Development of Power System Analysis Software Using Object Components

K. M. Nor[1], H. Mokhlis[2], H. Suyono[2], M. Abdel-Akher[2], A-. H. A-. Rashid[2], and Taufiq A. Gani[3]

[1]Depart. of Elect. Power Engineering
University of Technology Malaysia
Johor, Malaysia

[2]Depart. of Elect. Engineering
University of Malaya
Kuala Lumpur, Malaysia

[3]Depart. of Computer Science
University Science Malaysia
Penang, Malaysia

***Abstract*—this paper presents experiences of developing a power system analysis software using a combination of Object Oriented Programming (OOP) and Component Based Development (CBD) methodologies. In this development, various power system analyses are developed into software components. These components are integrated with graphical user interface components to build up a power system analysis application. By using both OOP and CBD methodologies, updating or adding new algorithm can be done to any specific component without affecting other components inside the software. The component also can be replaced with any other better component whenever necessary. Hence, the software can be maintained and updated continuously with minimum resources. The performance of the components is described in comparison with the non-component applications in terms of reuse as well as execution time.**

I. Introduction

Power system analysis software has evolved from purely numerical computation programs to sophisticated software application with many usability features to enhance user productivity. Most power system analysis applications now have tools to automate tedious data preparations as well as results post-processing. Examples of such productivity tools are friendly graphical user interfaces, intuitive one-line diagramming facilities to visualize power system, and database management system.

The tools are the products of continuous advancements in computer hardware and software technology. In order to develop a high quality power system analysis application, integration of software development from diverse expertise such as power system engineering, mathematics, software engineering, database, computer-aided design and user interface design is required. The multi-specialization requirement practically means that the application development has to be done with many specific tasks modules.

Development of an application in software modules has been practiced ever since early computer programming in the form of source code library. The main usage was to minimize the number of codes and to manage compilation of large applications in which only the modified parts are recompiled. The compiled libraries (binary parts) later are developed into a dynamically linked libraries (DLL) in the Microsoft windows and shared objects in UNIX. These are late binding libraries to optimize memory resources by optimizing the address space. With DLL, the possibility of composing reusable software modules to develop software application just like integrated circuits being used to build electronic products look to be just around the corner. In reality before real integration of reusable modules into software applications can be done meaningfully, many hurdles need to be overcome.

One of the hurdles is that libraries are language dependent where a library in one language cannot be used in another. Compilers generally do not conform to common standards and thus the libraries need to be compiled with different compilers. This is impractical as most developers cannot afford to support many library versions for the different compilers. No doubt that many programming languages can call other languages but it is only on a one-to-one basis such as a subroutine in FORTRAN language called from C/C++ language. Furthermore since languages such as FORTRAN or C do not provide standard for binary interoperability, most binary calling standard between languages are vendor specific compiler dependent.

Another hurdle in creating reusable software modules is in the form of dependencies. A module is at least dependent on another. Generally many modules have more than one dependency. These dependencies may be via other modules such when a part X is used a function in part Y which is dependent on part Z and thus making X indirectly dependent on Z. In a tightly integrated package a module dependencies extend deep into implementations of other modules and in many directions. In this case the integration of any module upgrades or any module replacement is a complicated task and prone to error.

An important principle about module dependencies is that a module should be concerned only on the outcome and not how the outcome was derived. Hiding information about an object and exposing only necessary things to a client is called encapsulation. Encapsulation is actually an important concept in object oriented programming (OOP) where the exact implementation of functions in an object and the exact format and layout of the object data is only of concern to the object itself. However there is no standard framework that exists in OOP through which software objects created by different vendors can interact with one another within the same memory address space. This has produced OOP objects that cannot interact across software module or software application boundary in a meaningful way. The OOP objects therefore are confined to code level reusability (class reuse), where it becomes language dependent and may even be compiler specific dependent [1].

OOP frameworks are characterized by a massive structure

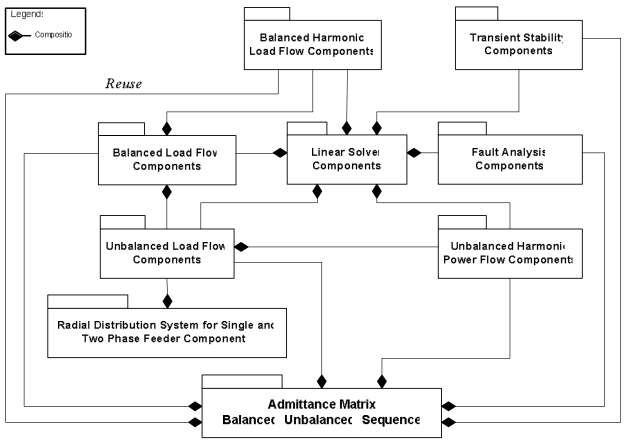

Legend:
Compositio
Balanced Harmonic Load Flow Components
Transient Stability Components
Reuse
Balanced Load Flow Components
Linear Solver Components
Fault Analysis Components
Unbalanced Load Flow Components
Unbalanced Harmonic Power Flow Components
Radial Distribution System for Single and Two Phase Feeder Component
Admittance Matrix
Balanced Unbalanced Sequence

