Object-Oriented Modeling with UML for Power System Simulation

GHERBI Ahmed
Laboratory of Automation, Dept of Electrotechnics, Faculty of Engineering, University of Setif, Algeria
a.gherbi@las-dz.com

Abstract— For the education purpose and training, the students have often subjected to the modeling and the simulation of the power system operation. The Unified Modeling Language (UML) is a language that helps to visualize, design and document models of software systems. It represents a collection of the best engineering practices that have proven successful in modeling large and complex systems. UML is widely used in designing large and reliable software systems required by banks and other corporate bodies. This paper demonstrates the use of the UML for the design and the analysis of electric power system software development.

Index Terms— Power system, Object-Oriented Modeling, UML, Education, Software Development.

I. INTRODUCTION

Modern power system has become inherently large and complex. A large number of computational software has been developed for power system analysis, planning or training ranging from load flow calculation to transient stability simulation. Energy Management Systems (EMS) and Distribution Management System (DMS) are examples of computer programs that were designed to monitor and control power system in real time [1]. For the education purpose and training, the students have often subjected to the modeling and the simulation of the power system operation. However, the use of traditional procedural and functional programming methodologies has proved their inadequacy for learning /teaching power system analysis. Recently, to overcome the disadvantages of traditional procedural softwares, the object-oriented programming (OOP) approach has gained wide spread acceptance in power system software development due to its advantages concerning software reliability, maintainability, extensibility and data integrity [2,3].

The principal aim of Object-Oriented Modeling (OOM) is to enhance the ease of use and reuse of systems and system components through the use of modularity, class-structuring, information hiding, and inheritance. In visual object-oriented systems, objects map directly to visual elements on the screen, and their various attributes and methods may typically be affected through user interactions, such as dragging element with mouse, disconnecting element, and changing element attributes by editing element properties dialog. The application of visual object-oriented techniques to power system simulation has well recognized advantage over “traditional” procedural programming [4].

OOP, which is a programming style that has been popular over the past years, focuses on bundling data and associated functions into an object. In so doing, the internal workings of the object can be hidden from other systems, with only a pre-determined public interface available. In this manner, software is more easily maintained over time, as software changes to one object do not necessarily affect other objects. It allows the construction of robust data structures, facilitating alterations in the existing software [2,3].

Different OOM techniques are in use today, each with its strengths and weaknesses. The Unified Modeling Language (UML) is one of the widely accepted methodologies for object-oriented software specification and description using visual notations [2]. The UML is a graphical representation useful for designing and understanding object-oriented systems. It provides industry standard mechanisms for visualizing, constructing and documenting software systems. It facilitates the design, development, deployment and maintenance of software systems. Actually, UML has become the modeling language of choice for object-oriented design. The application development has become rapid with highly simplified programming tasks comparing with the traditional software [2,3].

This paper demonstrates the use of the UML for the design and the analysis of electric power system software development. The power system is modeled as a set of linked objects according to a class hierarchy based on real word physical structure of the system. Objects communicate with each other by sending messages. Sending a message to an object is the same as calling a method of the object. Making use of UML diagrams, which cover the project development from the beginning until the final implementation, an object-oriented software, called "Object-Oriented Power System Simulator", (OOPSS) has been developed [5].

II. WHAT IS UML LANGUAGE?

The Unified Modeling Language is a standard language for specifying, visualizing, constructing, and documenting the artifacts of software systems before coding. It is independent of any programming language that will be used for coding the final application software. UML represents a collection of the best engineering practices that have proven successful in the modeling of large and complex systems [3]. The UML is quickly becoming a very important part of developing object-

[3]
oriented software and the software development process. The advantage of UML over other systems analysis tools is that, it reveals gaps and inconsistencies in the requirement's specification at very early stages of the design as well as providing the ease of understanding and the ability to modify visual modeling diagrams.

