

Concepts of Aspect-Oriented Modeling Applied to Optimal Power Flow Problems

Daniele A. Barbosa¹, Leonardo M. Honório¹, Armando M. Leite da Silva¹, Cristina V. Lopes²
¹Federal University of Itajubá, ²University of California, USA

Abstract - Optimization of complex systems demands advanced methods that are implemented in specialized software. Multiple combinations of optimization methods, objective functions, and constraints further complicate the problem of developing this software, making it hard to create, maintain, and evolve. To overcome this problem, this paper presents a new development methodology based on ideas of Aspect-Oriented Programming (AOP) applied to optimal Power Flow Problems. This new methodology supports a clean separation of concerns, and keeps dependencies to a minimum. The optimization method is self-contained and completely independent from the rest of the system; for each optimization scenario, the solution binds the optimization with the concrete problem at run-time. This approach improves the ability to deal with several different objective functions and constraints, providing flexibility, maintainability, and usability to the development and evolution effort without degradation of the computational time. To evaluate this model, it is compared with traditional OOP paradigm using several software metrics.

Index Terms — Aspect Oriented Modeling, Intelligent Systems, Optimal Power Flow, Software Paradigms.

I. INTRODUCTION

Optimization is an essential analyses tool when one needs to deal with the problem of search a complex solution space given one or more specific objectives. There are several domain problems with different characteristics where these tools are essential, such as chemical process, operational research, scheduling, electric power flow, etc. Among these, one of the most appealing domains is the electrical optimal power flow (OPF) problem. Basically, it consists on solving one – or more in case of multi-objective scenarios [1] – objective function, i.e. loss reduction, subject to several hundreds of technical and economical constrains. Mathematically, an electrical power flow can be seen as a huge non-linear and non-convex system with several sets of control variables that can be discrete or continuous. This variety makes the power flow problem to be divided into classes depending on what problem is the current focus. Each problem may present different mathematical characteristics such as linear, non-linear, combinatorial, dynamic, stochastic, and others [2]. In order to deal with these issues, several optimization methods were applied. The most common ones present in the literature are: Simplex, Interior-Point, Conjugate Gradient, Hill Climbing, Tabu Search, Genetic Algorithms, Ant Colony, Particle Swarm Optimization, Artificial Immune System, etc [3]-[10].

There are several commercial solvers dedicated to these problems. The main idea behind these solutions is to provide libraries with different optimization methods that enable a

developer to incorporate these functions in their own application. This is the base of structured-programming paradigm and there are several advantages and disadvantages of using such technique. One major problem is the lack of scalability. Basically, an optimization problem depends on one or more objective functions and a set of equality and inequality constrains. In an electrical power flow these elements have strong relationships among each other and must be combined depending on which problem is faced, generating thousands of possible scenarios. Due to that, specialized functions must be developed to work with all these possibilities, where each function is dedicated to a specific scenario and, any change in the problem –i.e. a new control variable – impacts in chances in all of them.

Trying to avoid some of the related problems, one could implement the optimization and power flow methods using OOP paradigm. However, an effective and efficient OOP approach requires subdividing the problem into classes, each one containing methods and properties of a well-identified part of the problem, and then composing them by mechanisms such as method calls and inheritance. Loose coupling between the classes leads to an increase in flexibility and maintainability, since a change in the code of a class does not necessarily imply changing the others. However, when the system has the complexity and dependency of optimization scenarios, software development problems still persist. Although it is possible to subdivide the problem into appropriate classes, their composition to assemble the optimization solution results in a complicated graph of dependencies that defeats the advantages of using OOP.

The search for better software development methodologies led us to Aspect-Oriented Programming (AOP) [11]-[13]. AOP is a programming paradigm that has been proposed to address problems such as the one described above. It works decomposing the problem into completely independent parts that will be further compiled together. It is impossible to generate the same solution using a different paradigm because AOP supports new composition mechanisms that allow new kinds of components, and that greatly decreases the number of dependencies among the classes. In AOP, problems are decomposed and modeled following the domain knowledge. Some parts of that model are composed with the others using OOP mechanisms, called components, but others require more advanced composition mechanisms, called aspects of the problem or aspectual components. In our case, these aspects are developed apart and are loosely coupled with each other. In the composition phase, the components and the aspects are put together by a process

called weaving, resulting in the final system.

