Object-oriented data model for computer aided design in electrical engineering

I. Salomie,¹ D.G. Elliman²

¹ Computer Science and Engineering Department, Technical University of Cluj, 15 C. Daicoviciu Str., RO-3400 Cluj, Romania
² Computer Science Department, University of Nottingham, University Park, Nottingham NG7 2RD, UK

ABSTRACT

The purpose of this work is to specify a set of concepts and abstractions of object-orientated nature, together with the structural relations among them, as the theoretical and practical bases for designing of ELECTROO, an Object-Oriented CAD environment in Electrical Engineering (EE).

As a first approach, we embedded the developed concepts and abstractions in an object-oriented scheme of a modeling system of power electrical engineering designs.

INTRODUCTION

An object-oriented data model is considered to be most adequate to represent data intensive, high structured, possibly recursive nested, complex objects, like electrical engineering designs.

Our purpose is to specify a set of concepts and abstractions of object-oriented nature and the structural relations among them, as the theoretical and practical bases for designing of ELECTROO, an Object-Oriented CAD environment in electrical engineering in general and power electrical engineering in particular. As a first approach, we embedded the developed concepts and abstractions in an object-oriented scheme of a modeling system of power electrical engineering designs.

Rumbaugh [5], Coad [1] and Coad [2] thoroughly approaches object-oriented methodologies for modeling, design and analysis. In Rumbaugh [5] an on-line diagram editor (OLIE) is proposed as a CAD tool for designing electrical power distribution systems. Neyer [4], has elaborated an object-oriented model based on local processing and graph traversing, for a particular example of a load flow in power electrical engineering.

Next, we shall introduce first, some fundamental concepts of our model. A Design is an ELECTROO modeling entity.
In the real world, it corresponds to an electrical engineering design. We shall use capital letters as the first letter of a word, to denote an ELECTROO abstraction. A real world design may contain schematic diagrams with names and values for components, calculational relations, constraints, etc. A design can be considered as a model of a real world complex object. A design must be integrated with the real world, therefore it must have inputs and outputs from/to real world. A design is an open entity and so is a Design.

A Subdesign is an ELECTROO modeling entity. In a real world design, a subdesign is a mean to manage complexity, using the principle of "divide et impera". In ELECTROO a Design entity may have zero or more Subdesigns. At its turn, a Subdesign may have zero or more Subdesigns. Design and Subdesign are composite modeling entities.

An ElDev, standing for Electrical Device is an ELECTROO atomic modeling entity. In real world they correspond to electrical devices like transformers, breakers, interrupters, motors, fuses, etc.

All modeling entities will be represented as abstract data types (classes) with an implementation and a behavior. The organization of the modeling abstractions are recorded in schemes. The schemes for Design and ElDev abstractions, are implemented by means of class hierarchies.

Data modeling concepts we used to elaborate the modeling schemes are based on two kinds of orthogonal hierarchies: an aggregation ("part-of") hierarchy and a generalization/specialization ("is-a") hierarchy. Using inheritance, one can add new data members and behavior to the above system supplied abstractions, reusing the already existing code with minimal effort.

MODELING OF ElDev ENTITY

In ELECTROO, ElDev (Electrical Device) is an abstraction for the set of electrical devices. Usually an ElDev has lots of associated data, out of which some are of prime importance. For example, according to Comsa [3], for a power transformer, the most important data are: \( S_n \) - nominal power, \( U_{n_HV} \) - high nominal voltage, \( U_{n_LV} \) - low nominal voltage, \( k_{12} \) transforming rate. In ELECTROO the ElDev model is defined for three levels (figure 1).

```
ElDev primary -----> ElDev user -----> ElDev Design
model model model
```

Figure 1: Levels of an ElDev model

ELECTROO conceptual model of an ElDev is shown in figure 2. The behavior part is only partly defined in the primary model, being completed through specialization, by the user, in the user's model of the ElDev, using structure and behavior inheritance.
The primary model of an ElDev consists of a class hierarchy with inheritance. The identification (Ident) part consists of an alphanumeric code which uniquely identifies the component among other components in a dictionary. The representation (Repr) part consists of a set of graphic primitives in a representation container. The connect part is based on a ConnectPoint abstraction. Each ElDev has a finite number of ConnectPoint instance objects in a connection container. The data part consists of declarations of the minimal amount of data members of ElDev. For example, for a Transformer it consists of declarations of data types suited to accommodate transformer's name (string), nominal power (int), nominal high voltage (float), nominal low voltage (float), transforming rate (float).

The behavior part of a primary ElDev model consists of a minimal set of methods that allows the ElDev to present itself (class it belongs to, name), to set and read the data members from/into an external environment, constructor and destructor for an ElDev.

The class hierarchy of the primary model is presented in figure 3.
Class Object is assumed as the basic class in OOP Languages, defining an object in its most abstract representation and behavior.

Class GObj is an abstract class, public derived from Object. It defines the structure and behavior of the most general graphic object. It is the superclass of all graphic classes with direct representation: Point, Line, Circle, Arc, Rectangle, etc.

Class Connect defines a connection point for an ElDev. A ConnectPoint object contains the index and the name of the connection point, a set associations of the connection point with electrical values and methods for computing the values, room for defining the opposite end of connection, useful for consistency checks.

Class Device, multiple inherits from GObj and Connect, therefore merges the representation and connection characteristics of an ElDev in an unique object. As data members it has two remarkable entities of container type, one for collecting ConnectPoints and the other one for collecting graphic primitives for representation.

A Device class is associated with iterators for a systematic traversing of the objects in containers. It also contains as methods, functions for adding and removing objects from containers.

Device abstraction serves as the template for all devices with electrical semantics, to be further developed like Transformers, Breakers, Motors, etc.