UML is derived from three other products: OMT (Object Modeling Technique) by James Rumbaugh, OOD (Object Oriented Design) by Grady Booch and OOSE (Object Oriented Software Engineering) by Ivar Jacobson. Since its introduction, UML had undergone continuous improvements several times until version 1.3 was proposed as the accepted standard in year 1999 [2,3]. The last version 2.1 of UML consists of 13 diagrams categorized in structural and behavioral modeling diagrams. This version provides an increased level of precision in describing the basic modeling concepts in the language and, in particular, their semantics, also an improved capability to model large-scale software systems [6].

A. 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. An object is a discrete entity with a well-defined boundary and identifies that encapsulated state (information or data) and behavior (function or operation). In addition, an object is a distinct instance of a given class that encapsulates its implementation details and is structurally identical to all other instances of that class. The object can represent a physical product or a functionality of the sub system. Therefore, the created object should characterize the adequate representation. However, one of the most important things about object classes is that the programmer, based on existing classes, can define new 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).

B. Stages of OOM using UML

In OOM, building a model of an application domain is achieved following three major stages:

a. Object-Oriented Analysis (OOA): This is a crucial phase in OOM. It is the process of analyzing a situation, system, or process in a way that focuses first on identification of things, then identifying how those things interact using the application-domain concepts. One particular method of OOA is the use of UML to describe a complex system from conceptualization to implementation.

b. Object-Oriented Design (OOD): During this phase, decisions are made about how the problem will be solved using the models from the analysis phase and the architecture descriptions. This includes System design and Object design.

c. Object-Oriented Implementation (OOI): During this phase, the developed objects and classes are finally translated in a particular OOP language (e.g. C++) to become complete software.

C. Object-Oriented Analysis and UML

Any complex system can be presented by a set of nearly independent views of a model. A single view is not sufficient. UML 2.1 provides many basic diagram types. Usually, it is not necessary to employ all diagram types - some will be more useful than others. Use case and class diagrams are used in all UML supported projects. The choice of diagrams created depends on how a problem is to be solved. These diagrams are divided into two general sets:

1) Structural Modeling Diagrams

Structure diagrams define the static architecture of a model. They are used to model the 'things' that make up a model - the classes, objects, interfaces and physical components. In addition, they are used to model the relationships and dependencies between elements.

a) Package diagrams

They are used to divide the model into logical containers, or 'packages', and describe the interactions between them at a high level.

b) Class and Object diagrams

Class diagram is used to identify classes and their inter-relationships: objects, attributes, and associations. It provides a static, structural view of a system without any consideration of time sequencing or object interactions. Similarities and differences are inspected and models optimized. A class diagram may contain classes and objects.

If there is no other class in such a diagram, the diagram is named an object diagram. They show how instances of structural elements are related and used at run-time. It defines the basic building blocks of a model: the types, classes and general materials used to construct a full model.

c) Composite Structure diagrams

These diagrams provide a means of layering an element's structure and focusing on inner detail, construction and relationships.

d) Component diagrams

They are used to model higher level or more complex structures, usually built up from one or more classes, and providing a well defined interface. They help to model the physical aspect of an Object-Oriented software system. It illustrates the architectures of the software components and the dependencies between them.
2) Behavioral Modeling Diagrams

Behavior diagrams capture the varieties of interaction and instantaneous states within a model as it ‘executes’ over time; tracking how the system will act in a real-world environment, and observing the effects of an operation or event, including its results.

a) Functional model (Use case diagram)

A use case is a description of some behavior that is required of the system, detailing how the system interacts with one or more actors. It plays a useful and important role in documenting aspects of a domain or system. An actor is an external (outside the system boundary) entity who in some way participates in the story of the use case. During this stage, Actions undertaken by those Actors, and the associated reactions of other Actors in the system, are defined and documented.

b) Dynamic model

It describes the behavior of the system. It consists on following diagrams:

1) Sequence diagram: It describes the dynamic behavior between actors and the system and between objects of the system. Data elements identified as having state are more fully developed. States are defined, logical progressions of states are documented, and triggers effecting state changes are identified. In the preparation of sequence diagrams one can verify requirement’s specification and scenarios against omissions and inconsistencies. Similarly, one can verify existing classes and objects, in case class diagrams have been created before.