Figure. 1 Power System Analysis Component Relationship: **The Evolving Part**

mostly because OOP implementations such as inheritance have not prevented introduction of cross dependencies among classes. The obvious consequence has been that objects were so tightly coupled that even the simplest application had to link the entire system [2].

Component technology is introduced to address many of the above issues. Szyperski [3] defines a component as: "*A software component is a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third party*."

In order for a component to be deployed independently, it must be well separated from other components and its environment. Since a component communicates with its environment only through interfaces, it must have clearly specified interfaces which are separated from its implementation. Component integration and deployment should be independent of the component development life cycle and there should be no need to recompile or re-link the application when updating with a new component. Examples of component technology are the Microsoft COM and OMG CORBA.

Not much work has been reported in the application of component technology for the development of power system analysis applications [4-8]. This paper discusses our experiences of developing a power system analysis application using object component based software engineering approach. The proposed software is categorized into two parts; they are a stable part and an evolving part. The evolving part is modeled as object software components whereas the stable part is modeled by a pure an OOP class hierarchy. The strategy adopted is to architect the application into a composition of components. Many of the decompositions require power system engineering expertise as they involve power system algorithm reformulation.

This paper will also present the test performance of this application and compare them to applications implemented in a non component based approach in terms of execution time. A measure of component reuse will be presented to show the achievement of the application architecture. Also the paper will discuss implications of the component based development on the application life cycle such as issues of software upgrade and maintenance.

II. Power System Analysis Software Architecture

The software architecture of a program or computing system is the structure of the system which comprises software elements, the externally visible properties of those elements, and the relationships among them. The power system analysis software is developed by composition of components. The component functions can be divided into two main parts which are the control or analysis components and the interface components.

The analysis components result from the decomposition of power system solution algorithms. The decomposed solution algorithms are encapsulated inside independent software components. The solution algorithm components relationship is drawn in Fig. 1 using Unified Modeling Language (UML) notation. These components perform different power system analysis such as balanced power flow, unbalanced power flow, transient stability, fault calculations, and harmonic analysis. The architecture, shown in Fig. 1, can be updated by adding new components for performing additional power system analysis; also any component can be replaced by other component which has efficient or improved algorithm. From the components relationship shown in Fig. 1, particular components are developed based on existing components such as unbalanced power flow components that reuse the balanced power flow components [7-8]. Therefore, many components are developed with minimum resources since other components contribute in their development.

In addition to the analysis components given in Fig. 1, graphical user interface (GUI) components are required for developing user friendly power system analysis application. The GUI consists of components to display graphical drawings such as single line diagram and presentation of results such as tables and charts. This will enable the application user to draw single-line diagram of a power system, and to perform power system analysis such as load flow, unbalanced load flow…etc.

The analysis and interface components are the software evolving part since they are subject to be updated or replaced to fulfill the user requirements. The software stable part is shown in Fig. 2; it does not do any type of calculations as it only represents the physical elements in a power system. The stable part is not modeled as software components but instead is modeled in a pure object oriented paradigm design. The model involves a set of classes related to each other by inheritance and composition to optimize maximum reusability.

III. Object Oriented Power System Model (OOPSM)

There is various object oriented power system models have been presented such as in [9-10]; these models are tightly coupled to the evolving parts. In order to produce high degree of extendable model, which can be extended in the future, the class hierarchy should be modeled at the right granularity.

