective devices to isolate the outage and reconfigure the circuits. Protectivedevices are normally installed on BranchDevices. In the case of a ProtectiveDevice connected as a tie device, a pseudo BranchDevice is created to carry the ProtectiveDevice. Using these associations, an intertwined object network can be built to match the physical configurations of the distribution circuits in the real world. In order to meet the requirements of a wide range of applications, the concrete device classes are derived at one or more low levels, depending on how detailed the distribution circuits should be modeled. For example, Line, Transformer and VoltageRegulator are derived from BranchDevice, and Load, Cogenerator and CapacitorBank are derived from ShuntDevice. As the inheritance hierarchy deepens, the concrete device object is described in more detail. Only the attributes and procedures specific to this particular type of device are defined in the class.

Another relationship modeled in the DCOM is Aggregation, which describes the circuit assembly-component relationship. This relationship is simple and straightforward; a distribution circuit contains substations and distribution feeders. Each substation or distribution feeder is composed of various types of distribution devices. Using Aggregation, the objects in DCOM can be well organized.

The proposed DCOM is only an object-oriented description of the distribution circuits for the development of our distribution analysis system. In fact, power systems can be modeled from different object-oriented viewpoints, depending on what type of software systems we are developing and what type of power systems we are modeling. For example, the object model of distribution circuits in DMS could be different from that of transmission networks in EMS. Industrial power systems could be modeled differently from utility power systems. However, what makes the inheritance really powerful is an OOP mechanism, which is also referred to as "one interface, multiple methods". It allows the one interface to be used within the abstract base class. The specific action is determined by

C. Implementation of the Designed Distribution Circuit Object Model Using Object-Oriented Programming

Implementation is the final stage of the software development. At this stage, a particular programming language is used to implement the object model developed during the design stage. An object-oriented language is an effective tool for implementation of the designed object models. This section focuses on the implementation of the designed DCOM using C++ and the object-oriented programming (OOP) techniques [13].

The most important feature of an object-oriented language is the object, a logical entity that contains both the attributes and the methods that manipulate that data. Attributes and methods specific to the object are usually defined as private or protected members, which are hidden from other class of objects. Access to the private attributes is normally accomplished through the public functions. This information hiding mechanism is referred to as encapsulation. Encapsulation prevents a program from becoming so interdependent that a small change may have massive rippling effects. In this way, the implementation of an object can be changed without affecting the applications that use it. The class definition of BranchDevice is given in Figure 8, where both data and procedures related to this object are defined in the class.

```
class BranchDevice : public Device
{
  protected:
    Bus* fromBus; // Pointer to from-bus
    Bus* toBus;  // Pointer to to-bus
    M3x1 current; // Current vector (fromBus->toBus)
  ......

  public:
    virtual M3x1 impedance(); // Impedance calc method
    virtual PowerLoss();     // Powerloss calc method
    ......
};
```

Figure 8. Example of Class Definition

Object-oriented programming languages provide strong support for the notation of inheritance. This mechanism allows a hierarchy of classes to be built, moving from most general to most specific. As shown in Figure 9, class Line and class Transformer are derived from the base class BranchDevice. These derived classes inherit all of the data and code from the base class, such as fromBus, toBus, current and PowerLoss(). Only specific data and code are added to define the derived class. This feature makes it possible to reuse code.

However, what makes the inheritance really powerful is an OOP mechanism, polymorphism, which is also referred to as "one interface, multiple methods". It allows the one interface to be used within the abstract base class. The specific action is determined by
the type of concrete objects involved. For example, a pure virtual function, `Impedance()`, is defined in the abstract base class `BranchDevice` without the actual implementation. This function is actually implemented by all of the subclasses of `BranchDevice` with different methods; the line impedance is calculated according to line size and physical configurations and the transformer impedance is calculated based on the short-circuit impedance, tap ratio and grounding type. This mechanism makes it possible to develop the distribution applications, such as load flow study, at a higher level without dealing with the detailed model of the distribution circuit components. For instance, when calculating impedance, it is even unnecessary to know whether the branch is a line or a transformer, since the pure virtual function `BranchDevice::Impedance()` will be automatically overloaded at run time by either `Line::Impedance()` or `Transformer::Impedance()`, depending on the type of the branch. Furthermore, this mechanism effectively restricts or minimizes the effects of the model changes on the developed applications. If we find it necessary to distinguish the overhead lines from the underground cables because of their difference in impedance computation, this change can be made by simply deriving the two subclasses and overloading the virtual function `Impedance()` within the derived classes. The load flow application may not require any changes, since it only handles its base class `BranchDevice`.

```cpp
class Line : public BranchDevice
{
protected:
  LineSize size; // Line size
  LineConfig config; // Line physical config.
  ... ....
public:
  M3x3 Impedance(); // Line impedance calc method
  ....
};
class Transformer : public BranchDevice
{
protected:
  double tapRatio; // Tap ratio
  ... ....
public:
  M3x3 Impedance(); // Xfmr impedance calc method
  ....
};
```

Figure 9. Example of Inheritance Implementation

An Association in the object model can be implemented using many approaches. Among these approaches, buried pointers are the easiest to implement. In this approach, an Association is simply implemented as an attribute in each associated object, containing a pointer to the related object, or to a set of related objects. For example, each `Device` object contains a pointer to its `GraphicElement` in the GUI and a pointer to its `PropertyRecord` in the database, as illustrated in Figure 10. Normally, Associations are implemented in both directions to support bi-directional traversals.

```cpp
class Device
{
protected:
  DeviceID id; // Device ID
  GraphicElement* graphicElement; // Association with GUI element
  Property* propertyRecord; // Association with database record
  ....
public:
  DeviceID ID() { return id; } // Accessing ID
  ....
};
```