2) State diagram: If an object’s behavior is more complicated, a sequence diagram is not suitable. It describes the dynamic behavior of an individual object.

3) Activity diagrams: An activity diagram shows flow control within a system. Use Cases, Class, and Sequence Diagrams are tied together. Flowcharts identifying participants interacting with classes and each other, modifying data and setting states, are defined.

III. OOA AND OOD FOR POWER SYSTEM MODELING

1) Definition of Functional Requirements

The first step in OOM is to develop a list of potential changes and needs and solicited the industry for comments on those changes. That information is collected and consolidated into a basic “functional requirements” list. This was then combined with previous knowledge to develop a basic list of functional requirements for the system.

2) Functional model (Use Cases)

The second step in analyzing the structure of the system is to identify the actors involved in the system as well as their various specific actions. Actors identified are listed below in Table I.
In this work, the structures of representative classes of the entities of the system, divided in different abstractions, are developed in packages. One distinguishes four main abstraction domains related with each other as illustrated by the graphical representation of Fig. 2.

a. **Man Machine Interface (MMI) abstraction domain:** This domain regroups all classes of MMI, as the windows, dialogue boxes, toolbars … etc.), which permit a convivial interaction between the user and the system (OOPSS) environment.

b. **Mathematical abstraction domain (ObjectMathLib):** It contains all necessary numeric calculations tools for running the integrated applications. These tools essentially regroup array operation methods usually used in the solution non linear algebraic and differential equations.

c. **Electric power system abstraction domain (Network):** This abstraction regroups the commonly used physical classes representing electrical components, e.g. busbar, transmission lines, transformers, loads, production units …etc., represented in Fig. 2 by their abstract class TDevice. Also, it contains the class TNetwork defined as a container of associated physical elements by the basic UML relationships.

d. **Object Oriented Power System Analysis Applications** (OOPSSA abstraction domain): it consists of the abstraction of the simulation and analysis applications applied to power system as: loadflow, transient stability, short circuit …etc. In this abstraction, PSApplication constitutes the basic class in the hierarchical classes of the representative applications. A dependence relation exists between abstractions, OOPSSA and TNetwork.

3) **Power System Object model**

The abstract class of the model is TDevice, which is a software object and defines any electrical system by its name and/or an integer number. Using the graphical UML representation, TDevice is represented by a rectangle with three compartments containing respectively the name, attributes and methods (operations) of the class (Fig. 3).

To be able to build power system models graphically, TDevice has been derived from the TGraphicControl component of Borland C++Builder Library [7] (Fig. 4). Thus, all descending objects from TDevice are implemented as C++Builder components that encapsulate, in a single item, not only data and methods (functions) but also graphical symbols.

Fig. 3. UML representation TDevice class

The next step of analyzing the structure of OOPSS was the dissection of the existing software specification into its core components. Object decomposition of a system reflects the physical real-world components themselves. Subject decomposition makes use of the ability for objects to inherit characteristics from other components. The design concept of power system model, called ObjectPS, is the heart of the software architecture. It consists on the aggregation of the set of associated electrical devices similar to physical real-world power system (Fig. 5).

Fig. 4. Abstract class of the proposed model

Fig. 5. Aggregation of TNetwork

Based on inheritance principle, ObjectPS has been structured as a tree. This mechanism allows building models and algorithms in a way that is easy to understand, debug and modify so that it can be applied to a larger spectrum of power system analysis problems. Fig. 6 shows a suitable architecture of the proposed prototype system using the UML notations.
This diagram describes the power system architecture based on its real world structure. This architecture considers the number of connectivities (of TBus type) of each electric element with the other elements of the system. Devices’ classes and their relationships are represented using OOP concepts such as inheritance, associations, aggregation... etc.