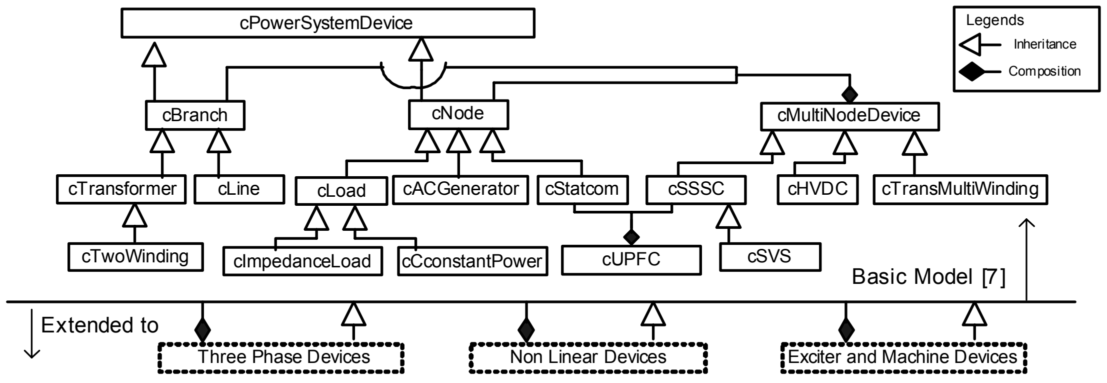


Figure. 2 Object Oriented Power System Model (OOPSM): **The Stable Part**

In the proposed development, the physical electrical network is represented by a standalone class library. The library models the power system devices such as busbars, generators, transformers, transmission lines, loads…etc. The proposed approach is to design a model based on the primitive data of the basic linear circuit elements i.e. node, branch, and source. These classes are developed such that they become common ancestor for the other elements in a power system. This is because any electrical devices can be modeled as an equivalent circuit comprising of these primitive elements.

Using the UML notation, all classes are used to model power system devices and their relationships are drawn in Fig. 2. A base class, **cPowerSystemDevice,** is designed to share all common data and method for its derived classes. Basic properties such as the number of devices, the identity number (ID) and the name of the device are defined in this class. These properties are common to any type of devices in an electrical network. The basic device model [7] shown in Fig. 2 is extended for modeling new devices such as three-phase power system elements, non-linear devices, and exciter systems and turbine governing systems.

IV. Software Object Components

The power system analysis application was developed by integrating various object components. These components are classified into three categories. The first category includes the components that are developed in our research group (In-House Developed Components). They include the components that are required for solving the power system problems. The second category is the components that are developed from legacy systems. The legacy systems are pre-existing systems that was created using other design methods and techniques [11]. These legacy systems can be wrapped and integrated with the other components. The last category includes the Commercial Off-The shelf Components. This category of components is not power system engineer's proficiency area; therefore they are acquired from a third party vendor. In the proposed development, they include components for visualization of the single-line diagram of an electrical network and others for presentation the results of the power system analysis computational engine.

A. Newly Developed Components (In-House Developed Power System Analysis Components)

1) Balanced Load Flow Object Components

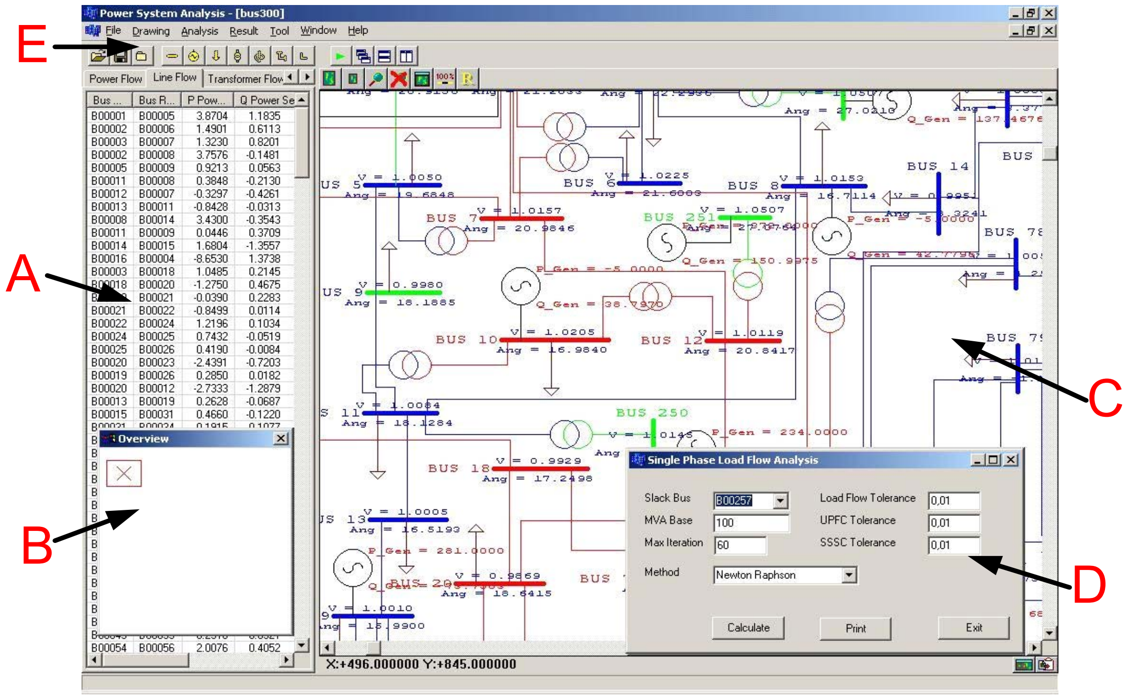
A balanced power system is solved by two components which utilize Newton Raphson method (NR) and Fast Decoupled method (FD) respectively. The components are inherited from a common ancestor base class. The base class contains common data and methods that are required for any balanced power flow algorithm. The base class also contains interface methods for exchanging data between load flow components and other systems. The detailed mathematical models for both NR and FD components can be found in [7].

