APPLICATION OF THE OBJECT-ORIENTED APPROACH TO POWER SYSTEMS PROBLEMS

Tharam S. Dillon
Expert & Intelligent Systems Laboratory
Applied Computing Research Institute
and
Department of Computer Science & Computer Engineering
La Trobe University
Bundoora, Vic. 3083, Australia

1.0 Current State of Computer Technology for Power Systems

Over the last three decades, the use of computer technology in power systems has become all pervasive. It has been widely used for solving a large number of problems and the types of applications include: [58]

- transaction processing involving collecting, updating and accessing a database
- power system analysis and design through numerical simulation and calculations
- effective on-line display of information
- information transfer and availability
- monitoring and control.

More recently we have seen the entry of the new technologies of expert or knowledge based systems and neural networks.

The initial effort in the research and development in power systems computation has been directed towards:

- the development of better algorithms that are faster and have better convergence characteristics
- the development of more suitable computer based models
- development of on-line databases
- development of effective displays
- more recently the development of expert systems to address particular problems

Quite properly, the major effort has been directed at development of computer-based techniques that addressed specific problems in the power system area. However the very penetration of computers into all aspects of power systems problem solving has created large complex computer systems. These systems generally involve the interaction of several application programs. Furthermore the power systems themselves have continued to grow in size and complexity. To cope with this, power system engineers have had to build new functionality into existing programs and build entirely new programs that address issues such as voltage stability, on-line security assessment, expert system based alarm handling systems etc. The important point is that these have had to interact effectively with existing programs and databases.

Furthermore since the power system is not static but continues to grow and change, these changes have to be reflected in existing databases in a way that does not invalidate the interaction of the application programs with them. An additional phenomenon of interest over the last few years has been the need to replace existing control centres with more modern hardware. This has led, in some cases, to severe difficulties in being able to transfer existing software to these new configurations causing the ditching of considerable investment in software development and personnel training. The same has also been experienced where a major software system such as a database management system has been replaced with one utilising more modern technology and concepts.

These factors indicate that there are a number of problem areas that need to be addressed in the field of power systems computation and these are:

- to create the ability to re-use software, both application programs and databases
- to provide ease of modification and extension; this has two facets namely (i) allow for modification of the power system itself and (ii) allow for added and or modified functionality
- to provide a good correspondence between power system components and their representation in software
- to provide user friendly interfaces that interact with the power system engineer in the way that s/he perceives the problem solution
- to allow for ease of interoperability between different kinds of software technologies such as traditional application programs, databases, knowledge based systems and neural networks
- to produce a systematic and compatible analysis, design and modelling approach.

One approach that offers some hope of tackling some of these problem areas is the object-oriented paradigm. In this paper, we explore its potential for addressing these. In Section 2, we provide a brief resume of the key concepts. In Section 3, we explore the broad areas which the technology can address. In Section 4, we carry out a survey of
existing power system applications. In Section 5, we evaluate the advantages and limitations and examine future application areas and provide conclusions.

2.2 Overview of Object-Oriented Concepts

In this section a brief overview of an object-oriented system (OOS) is presented, outlining the key concepts, a complete discussion of these can be found in the book by Dillon & Tan [15]. The central notion in an object-oriented system is an object. An object is an entity that contains both the attributes that describe the state of a real-world object and the actions that are associated with the real-world object. It is designated by a system-wide object identifier or object name.

Within the context of an object-oriented language (OOL), an object would capture both the data and the procedures associated with the real-world object. The data or variables that characterise the state of an object are known as slots or attributes. The procedures or actions that change the state of an object, are referred to as methods.

Attributes characterise or describe the properties of an entity, for example, in an object representing a generator called Generator 1 (shown in Figure 1), the actual active and reactive power rating of the generator, the actual active and reactive power produced the maintenance period etc. would be real-world quantities which would be represented as attributes associated with the generator object. In addition to attributes, an object also has a number of methods or operations that can be performed by the object. In the case of the generator object, there could be a method which connects or disconnects the generator from the power system. There could also be a method which determines the actual power being supplied by the generator.

An important feature of an object is that the attributes associated with the object can only be directly altered or accessed by the methods of the object itself. However, a number of object-oriented languages do not exhibit this feature, particularly the expert system or knowledge-based system shells. It is, however, emphasised in object-oriented design. Furthermore, the methods in an object can only directly manipulate data associated with this object. This property of an object is known as encapsulation.