The literature presents several reports about different approaches of how to apply and achieve AOP functionalities [7], [11]. The application of this paradigm in an OPF system represents breaking all the strong relationships present in the system: the inner elements of the electrical power flow domain and the outer components representing the optimization methods.

Several electrical power flow situations will be analyzed to demonstrate these benefits. A system with 3 optimization methods, 4 objective functions, and any combination of available control variables will be implemented and tested. For that, this paper is presented as follows: Section 2 describes the electrical optimal power flow problem; Section 3 presents the optimization methods used in the solution. In Section 4, problems related to the development of an OPF using OOP methodology are shown, in Section 5, an approach based on Aspect-Oriented Modeling is presented to avoid these problems and finally, Section 6 shows the conclusions.

II. THE ELETRIC OPTIMAL POWER FLOW PROBLEM

An electric power system [14] is composed of several electrical equipments such as generators, transmission lines, transformers, shunt capacitors, etc. Its main goal is to generate, transmit, and distribute electric power to customers obeying several constrains. The most important one is that it is not possible to store electrical power and, therefore, the amount of generated energy at any given time must be the same as the amount consumed, duly discounted the transmission losses. In order words, the power flow balance must be null. To fulfill these conditions, human operators must handle hundreds, sometimes thousands, of variables in order to control the system, driving energy flow from sources to consumers. During this process, the system must remain physically stable, from both static and dynamic points of view, technically reliable, and economically interesting for all market agents involved in this process. The previously described conditions can be mathematically stated as follows:

$$y = (M_{cl}). \quad (1)$$

$$g_i(V, \theta, y, x_{cb}) - g_G + g_L = 0$$

Where y is the system admittance matrix; x_{cl} is the control variables related to the models of the transmission elements (e.g. lines and transformers); x_{cb} the control variables related to the models of the buses (i.e., power stations); g_i , g_G and g_L are the vectors of power injections, power generations and power loads at each bus, respectively. If any change happens in the control variables, the system is driven into a different operating point, changing values of several measures such as voltage magnitudes V and angles θ (named as state variables), and also active/reactive flows, etc., spread along the power network. This leads to a condition where finding the best operation point under a desirable scenario is challenging. This problem is named as optimal power flow or simply OPF.

As previously stated, an OPF is a non-linear, non-convex

and large-scale problem, mathematically described as follows:

$$\begin{aligned} & \text{Minimize} && f(x_s, x_c) \\ & \text{subject to} && g(x_s, x_c) = 0 \\ & && \underline{H} \leq h(x_s, x_c) \leq \overline{H} \\ & && \underline{x_s}, \underline{x_c} \leq x_s, x_c \leq \overline{x_s}, \overline{x_c} \end{aligned} \quad (2)$$

Where x_s is a $ns \times 1$ vector of state variables (voltage magnitudes and angles at load buses, etc.); x_c is a $nc \times 1$ vector of control variables (voltage magnitudes at generation buses, power generation, shunt capacitor allocation, transformer taps, load in buses, etc.); f is a scalar function representing a power system quantity to be optimized (e.g. economic dispatch, transmission loss reduction, transmission equipment overflow, loadability, load shedding); g is the active and reactive power balance equations; and h is a $m \times 1$ vector of constraints associated with the limits of some network power values such as transmission lines flows, reactive generation, etc.

There are several problems related to different operational situations that involve solving (2). Some of those are presented in [15,16].

To design an Optimal Power Flow software, it must be taken in account the capability to deal with all these problems besides the ones regarding the way of how to operate an optimal power flow system where, in order to reach a good operating point, several controls, objective functions and constrains must be tested, leading to a scalability problem. To handle this option interlacement, several logical conditions must be considered in order to attend each variation as the ones shown in Figure 1. Taking in account just three optimization methods, the multitude of optimization formulations yields, at least, 192 possibilities if only one type of control is accepted, or over 967.000 possibilities ($3 \times 8 \times 8!$) if combinations among controls are considered. The code effort to treat these logical switches is a huge problem in OPF softwares.