The NR and FD components have the same interface since it is developed based on the OOPSM and inherited from the base class. The NR and FD components encapsulate their own solution algorithm only. This makes a power flow object versatile. For example, if a load flow object has been declared as an instance of the NR component in an application or a component system, changing the declaration to become an instance of the FD component will not require any change in the code implementation of that application or that component.

The behavior of the components is controlled by two flag interfaces. The first flag enables the user to perform one iteration balanced power flow, and then the solution is hold inside the component till the solution is resumed in next iterations. The second flag checks the convergence inside the component. Therefore the two flags enable the user to perform the iteration process inside or outside the component. This is useful and powerful in many applications or computational systems that reuse the power flow as a part of their solution process.

2) Unbalanced Load Flow Object Components

Unbalanced load flow can be solved using phase coordinates without simplifications. However the advantage of the application of symmetrical components that the size of the problem is reduced in comparison to phase coordinates load flow. The unbalanced load flow in the proposed architecture takes the advantage of the application of component technology and the sequence components since the balanced


Power System Analysis - [bus300]
File Drawing Analysis Result Tool Window Help
Power Flow
Line Flow
Transformer Flow
Overview
E
A
B
C
D

Bus ...	Bus R...	P Pow...	Q Power Se
B00001	B00005	3.8704	1.1835
B00002	B00006	1.4901	0.6113
B00003	B00007	1.3230	0.8201
B00002	B00008	3.7576	-0.1481
B00005	B00009	0.9213	0.0563
B00011	B00008	0.3848	-0.2130
B00012	B00007	-0.3297	-0.4261
B00013	B00011	-0.8428	-0.0313
B00008	B00014	3.4300	-0.3543
B00011	B00009	0.0446	0.3709
B00014	B00015	1.6804	-1.3557
B00016	B00004	-8.6530	1.3738
B00003	B00018	1.0485	0.2145
B00018	B00020	-1.2750	0.4675
[illegible]	B00021	-0.0390	0.2283
B00021	B00022	-0.8499	0.0114
B00022	B00024	1.2196	0.1034
B00024	B00025	0.7432	-0.0519
B00025	B00026	0.4190	-0.0084
B00020	B00023	-2.4391	-0.7203
B00019	B00026	0.2850	0.0182
B00020	B00012	-2.7333	-1.2879
B00013	B00019	0.2628	-0.0687
B00015	B00031	0.4660	-0.1220
B00021	B00024	0.1915	0.1077
B00054	B00056	2.0076	0.4052

Single Phase Load Flow Analysis

Slack Bus	B00257	Load Flow Tolerance	0,01
MVA Base	100	UPFC Tolerance	0,01
Max Iteration	60	SSSC Tolerance	0,01
Method	Newton Raphson		

Calculate Print Exit

X:+496.000000 Y:+845.000000

A: Balanced power flow result window
B: Viewing of single line diagram window
C: Single line diagram design window
D: Calculation form control window
E: Buttons/pull down menus for user program control

Figure. 3 Main Window of the developed power system analysis software: **The 300 Bus IEEE Test Case**

power flow can be reused for solving the positive sequence power flow without any modification [8].

The unbalanced power flow using symmetrical components is decomposed into three-sub-problems that include a positive sequence power flow. The symmetrical components have been used to solve unbalanced systems with accurate results that are tested with other algorithms developed in phase coordinates. The complete mathematical models and the solution algorithm for the unbalanced load flow in symmetrical components are given presented in [8]. The unbalanced power flow has been extended for solving radial systems that contains unbalanced laterals and will be reported in near future.

The interface of the unbalanced power flow is encapsulated inside one base class. The class is developed using object oriented approach; since it handles the balanced load flow as an object. This object behaves as black box, it communicates with its surroundings with its interface.