In order to communicate with an object one uses messages. An object with attributes and methods is dormant. It can be made active when it (called receiver) receives a message sent by other objects or sources (called sender). A message has an associated method within the object. When a message is received by an object, the associated method will be activated. This object becomes active, and at the completion of execution of the method it returns the result to the sender object. This process is illustrated in Figure 2. There is, thus, a very clearly defined public interface for an object.

![Figure 1 Instance_Object_Generator_1](image)

![Figure 2 Message-based communication](image)
An example is given in Figure 3 of an Object A involving the method, Determine_Active_Power_Supplied. The method in Generator1 performs the calculation and returns the active power supplied in Object A.

An important feature of an object is that the receiving object determines how it responds to the object. For instance, the associated method, Determine_Active_Power_Supplied, could involve quite a different technique for calculating the active power if it is a thermal generator or a hydro generator. Nevertheless, the sending object is not concerned with this. This feature of being able to respond in different ways to the same message by different objects is referred to as polymorphism.

Collections of objects with similar characteristics, such as the same attributes and methods, can be grouped together into a class. Similarly, classes with similar characteristics can be grouped together into a higher level class called a superclass. The lower level classes are referred to as subclasses. A class is itself represented by an object. Therefore, an object can be a class object or an instance object. For the purposes of this paper, when we use "class" we mean a class object, and when we use "instance" we mean an instance object.

Thus, a brown coal generator is a subclass of a thermal generator which is a subclass of the class generator. Hence the ISA link between them. These different classes can be arranged in a class hierarchy, as shown in Figure 4. Generator1 could be an instance object of brown coal generator.

Note that a class lying lower in the hierarchy is a specialisation of the immediate higher level class, and would include additional attributes or methods. A lower level class or instance object in the hierarchy inherits attributes, methods and protocols from a higher level class in the hierarchy. This provides a very competent means of storage.

There are many situations where one comes across objects that consist of subobjects or components. When one designates such objects at a higher level of description, one may merely want to know the type of the components or parts involved, without being bothered with the details that characterise each component or part. Otherwise one could end up with a very cluttered description with too much detail. On the other hand, one may need the details related to one of the parts. Such objects occur relatively frequently in engineering or engineered systems.

A composite object consists of a collection of two or more heterogeneous, related objects referred to as component objects. The component objects have a part-of or a component-of relationship to the composite object. When a composite object is instantiated to produce an instance object, all its component objects must be instantiated at the same time. Each component object may, in turn, be a composite object, thus resulting in a component-of or composition hierarchy. The composition hierarchy is distinguished from the ISA hierarchy in that the former describes a part-of relationship and the latter describes a specialisation or an ISA relationship.

In Dillon and Tan [15], it is shown how the notion of objects can be extended from their definitions for OOP to include the important characteristics of frames and rule sets so as to permit modeling of knowledge based systems. They call these extended objects by the term objects. Thus, Dillon and Tan [15] state that the notion of an object, as considered above in OOP, can be extended to an object by:

1. For objects, permitting, the attachment of demons to a slot.
2. For objects, permitting the attachment of rule sets to an object. The rule sets could be:
   (i) attached to a slot and activated when the slot is accessed in the manner of a demon or
   (ii) it could be called from within a method.
3. Allowing for pattern matching rules to be added to the system.
4. Allowing for inheritance to be stopped at any level.
5. Permitting the specification of reasoning hierarchies through the use of objectfs with associated rule sets procedures.

3.0 Areas and Stages of Use

If one examines the full range of power systems software one can distinguish three broad areas namely:

1. traditional software applications
2. database systems
3. knowledge-based systems.

Traditional software applications include both software that aids solution of power systems problems such as load-flow, stability calculations, state estimation, real-time control as well as other software in a power system utility such as accounting and financial packages, payroll systems and human resources management software.

The methods of development of this software include an analysis stage leading to the development of a model of the problem area of interest. Among the methods previously used are power system component and system modelling normally with a mathematical representation, structured analysis methods [12, 54], resulting in a data flow diagram or dataflow and control flow diagram [18]. These models generally speaking involve the procedural aspects of the system. This is normally followed by a design stage when the modular decomposition of the system is defined. Lastly, the system is implemented. By and large in the area of traditional software development, the analysis, design and implementation stages are each appropriately addressed in the power systems area, even if this is not done explicitly and consciously.

Databases include both online databases for control centres, offline databases for power system calculations such as for production planning, databases of employee as well as financial data.

In the computer science field, it is now widely recognised that the process of development of a database involves three distinct stages, namely the developments of:

(a) conceptual data model resulting from data analysis
(b) logical data model resulting from database design
(c) physical data model and implementation.