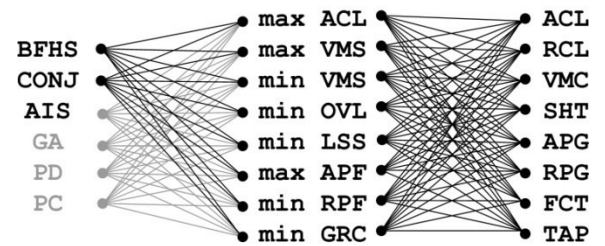


Figure 1. Possible configurations that an OPF can appear

Where:

- BFHS - BFHS Optimization Method
- CONJ - CONJ Optimization Method
- AIS - Artificial Immune System Opt. Method
- GA - Genetic Algorithm Optimization Method
- PD - Primal-Dual Optimization Method
- PC - Predictor-Corrector Optimization Method
- MAX - Optimization Objective to maximization;
- MIN - Optimization Objective to minimization;
- ACL - Active Load (loadability)
- ACL - Active Load Control (load shedding)

APF	-	Active Power Flow
APG	-	Active Power Generation Control
FCT	-	FACTS Control
GRC	-	Generation Cost (economic dispatch)
LSS	-	Transmission Losses
OVL	-	Transmission Overload
RCL	-	Reactive Load Control
RPF	-	Reactive Power Flow
RPG	-	Reactive Power Generation Control
SHT	-	Shunt Susceptances Control
TAP	-	TAP control
VMC	-	Voltage Magnitude Control
VMS	-	Voltage Magnitude State

III. ELECTRICAL POWER SYSTEMS OPTIMIZATION

One of the issues related to OPF solution process is that there are several sets of problems where, for each one, a different method of optimization is the most indicated. For instance, if one is studying electrical expansion planning, the Mont Carlo methodology along with a Linear Optimal Power Flow Algorithm is the best approach. In other scenarios, the optimization technique depends on the type of the control variables (integer, complex, real or mix), the number of used objective functions and the problem itself. This makes impossible for just one optimization methodology properly fulfill all these necessities. Thus, it is desirable for optimal power flow softwares to have access to different approaches. To demonstrate the capability of AOP of dealing different types of optimization techniques, four different methods were implemented; Best-First Heuristic Search[7], Conjugate Gradient[7], Artificial Immune System[8] and Primal-Dual Interior Point[16].

IV. OPF SYSTEM DEVELOPMENT USING ORIENTED-ORIENTED MODELING

The OPF system developed takes into account scenarios where a user can choose the optimization method, the objective function, and a combination of control variables and limits. The simplified OOP conceptual model of this system is shown in Figure 2, and will be explained next.

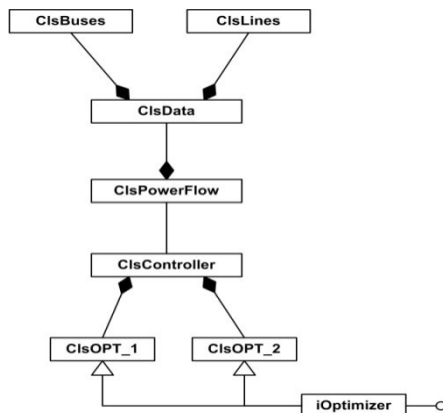


Figure 2. Simplified OOP-OPF conceptual diagram

First, it should be noted that there are several ways to design OPF using OOP. The OOP diagram Figure 2 shows a design centered on the *ClsPowerFlow* and *ClsController* classes. They encapsulate the tangling of information about the optimization methods with the information of the concrete domain. The *ClsPowerFlow* class holds mathematical and physical information used to carry out any action on electrical power systems. It is responsible to provide the objective functions and constrains for both functions as transmission line flow and variables as voltage in a given bus. This knowledge has a strong relationship with its variables, represented by *ClsBuses* and *ClsLines* that were encapsulated into the *ClsData* class. The *ClsPowerFlow* class along with its dependencies has by itself the capability to deal with ordinary power flow problems. Analyzing the other side of the diagram, the *ClsOPT_1* and any other class that implements an optimization method has strong relationship with the *iOptimizer* interface. It enables the utilization of dynamic dispatch (DD) ensuring lower maintenance because if a new optimization class is added, the only part of the code that needs to be changed is the one related with the logical treatment (inside the *ClsController*), the rest of the code remains unchangeable. The *ClsController* contains all variations regarding the optimization method, objective function, chosen constrains, and control variables. It is responsible to manipulate all the necessary logical treatment to carry out the system. These logical switches are divided in two parts. List 1 shows part of the Controller class where the optimization method, and the objective functions are instantiated, List 2 shows the part that deals with selection of control variables, and List 3 shows the part that calls the optimization method.