Figure 10. Example of Association Implementation

An Aggregation can be implemented using a collection class object, such as a dynamic array or a list. With the collection objects, a container object can group and manage all of the component objects. As shown in Figure 11, a generic list object based on the template technique is used to assemble various distribution devices in a container object, `DistributionFeeder`. Normally, a set of methods are provided to access and manage the component objects in a container object.

```cpp
class DistributionFeeder
{
private:
  List<Bus*> allBuses; // Bus container
  List<Line*> allLines; // Line container
  ....
public:
  DistributionFeeder(); // Constructor
  ~DistributionFeeder(); // Destructor
  Bus* Bus(DeviceID); // Accessing bus based on ID
  ....
};
```

Figure 11. Example of Aggregation Implementation

D. Evaluation of the Applied Object-Oriented Approach

Based on the implemented DCOM, a variety of distribution applications, ranging from circuit topology analysis to three-phase distribution load flow, have been developed for distribution power system operations. In order to evaluate the effectiveness of the applied object-oriented approach, some of the viewpoints formed during the development of the distribution applications are summarized here.

- The developed object-oriented DCOM is capable of supporting a diverse set of distribution applications. Since distribution circuits are modeled based on the real-world concepts, such as device objects and physical configurations, the developed object model can be used by various distribution applications. The hierarchical structure of the object model allows the distribution circuits to be modeled at various levels for different applications. For example, the circuit topology analysis only requires the information from the objects at the configuration-based level, while the three-phase distribution load flow requires that distribution circuits be modeled at a low level and in more detail. This capability makes the integration of various distribution applications quite convenient, since all of the distribution applications share the same data structures.

- The object-oriented approach fully supports the incremental development of the distribution applications. The data abstraction and the polymorphism mechanisms allow the distribution applications to be developed at a high level. As the development progresses, the designed DCOM can be easily maintained and enhanced from its primitive version. The effects of the model changes on the developed applications can be eliminated or minimized. For example, the development of the three-phase distribution load flow involved deriving the branch device subclasses and the shunt device subclasses for calibration of the distribution circuit model. All other developed distribution applications were completely shielded from these modifications, since they were developed using the base classes.

- The developed DCOM provides a high-level programming environment for distribution application development. Through
the API, it provides a large number of public functions to perform basic operations for distribution circuit analysis. This high-level programming interface allows the application programmer to develop various distribution applications without dealing with a detailed data structure, thus speeding up the development of distribution applications. For example, since the API provides a variety of circuit tracing functions, the developed circuit tracing application is only a framework that integrates these functions according to the device types and tracing directions. These circuit tracing functions can also be reused by some other applications, such as trouble calls analysis, optimal network reconfiguration, etc.

- The major disadvantage of using an object-oriented language is the inefficiency in data access. Since the applications cannot directly access data from the DCOM, but only through the public functions in the API, additional overhead may be generated by the calling and returning mechanism. Although most public functions can be defined inline, thus avoiding the additional overhead caused by function calls, the virtual function’s performance cannot be improved at all. This is because virtual functions rely on a “late binding” mechanism, which refers to function calls that are not resolved until run time. Therefore, deeply nested subclasses may result in the inefficiency in data access. A tradeoff must be made between the depth of the inheritance hierarchy and data access efficiency.

IV. CONCLUSIONS

Object-oriented technology has been successfully applied in the development of large-scale software systems. This technology is more than just a way of programming. It is a way of thinking abstractly about a problem using real-world concepts. The object-oriented technology can be applied throughout the life cycle of software development and can lead to a robust and extendible software system.

As an illustration of the object-oriented approach, the development of a distribution analysis system using the Object Modeling Technique has been described in this paper. The efforts have been focused on the investigation of a generic distribution circuit model based on the object-oriented concepts and the implementation of this model with object-oriented programming techniques. The developed Distribution Circuit Object Model is capable of supporting the incremental development of a wide range of distribution applications. The performed case study has demonstrated the prospect of object-oriented technology for wide applications in the area of power system software development.

V. REFERENCES

[1] A Proposed Common Information Model, EPRI-API Project (RP 3654-1)

Jun Zhu received his B.S. degree from Huazhong University of Science & Technology, Wuhan, China, in 1986, his M.E. degree from Nanjing Automation Research Inst. (NARI), Nanjing, China, in 1989, and his Ph.D. degree from Clemson University, Clemson, SC, in 1994, all in Electrical Engineering. From 1994 to 1995, he was with Advanced Control Systems Inc., working on distribution automation. Presently, he is working with Operation Technology, Inc. as a senior research engineer. Dr. Zhu’s research interests include applications of computer techniques to power system simulations.

David L. Lubkeman is an Associate Professor of Electrical and Computer Engineering at Clemson University, South Carolina. He received his B.S., M.S., and Ph.D. degrees in Electrical Engineering from Purdue University in 1979, 1980, and 1983 respectively. Dr. Lubkeman’s research focus is on computer applications in power distribution system analysis, operations and design. Dr. Lubkeman’s e-mail address is lubkeman@eng.clemson.edu.