The topology information is stored as links between objects derived from class TSeriesDevice, objects of class TShuntDevice and the class TBus representing a busbar (node) in the electric network. From each class, derive a number of refined classes (objects). For example, this class plays a primordial role in the conception of the hierarchy of classes proposed. The class “TBranch” is a child-class of the abstract class “TSeriesDevice” and it contains all branch facilities having impedance like transmission lines and transformer subclasses. Therefore, data attributes and methods are inherited from public member of Branch class as well as the published properties and events of graphical representation. In transient stability analysis generating units (plants) are the most important elements. Electric power is generated using synchronous machines, which are driven through steam, hydraulic or nuclear turbines. The generator constitutes the important part in power system transient stability studies. The Fig.7 depicts the communication links and information flow between power plant blocks.

Therefore, the Plant class aggregates the classes TGenerator, T AVR, TGovernor representing respectively the physical elements of the generator (synchronous machine), automatic voltage regulator (AVR), speed governor...etc.

4) Applications modeling

The representation of the analysis applications by a hierarchical structure of classes tends to be much simpler than the representation of the electrical physical components, because these applications are represented as abstract concepts having no true physical structure. In this work, the analysis applications of electric network operation are represented in hierarchical classes independent of the physical elements structure. These applications are associated to TNetwork object, because each application requires a set of information from power system elements, and use some necessary mathematical methods such as: array numerical methods, algebraic and differential equations solutions, optimization problems. Fig.8 illustrates the proposed hierarchical structure of the power system simulation applications.
The abstract class is \emph{PSApplication} which includes the common operations and data of all analysis methods. The transient stability analysis is one of the most important application usually required in the conception and the operation of power system. It has been integrated in OOPSS system as a independent module called \emph{OOTSA}. The \emph{TransStab} class, which represents the transient state of the electric system, should inherit its data and its code from the \emph{LoadFlow} class. The Run() method is used for the transient stability calculation, and consists on the solving in the time domain of a set of nonlinear differential equations describing the generator dynamics. The \emph{TransStab} class communicates with \emph{ObjectMathLib} library, \emph{OODB} (Object-Oriented DataBase) and/or \emph{OOGUI} (Object-Oriented Graphical User Interface) modules and also the analysis tools class, called \emph{PSATools} (Power System Analysis Tools). This class consists on an aggregation of necessary tools, e.g. Artificial Neural Networks (ANN), Genetic Algorithms (AG), for running some applications as depicted in Fig.8.

5) Dynamic model

\begin{enumerate}
\item a) Sequence diagrams

A transient stability model is composed of set algebraic and differential equations. The sequences followed by the program for the calculation and the analysis of this phenomenon are illustrated by the sequence diagram (Fig.9). The arrows describe the messages exchanged between the objects. Indeed, in the simulation of the transient stability, a load flow study is first realized to get a set of operating conditions in steady state of the system using one of the numeric algorithms: Gauss, Gauss-Seidel or Newton-Raphson.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{fig9.png}
\caption{Sequence diagram for transient stability calculation}
\end{figure}

\item b) Activity Diagrams

The different methods and techniques used in the analysis of the transient stability vary from traditional deterministic methods to the recent methods using the artificial intelligence. In this project, we have included the most popular numerical integration methods (Euler, Euler Modified and Runge-Kutta) for the simultaneous solution of the set of ordinary differential equations associated with the machines and their control systems. An example of an activity diagram adopted for the analysis of the transient stability using a time domain method is illustrated in Fig. 10.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{fig10.png}
\caption{Example of activity diagram adopted for transient stability analysis using a time domain method}
\end{figure}

IV. CONCLUSION

This paper presents an application of UML notations to develop an object-oriented software platform for power system simulator. It has been shown that the object-oriented technology can be successfully applied in the development of large-scale software system. OOP decomposition for the power system, that follows the physical structure of the system, using a hierarchical class structure of the aggregation and inheritance relationship types, has been proposed. This software development satisfies the requirements of flexibility, extensibility, maintainability and data integrity.

REFERENCES