List 1 – Snippet of code 1 from Controller class

```

public ClsController
...
iOptimizer optimizer;
DlgFunction dFdx;

switch (method) {
case "CON":optimizer = new ClsOOPCON(); break;
case "BFHS":optimizer = new ClsOOPBFHS(); break;...
...
switch (objectivefunction) {
case "f_PowerFlowAtualization":
optimizer.OFunction = new
DlgFunction(nOPFlow.f_FlowAtualization);break;
case "f_TransmissionFlow":
... break; . . . }

```

List 2– Snippet of code 2 from Controller class

```

for (int i = 0; i < Length; i++){
switch (Ctrlvariable[i]){
case "C_XiFact":
Ctrl[i]=OPFlow.obLines.C_XiFact; break;
case "C_Tap":
Ctrl[i]=ObPFlow.obLines.C_Tap; break;
case "S_Loss":
Ctrl[i]=ObPFlow.obLines.S_Loss; break;
...}

```

List 3– Snippet of code 3 from Controller class

```

switch (method) {
case "CON":optimizer.run(); break;
...
case "PDIP":
if {ObPFlow.Check_For(f_dFdx_FlowAtualization)}
{ dFdx =
DlgFunction(nOPFlow.f_dFdx_FlowAtualization);
Optimizer.run(dFdx,PARAMETERS)}
Else { Return Error("unvalid dFdx");}break;... }

```

An analysis of List 1 shows two OOP techniques that are known to be very useful: (1) the optimization methods are specific classes that inherit the *IOptimizer* interface. Due to that, it is possible to use dynamic dispatch. Thus, one unique object, the optimizer is capable to holds any optimization method depending just from the instantiation. In that way, the rest of the code does not need to be prepared to changes in the optimization domain. In (2), the objective function is set using method delegates, a technique similar to function pointers in C enabling to pass methods around without having to know, at compile time, which one will be invoked. The code in List 2 sets the control and state variables according to the specifications of the user. List 3 shows a major drawback on using OOM methodology when a more complex logical is used; some optimization methods demands different parameters than orders, for instance, if the primal dual interior point is chosen, it is necessary to have the first and second derivatives of the objective function. Nevertheless, if other method is chosen, the derivatives are not needed and could not be implemented. To work with scenarios like this, the OO software must provide check codes like the one in List 3.

V. OPF SYSTEM DEVELOPMENT USING ASPECTS AND REFLECTION

An AOM project consists in two types of components; OO-based and aspect-based. First, the problem is modeled and decomposed according to the domain knowledge. The traditional components of the model are created and subdivided using traditional OOP paradigm, but in a way to avoid cross-cutting. The aspects are responsible for describing high-level features that were not considered at first place and handle parts of the system with high variability and interdependencies among those and the other components. Therefore what differentiates an aspect from a regular component is its composition with the rest of the system. A regular component takes an element of the model and subdivides it using OOP relations both internally and externally. An aspect takes an element of the model and subdivides it using OOP relations only internally. The connections between an aspect and the rest of the system are defined in a third element: a binder. The binder uses meta-information about the components, such as their classes, their methods, etc., in order to compose them. The composition of the system is done through a *Weaver*, an element that takes the binding instructions and composes the different components together.

A. The OPF-AOP Solution

The AOP methodology is a very powerful paradigm but requires more time to modeling and coding. Considering an OPF scenario, it is possible to apply AOP in several parts as error handler, jacobian assembly and others. However, using aspects in all these layers would yield highly complex software and it would lose the advantage of AOP. Thus, it is only worth applying this methodology in cases where using regular OOP would be difficult, like in the situation shown in Figure 1.

Trying to apply software engineering best practices, the OPF modeling using AOP is explained as follows. Starting with the analyses provided by the OOP paradigm (Figure 4), it can be seen that all classes are connected among them through dependences. This scenario implies that cross-cutting functionalities are spread across the system and changes in one class impact others.