At the next level, the representation, connection points and main data members are set up for various classes of electrical devices. For example, for a Transformer, the RepresentationContainer will be "filled" with two Line objects and two Circle objects, corresponding to the graphical shape of a power transformer, while the ConnectContainer will be filled with two ConnectPoint objects. An instance object of this class, "knows" about itself that it is a transformer, how can it be connected and its main parameters.

**User ElDev model**

In a catalog data sheet, an ElDev has many associated data. At the same time, many CAD applications need only certain data items, beside those already embedded in the primary model. The user is allowed to extend the primary model, by specializing the abstraction associated with the primary model, generating a user defined ElDev model. For example, for a Transformer, the user can add data members for: Uscl (%) - shortcircuit relative voltages;

DPsc - active power nominal lose in shortcircuit operation and methods for converting the power transformer into the equivalent diagrams for shortcircuit computations.

Figure 4 shows that a designer can derive as many specialized subclasses of a superclass, as he or she needs. Classes in the hierarchy are represented as double line rectangles, while instance objects are represented as a simple rectangle. In the example figure: Transformer is the superclass, TransformerApp_i and TransformerApp_j are subclasses, T1_App_i, Tn_App_i, T1_App_j, Tm_App_j are instance objects of specialized classes, owned by a Design.
MODELING OF Design ENTITY

The OO data model of an electrical engineering design consists of a class hierarchy with inheritance. The root of this hierarchy is ELECTROO entity Design. A Design entity must embed the structure and behavior of a real world design.

Structural specifications
In developing of structural specifications for Design, we made use of the following criteria:
* A top-down approach of any design, using functional decomposition to manage complexity, breaking any complex Design into Subdesigns and so on.
* The possibility of reuse of previously designed and tested Subdesigns as subparts of other Designs of a higher complexity. Such Subdesigns can be parts-of other Designs.
* A two dimensional approach, on horizontal and vertical direction, figure 5, of the structural complexity of any electrical engineering Design.

In the Design space, "d1" has a marked horizontal 2D complexity, belonging to the so called "flat designs" class, while at the opposite side, "d3"
Software Applications in Electrical Engineering

has a deep vertical 2D complexity, indicating a Design with Subdesigns, which at their turn have Subdesigns.

![Diagram of Design and Subdesigns]

Identifying the modeling abstractions
For structural modeling of complex electrical engineering designs, ELECTROO uses two abstract data types (ADTs).

**ADT Generalized Tree (GT)** A GT ADT is used for modeling the global structure of a Design. A GT ensures the two necessary dimensions for modeling the structural complexity of any EE Design. The vertical complexity by means of GT recursivity, while the horizontal complexity by means of the unlimited number of sibling nodes, each node representing a Subdesign.

**ADT NodeContainer** A container is used for collecting the instance objects, members of each node in the GT. Such a container, associated with each node may own composite objects (of Subdesign type) or AtomicElement objects like ElDevs, Wires, Buses, Junctions. For container iterating purposes a NodeContainerIterator has been implemented.

For modeling purposes of the atomic objects owned by the container, at the current level of technology, we have identified the following abstractions:

**InterDesignConnections**, an abstraction used for electrical connections between composite entities, Design - Subdesign, or Subdesign - Subdesign. They act as slots when in a current Design or Subdesign, a previous designed
Subdesign is imported. A set of electric parameters are associated with instance objects of this class.

**InDesignConnections** is an abstraction which generalizes objects for local electrical connections like wires, buses and junctions.

**Electrical Devices (ElDev)**, abstraction which groups the large set of standardized electrical devices like transformers, breakers, etc.

Figure 6 presents in an example of a conceptual Design, all the above abstractions.

**Structural relations in a Design object**

After identifying the abstract modeling entities, the next step is to structure them by means of structural relations of aggregation type (part_of) and specialization (is_a).

Following Rumbaugh [5], we used diamond signs to symbolize the aggregation relation and triangles to symbolize the generalization / specialization relation. To indicate the multiplicity of the aggregation we used the symbols "o", meaning "many, in the range 0..n" and "*", meaning "many, in the range 1..n".

The structural relations among the above concepts and abstractions are shown in figure 7. In the figure, we can see that a Design can aggregate an indefinite number of Subdesigns and at least one AtomicElement, while a Subdesign can recursive aggregate an indefinite number of other Subdesigns. Termination is ensured by Subdesigns having only Atomic elements in the...
associated containers of member objects.

**Behavioral specification**

Each instance object of a Design or Subdesign class will have a behavioral specification.

At the topmost level, entity Design, it is likely the associated methods to have an overall character (to propagate through all the design).

At this level, we specified methods for presentation and report generation, different queries on electrical consistency and load flow, generating application dependent views of the model, the list being opened for new methods. Actually, the user is not allowed to modify the structural relations among entities, but is desired to add new behaviors to the modeling entities at any level, by specializing classes.

At the same time, this level provides the interface for an incremental construction, save and edit of the model entities.

**CONCLUSIONS AND FUTURE WORK**

The proposed modeling system has been partly implemented in a Borland C++ environment and tested for its capabilities of topological modeling on an electrical power supply system for a factory department with one small workshop consisting of seven Subsystems at the root level: the power transformer station 20kV/0.4kV, workshop feeder panel, a lathe machine subsystem, a milling machine subsystem, a drilling machine subsystem, lighting subsystem and fanning subsystem.

The maximum 2D vertical complexity of the power supply design was of 3 nested Subdesigns.

The necessary data for populating the Design with specific entities have been supplied from files.

For future work, we plan to specify and implement an interactive Layout Editor for a WYSIWYG approach of model generation, support for persistent objects and versioning.

**REFERENCES**