The conceptual data model tries to capture the structure of data in the real world, while the logical data model reflects in part the capability available in the DBMS to model the structures. The physical data model defines the actual physical storage structures adopted.

Unfortunately in the power systems area not enough attention has been paid to the first two stages namely conceptual data modelling and logical data modelling. The main concentration has frequently been on the physical data model. This has created considerable difficulties in interfacing new applications to existing databases.

It is a matter of utmost urgency in the future, that appropriate data models form the basis of future databases here.

The early development of knowledge based or expert systems in the power systems area frequently involve an unstructured approach with a knowledge engineer working with a domain expert, or the domain expert himself carrying out knowledge acquisition and coding pretty well in an ad hoc fashion. It is becoming increasingly understood that the development of knowledge based systems has to be carefully structured [15] with an analysis or knowledge acquisition phase, a design phase and an implementation stage. An important difference, however, is that the conceptual model developed after the analysis phase involves a model of the problem domain as well as the problem solver [15]. Hence, after the first pass through the analysis phase, it is very likely that only a partial model will be developed. This is then the basis of design and implementation of a prototype. Experience with the prototype enhances the understanding of the problem solver's solution methodology. This is then used to enhance the conceptual model.

An examination of these three different areas of software reveals that the underlying process of development involves three distinct stages as shown in Figure 5.

Thus one can distinguish three definite representations:

1. the conceptual model
2. the software structure model
3. the implementation.

The conceptual model consists of the model of the portion of the real world of interest, as discussed in the last section. It is a representation of the essential characteristics of the real world that are important for the problems that the software system is meant to address. This model is arrived at by a process of analysis or knowledge acquisition. No assumptions are made about the nature of the software structures that will be used to encode the structure of the software system. Thus, with respect to programming, no assumptions are made about the type of data
Alternatively, no restrictions should be placed on the type of knowledge constructs (production rules, semantic nets, etc.) available for the knowledge representation in the implementation of the knowledge-based system. For example, in the case of databases, no assumption is made that the typical data model or software available for implementation is a relational system or a network system. However, a set of structures has to be chosen which can be used in the description of the model. The choice of these structures will be determined not so much by the structures available in the software for implementation, but by the following factors [15]:

1. the expressive power of the structures to capture the different classes of entities, the different classes of relationships, the different types of transformations and the different classes of constraints that exist in the real world;
2. the ability to characterise both the static and dynamic properties of the real world;
3. the ability to be non-committal about aspects that cannot be fully determined in the conceptual modeling stage;
4. the ability to be able to characterise imprecision and uncertainty that exists in the real world;
5. the ability to capture state-dependent and time-dependent characteristics of a system;
6. the ability to adhere to the principle of correspondence so that structures in the real world can easily be recognised within the model;
7. the ability to capture evolution of the real world.

The process of construction of the conceptual model is a process of analysis and knowledge acquisition. Thus, in software engineering it is normally referred to as system analysis, in database systems as data analysis or functional analysis, and in knowledge-based systems as knowledge acquisition.

Once the conceptual model has been defined and verified, the process of its transformation into the software structure begins. Since the software structure defines the basis of the software implementation, sufficient attention has to be paid to the classes of structures that are available in the implementation medium when defining the software structure. In the traditional software engineering situation, the software structure is the program structure, in the database case it is the logical data model, and in the knowledge-based system it consists of the final knowledge structures and inference mechanisms to be used in the knowledge base.

During this process of transformation, the conceptual structures are transformed into the set of acceptable software structures. This process is referred to as design. The software structure model of the system should provide the data structures, the knowledge structures if appropriate, the functions, procedures and methods, the methods of control and inference if necessary, and the modules in the system.

Finally, an implementation of the software system is developed using the particular software tools selected for the task. This essentially involves coding the system using the appropriate tool. The tool could be a 3GL language such as Pascal or C, or a 4GL language or an object oriented language such as C++ or Smalltalk. Alternatively, it could be a database management system in the case of a database. For a knowledge-based system it could be a knowledge engineering or development environment shell. The implementation is translated by the appropriate tool to develop the target code for the machine.

During each of these stages, namely the development of the conceptual model, the software structure model and the implementation, it is necessary to carry out some verification and validation. The form of this is different for each stage.

The object oriented paradigm with its capability to capture static and dynamic aspects of entities is a
suitable medium for representing the conceptual model [15, 9], the software structure model resulting from the design [5] and the implementation using languages such as C++, Smalltalk and Eiffel. When it is used in all the three stages a significant advantage is that the transformation of one model to the other does not involve an entirely different set of structures. However, it is now recognised, that it can be used in any of the stages without necessarily being used in the implementation or even the design stage. Even in these cases considerable leverage is obtained.