Standardization is the main key to correctly designing a dynamic AOP solution. In this work, methods and variables in question were implemented according to specific interfaces. This is because the reflection procedure enables reading and calling, during run-time, any function or variable. To use this characteristic in an OPF solution means that a user can get metadata of specific component and use them in a generic domain, represented in this article by the weaver. The metadata is given by objects containing all types of information regarding classes. To better understand this concept, consider a class *ClsBus* that stores, at *objBus*, all variables, properties and methods of a power flow bus, and reason as follows. Suppose that it is necessary to analyze a given field, the bus active power flow generation represented by *C_Pg* for instance, present at *objBus*. Using traditional OOP methodology, it would be necessary to have pre-implemented a logical treatment in order to identify the desired field and, then, return its value. However, using reflection, it is possible to get its value using the meta-information, as shown in List 3.

List 3 –Field Invocation through Reflection

```

public class ClsBus {
public double C_Pg; // active generation
public double S_Teta;// angle
...
public double FieldValue(string fieldName)
{ 1Type myType = this.GetType();
2FieldInfo Minf = myType.GetField(fieldName);
3return (double)Minf.GetValue(this);}

```

The reflection protocol for doing this procedure is present as follows:

- Tag 1 – the specific type of that (meta)object is retrieved. In this simple example, the type is *ClsBus*, but in a real implementation it could return several different types.
- Tag 2 – the metadata of a desired field (fieldname) is retrieved and stored in *Minf*. This field represents the meta-information of a control or state variable whose name is given as a string, i.e. “C_Pg,” “S_Teta,” etc.

Tag 3 – the meta-information is used to retrieve the value of the desired field and its value is returned as double.

The noteworthy property of this methodology is that the names of the power system elements and of the state and control variables of the optimization problem are variables themselves – they are not known at programming-time. By using these (meta)variables rather than hard-coded variable names and types, the complicated logic pertaining to mapping the user’s problem formulation with the internal types and functions of the program is avoided.

Figure 3 and Figure 4 show how interfaces were used in this implementation. Regarding the power flow domain, all functions present in the system as LSS, OVL, and others, have used the *IFunctions*, while all variables have used the *IBusVariables* or *ILinesVariables* interface. Regarding the optimization aspect, both methods were implemented using the *iBaseOpt*, *iGenericFunctions* and *iGenericVariables* interfaces. The last two are particularly interesting because they are abstract variables and functions that are connected with the real ones during run-time. If any other method was implemented, it would also use the same set of interfaces.

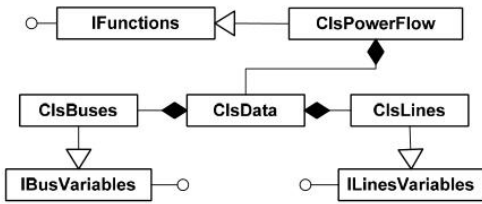


Figure 3. Power Flow Aspect

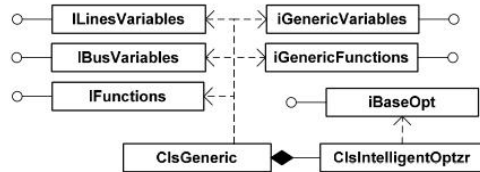


Figure 4. Optimization Aspect.

To apply this technique in the OPF problem, we designed the weaver component/class diagram shown in Figure 5. It has two classes, *ClsGeneric* and *ClsIntelligentOptz*, which are responsible for accessing meta-information from the interfaces. The dot-connections present in the figure represent no strong connection among these elements. Furthermore, the classes are only prepared to access the meta-information contained in the interfaces and, thus, any change in a variable, function or optimization method would have no impact in these classes.

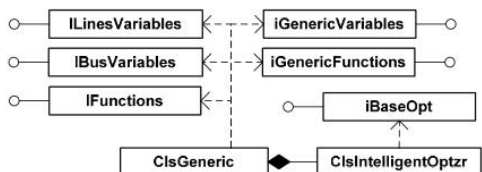


Figure 5-OPF using Aspect Modeling Solution

Through this access, the AOP system is able to display all useful information about variables, functions and optimization methods present in the aspects, enabling the user to set and execute a desired OPF configuration.

Therefore, the final solution has three components corresponding to the two aspects and their compositor: power flow (Figure 3), optimization (Figure 4), and weaver (Figure 5).

During run-time, the user sets the system that will be optimized, and the respective metadata (pertaining to the control and state variables, objective functions, and optimization methods) is set in the weaver component (*ClsGeneric* and *ClsIntelligentOptzr*). The generic variables and functions used in *ClsIntelligentOptzr* store the metadata that has been set in the beginning of the simulation, and reflectively invoke the concrete methods and fields of the power flow component.

The presented solution enables the separation of the power flow problem from the optimization techniques during design time, supporting the logic in Figure 1, but avoiding its explicit coding shown in Lists 1 and 2 altogether. Using this methodology, the logical treatment is no longer part of the solution, the number of objective functions and control variables do not imply complexity growth, and, furthermore, it enables full detachment of the aspects. In other words, changes in one do not impact others.

An informal and simplified way to describe the AOP-OPF solution is shown in Figure 6. The real objects, represented by shadowed boxes, use the unique signature present on their interfaces defining singular aspects. The weaver, a component designed to recognize those specific signatures, composes these aspects together generating the final solution. In this example, the generated assemble is an OPF with the objective function of minimizing overload, using TAP as control action and the BFHS algorithm as optimization strategy.

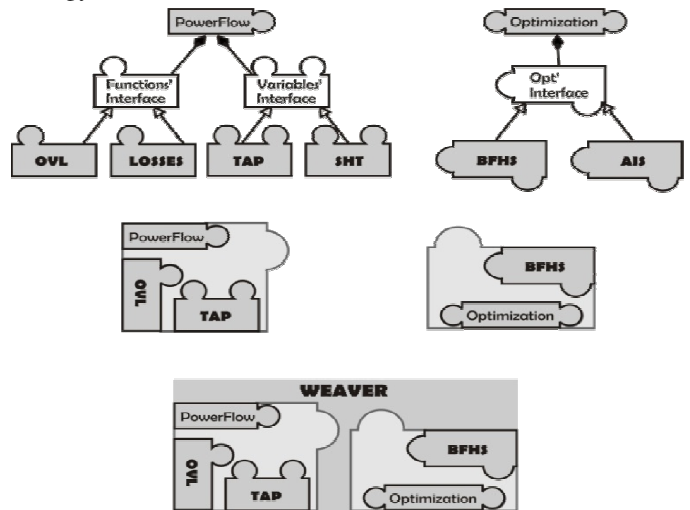


Figure 6 - Info-Graphic of AOP-OPF solution

The AOM methodology broke all external relationships among the components, preserving just the internal ones. Even though certain connections still exist, they provide no impact

among each other in case of changes. This new layout is interfaced-based, which means that if any new variable, objective function or optimization method is added to the system, the weaver is able, through reflection, to recognize and invoke the new functionality without any further modification in its code.

The final remark of an AOP-OPF solution is, regarding the computational time and precision, the same as the one provided by the OOP-OPF software. More details about numerical results can be found in [7].

B. Software Metrics Analysis

This section performs analyses about the final solution given by each presented paradigm. For that purpose, the literature shows several indexes focusing on measuring the software quality in terms of, among others, complexity, maintainability, scalability, modularity, effort, and costs. The most common ones are: Source Lines of Code, Cyclomatic Complexity and Halstead Complexity.

The *SourceLines of Code* (SLOC) is a metric used to measure the amount of code in a software program. It is typically used to estimate the amount of effort that will be required to develop a program, as well as to estimate programming productivity or effort once the software is produced. The *Cyclomatic Complexity* is a metric used to measure the complexity of a program. It directly measures the number of linearly independent paths through a program's source code. The *Halstead Complexity* metric focuses on measuring a program module's complexity directly from source code analyzing the operators and operands in the module. It is a strong indicator of code complexity and it is often used as maintenance metric.

In a practical scenario, taking in consideration the presence of 3 optimization methods, 4 objective functions and 5 variables the final effort of each development paradigm is shown in Table 1.

Table 1- Software Metrics with om = 3, ob = 4; var = 5.

	OOM	AOM
<i>SLOC</i>	105	60
<i>Cyclomatic</i>	11	3
<i>Halstead</i>	251.491	214.516

Comparing these two paradigms it is possible to note that, a more abstract model is able to better representing the problem. It also generates more code lines if the entanglement is low, however, as the necessity of more functionalities grows, the OOP design is able to maintain the software complexity under control.

VI. CONCLUSIONS

In this paper, the use of Aspect Oriented modeling and programming as a new approach to optimal power flow software development has been proposed. Software Metrics has been used to measure the performance of different

paradigms.