When representing an entity by an object or objectf, it is important that all attributes and methods that are likely to be used by different applications in the system should be modelled. For a particular application a perspective or view of the object that provides the attributes or methods of direct interest here can be employed [15].

4.0 Applications to Power Systems and Utilities

The first applications of the Object Oriented Paradigm can be found in [55,56,34].

Table 1 gives a categorisation of the Object Oriented Applications by area and stage of application.

<table>
<thead>
<tr>
<th>Stages</th>
<th>Areas</th>
<th>I. Traditional Software Applications</th>
<th>II. Databases</th>
<th>III. Knowledge-Based Systems</th>
</tr>
</thead>
<tbody>
<tr>
<td>I.</td>
<td>Conceptual Modelling</td>
<td>44,40,19,50,20,61</td>
<td>5</td>
<td>44,22,23,46,50,38,11,32,37,27,31,24,7,34,51,52,62</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>44,39,48,42,30,47,22,49,35,46,23,26,46,50,38,11,32,37,27,24,7,37,51,52,63,60</td>
</tr>
<tr>
<td>II.</td>
<td>Software Structure</td>
<td>2,36,3,44,40,19,50,20,12,53,41,59</td>
<td>2,29,25,50,28</td>
<td>1,44,39,48,30,47,22,49,42,35,26,46,50,38,11,32,47,45,51,52</td>
</tr>
<tr>
<td></td>
<td>Modelling</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>III.</td>
<td>Implementation</td>
<td>2,36,3,44,40,19,12,53</td>
<td>2,29,25,21,34</td>
<td></td>
</tr>
</tbody>
</table>

Table 1 reveals that by far the greatest number of papers are in the knowledge based systems area, followed by the traditional software applications and the lowest in the database area. Another trend that is also clear is that the number of papers in Stages II and III namely, the software structure model (the output from Design) and the implementation. Whilst there are quite a few papers in conceptual modeling of knowledge based systems, the number in conceptual modeling of traditional software applications is much lower. The conceptual data models are almost non-existent for power systems. This is a serious lack and considerable effort needs to be devoted to this area.

Traditional Software Applications

Traditional software applications of the object-oriented paradigm include:

1. Development of graphical environments, graphics interfaces and man machine interfaces [12,19,41].
2. Simulation environments and tools [2,3].
3. Traditional power system numerical calculation software such as for:
   - load flows [36].
   - cogeneration plant [40].
   - switching [44, 59].
   - all power system applications software in an EMS [20,50].
4. Software related to EMS data gathering, display and communications [53, 62].

Considerable advantages are reported in the area of graphics and man-machine interfaces. These form fairly natural applications for object oriented technology.
This experience has been found in fields other than power systems, with the defacto standard software such as Windows moving in this direction. Simulation environments through the use of the object oriented paradigm appear to have successfully exploited the flexibility to use different components of a particular simulation package in different combinations for a variety of studies. Here again the encapsulation and the uniform nature of the external interface, and polymorphism give the object oriented paradigm a distinct advantage over conventional procedural based simulation environments. In the area of traditional power system numerical calculation software considerable success seems to have been achieved in the switching, co-generation, laying down a basis for all power system application software in an EMS. In the latter, the total power system application software in an EMS can be around a half a million lines of code, with several different functions seeking to access similar data. A uniform paradigm that exploits encapsulation and polymorphism will aid maintainability and replacement of specific functions with more sophisticated ones. Mixed results are reported in the application to the load flow problem, in terms of speed of processing. It is important to realise that greatly improved speed is rarely a claimed advantage of object oriented systems. Increased speed can be achieved for example with knowledge based systems with the object oriented paradigm through the use of the problem partitioning characteristics provided by ISA hierarchies (and their associated class structure) and composition hierarchies (and their associated component objects).

No strong evidence has yet been produced with regard to the success of the use of the object oriented paradigm for EMS data gathering, display and communications software. Applications in fields other than power systems suggest that this will be a fruitful area of application. Its use will provide a more open architecture for this EMS software permitting one to replace major components of this software with more advanced versions in the future. Some recognition of this is reflected in reference [5].

Database Applications

The object oriented applications in the database area include:

1. Databases for Control Centres [1,5,29].
2. Databases for providing a data interface between a traditional real time database and an expert system [25,27,28,50].
3. Databases for simulation environments [2].
4. Structuring facilities for other data types [21].

The encapsulation facility, particularly the ability to combine in a single object both the data and procedural aspects, provides significant advantages in the storage and manipulation of data. This encapsulation facilities makes easier future extension, reuse and replacement of various parts of the database or the applications software which need to interact with this.

This has generated some interest in applying the approach to control centres with a view to providing a foundation for an open architecture. This appears to be a promising avenue of future application. However, most of the effort seems to be concentrated on the design aspects, very little is devoted to the prior stage of analysis and conceptual modeling [13,14,15]. The dangers inherent in such an approach is that the major advantage of the object oriented paradigm as a facility of systematic structuring of both software and databases could be lost making interoperability between these more cumbersome than is necessary [15].

This feature of creating ease of interoperability is the major motivation for the second group of applications namely interfacing a traditional real time database with an expert system. Several designers and implementers of expert systems have noted the difficulty of having the expert system interact directly with a real time database. Hence the structure indicated in Figure 6 has been often used.

![Figure 6 An Object Oriented Database as an Interface Between a Real-Time Traditional Database](image)

Such an approach has been found to be highly effective as the knowledge structures of the expert system themselves frequently rest on the object oriented paradigm. The next category of application is to provide a database for a simulation environment. Here one notes that the flexibility arising from encapsulation and polymorphism provides the leverage gained.

The last class of application is its use to structure relational entities, where an attempt is made to exploit the hierarchical structuring facility provided by the object oriented paradigm.
Knowledge Based Systems

By far the largest number of applications of the object oriented paradigm is in the area of knowledge based systems (KBS) as shown in Table 1. The applications in the KBS area include:

1. Diagnostic systems [22,23,37,44,49,50,51].
2. Alarm handling systems [7,24,31,32,33,34,45,60,63].
3. Restoration systems [26,40].
4. Distribution system applications [7,30,39,42,48].
5. Contingency and security analysis [11,38,46].
6. Design [34,47].

Most of the diagnostic applications are for system level diagnosis except for reference [49], which is concerned with equipment level condition monitoring. The object oriented paradigm is used in two different ways in these applications. Firstly, it is used to model the power system and protection components such as circuit breakers, buses, lines, relays etc using objects. Secondly it is used as the basis of a solution hierarchy, using subsystems and a composition hierarchy. It is used both for representations in the heuristic approaches as well as the model based approaches. These group of applications, which use object class and composition hierarchies to narrow the scope of diagnosis at any stage are extremely suitable applications for the object oriented paradigm.

The alarm handling applications use objects in two ways to model the network of interest and to decompose events using an ISA hierarchy. This allows one to focus the search for the nature of the event associated with the alarms. Again these are very successful applications of the object oriented paradigm. The restoration systems use objects for representing parts of the network or restoration hierarchy.

Distribution system applications include diagnosis, alarm handling in a distribution control centre, and operational centres.

Contingency analysis and security analysis includes methods for contingency ordering and selection as well as security analysis. Design uses include configuration determination.

Note in each of these areas, the use of object oriented systems has been more recent. The early expert systems utilised rule based systems but some of the problems associated with purely rule based systems have caused researchers and developers to turn towards objects, classes, and composite objects as a means of structuring knowledge.

This structuring permits one to carry out a more focussed consideration of the rules.

An interesting new application of the object oriented paradigm is for structuring knowledge in neuro expert systems [24] that are hybrids containing both neural networks and symbolic knowledge. Another is its use in knowledge acquisition on sets [62].

Conclusions

The object oriented paradigm is likely to have a major impact in all the three areas of traditional software, databases and knowledge based systems. Arguably this will be the dominant computing paradigm in the next ten years. The characteristics of objects such as the:

- presence of a structuring facility through the use of classes, ISA hierarchies, aggregation and composition hierarchy
- encapsulation
- polymorphism
- a uniform communication interface for all objects

will assist power system computation researchers and developers to produce:

- more open systems that permit reuse and replacement of software components
- better maintainability
- extensibility and flexibility
- more focussed consideration of knowledge components such as production rules and frame like characteristics
- more systematic development of data models and databases.

However extensions will have to be made to the object oriented paradigm such as for data modeling [15] and real time semantics [57], in order for it to have suitable modeling power for the full gamut of power system problems.

REFERENCES


41. Pollinger, S. J., Liu, C.-C. and Damborg, M. J. 'Design guidelines for object oriented software with an EMS man-machine interface application'. Proceedings of the


53. Wu, F., Lo, T. and Lun, S-M. 'Object-oriented simulation of a FDDI-based distributed EMS.