The AOP design seems effective for several reasons: (1) it makes intuitive aspectual components; (2) disaggregates the power flow from the optimization techniques during design time as shown in Figure 11; (3) using just two components, it makes the weaver complexity minimum; (4) it uses an intermediary weaver in the optimization concern enabling the implementation of several optimization techniques with no code modification in any of the already-developed components; (5) the final solution is assembled at run-time according to the user's statements, which no longer has the logical treatment shown in Figure 1.

VII. REFERENCES

- [1] Abido, M., "Environmental/Economic power dispatch using multiobjective evolutionary algorithms." IEEE Transactions on Power Systems, Issue 4, Vol. 18, pp. 1529-1537, Nov. 2003.
- [2] Nemhauser, GL, Kan, AHG Rinnooy and Todd, MJ., *Handbooks in Operations Research and Management Science*. Hardbound, Holland : Elsevier, 2005.
- [3] Graville, S., "Optimal reactive dispatch through interior point methods." IEEE Trans. on Power Systems, Issue 1, Vol. 19, pp. 136-147, Feb. 1994.
- [4] Almeida, K.C. and Salgado, R., "Optimal power flow solutions under variable load conditions." IEEE Trans. on Power Systems, Issue 4, Vol. 15, pp. 1204-1211, Nov. 2000.
- [5] Binato, S., Oliveira, G.C. and Araújo, J.L., "A greedy randomized adaptive search procedure for transmission expansion planning." IEEE Transactions on Power Systems, Issue 2, Vol. 16, pp. 247-253, . 2001.
- [6] Pelikan, M., Goldberg, D.E. and Lobo, F.G., "A survey of optimization by building and using probabilistic models." Computational Optimization and Applications, Issue 1, Vol. 21, pp. 5-20, Jan. 2002.
- [7] Honorio, L.M., et al., "Intelligent optimal power flow system development using aspect-oriented modeling." IEEE Transactions on Power Systems, Issue 4, Vol. 22, pp. 1826-1834, Nov. 2007.
- [8] Honorio, L.M., Leite-da-Silva, A.M. and D.A.Barbosa., "A Gradient-Based Artificial Immune System Applied to Optimal Power Flow Problems." Lecture Notes in Computer Science, s.l. : Springer, Vol. 4628, pp. 1-12, Aug. 2007.
- [9] G.L Torres, and V.H. Quintana., "On a nonlinear multiple-centrality-corrections interior-point method for optimal power flow." IEEE Transactions on Power Systems, Issue 2, Vol. 16, pp. 222-228, May. 2001.
- [10] Yoshida, H., et al., "A particle swarm optimization for reactive power and voltage control considering voltage security assessment." IEEE Trans. on Power Systems, Issue 4, Vol. 15, pp. 1232-1239, Nov. 2000.
- [11] Kiczales, G., et al., "Aspect-Oriented Programming." Finland : Springer-Verlag, 1997. European Conference on object-Oriented Programming. pp. 220-242.
- [12] Mendhekar, Anurag, Kiczales, Gregor and Lamping, John., *RG: A case study for Aspect-Oriented Programming Technical Report SPL97-009 P9710044*. Palo Alto, CA : Xerox Palo Alto Research Center, 1997.
- [13] Sobral, J.L., Monteiro, M.P. and Cunha, C.A., "Aspect-oriented support for modular parallel computing." 2006. AOSD06: Proceedings of the 5th international conference of Aspect-oriented software development.
- [14] Kundur, P., *Power System Stability and Control*. New York : McGraw-Hill, 1994.
- [15] Souza, A.C.Z., et al., "Increasing the loadability of power systems through optimal-local-control actions." IEEE Transactions on Power Systems, Issue 1, Vol. 19, pp. 188-194, Feb. 2004.
- [16] Liu, M., Tso, S.K. and Chang, Y., "An Extended Nonlinear Primal-Dual Interior-Point Algorithm for Reactive-Power Optimization of Large-Scale Power Systems with Discrete Control Variables." *IEEE Transactions on Power Systems*. Nov. 2002, Vol. 17, 4, pp. 982-991.
- [17] Rashid, A., Moreira, A. and Araujo, J., "Early aspects: a model for aspect-oriented requirements engineering." Essen, Germany : IEEE Computer Society Press, Los Alamitos, CA, 2002. IEEE Joint International Conference on Requirements Engineering. pp. 199-202.