The behavior of the power flow object is controlled by its flag interfaces. The first flag interface is adjusted such that the positive load flow object performs one-iteration and then the solution is hold inside the object till the solution is resumed. The second flag interface checks the mismatch tolerance is acceptable or not at every iteration. The solution of the positive sequence network is returned outside the power flow object at every iteration by its own interface.

3) *Fault Analysis Component*

A base class for fault calculation components was designed to share common properties for balanced and unbalanced faults. The balanced and unbalanced faults have common positive sequence impedance. However balanced fault only uses positive sequence impedance in determining the voltage value under fault condition. In addition to positive sequence impedance, unbalanced faults reuse both the negative and zero sequence impedances.

In the fault calculation design, positive sequence impedance are declared in the base class and granted to balanced and unbalanced components. The unbalanced fault is broken down to three types, encapsulated inside three different components. They are the single line fault, double line fault and line-to-line fault. Therefore, each type of the unbalanced faults is independent on the other types and changing or updating any component will not affect the other components. The fault calculation component system reuses the same power system model exhibited in Fig. 2.

4) *Transient Stability and Harmonic Analysis*

In addition to the power flow and fault calculations object components described above, other components have been developed for transient stability analysis and balanced harmonic power flow analysis. Now a component for unbalanced harmonic power flow analysis is being investigated and tested.

5) *Data Preparation Components*

This group of components is responsible for preparing the input data for the power system analysis components. The group includes interface components for the following: standard IEEE data format, sequence components data for different devices, machine and exciter data, unbalanced power system data for line, and non linear devices data.

The advantage of separating the data preparation components from the analysis is that, in many cases, the system data will be available in different formats. Therefore, the user of the software application will be able to execute analysis in different input data formats, i.e. without drawing the single line diagram.

B. *Wrapping Legacy Code*

Software components allow reusing pre-existing codes that was created using other designs and technologies. This is done in the proposed development for the linear solver components. The linear solver components employ the SuperLU library [12]. SuperLU library utilizes many latest computation techniques, such as graph reduction technique in matrix factorization and it can handle very unsymmetrical matrices. SuperLU library was developed using C language and contains four different routines. The routines were designed to solve sparse matrices with single and double precision in real and complex formats.

The four SuperLU routines are encapsulated inside four independent components without modifying their original code [7]. Wrapping the codes into components allows it to be integrated with other components in the software. The wrapper was developed using an object oriented interface. This interface includes methods such as SetObject() and GetObject. The object in these methods refers to the standard sparse data storage format such as compressed column storage or compressed row storage. These objects called the SuperLU library functions which are written in C Language.

C. *Commercial Off-the shelf Components*

Many components such as graphics and data base are available commercially. Therefore, acquiring these components from third party vendors accelerates the development time and increase the software capability. These components include DbCAD component for Graphical Database, Cad Engine for single line diagram, and TeeChart Pro for 2D and 3D for charting design. The DbCAD component is a library which has many services for Computer Aided Drawing (CAD) applications. DbCAD manages the graphic entities as single vectors, which are selectable, editable, and displayable in the graphic window with specified properties (color, layer, line type, etc). The TeeChart Pro component is a charting component, which has many features such as creating various chart types including 2D and 3D charts. The TeeChart Pro component is used for plotting the dynamic behavior of synchronous machines in transient stability simulation.

V. APPLICATION SOFTWARE EVALUATION

A. *Application Description*

The window, shown in Fig. 3, displays a prototype of the proposed CBD power system analysis application. The application is developed from the components given in Table 1. The window shows the IEEE 300 bus single line diagram. The application has many facilities that makes it user friendly. The main application window contains features needed by the software application such as main menu, tool bar, image list, etc. The software application uses multiple document interfaces. More than one drawing forms can be opened within the main application window at the same time. In the dialog box that is shown in Fig. 3, which is marked as D, a slack bus and a calculation method needs to be selected. The slack bus is chosen from a combo box containing all of the available busses in the diagram. The calculation methods, i.e. load flow or fault calculation, unbalanced power flow…etc. are chosen from another combo box as well. Other parameters, i.e. as base for MVA, maximum iteration, and tolerances, need to be keyed in. After keying in the parameters, the calculate button is clicked to start any power system analysis calculation.

B. *Component Reuse in the Software Application*

The power system analysis application is developed by integrating our own components, legacy components, and commercial off-the shelf components. The reuse measurement for these components is carried out by counting the total number of components reused and those developed from scratch to perform a specific power system analysis. The reusability of components in the power system analysis application is presented in Table 1. For example, seven components are required to solve unbalanced power flow; only one component is developed from scratch. Therefore the reuse percentage is equal to 86%. The average percentage of reuse for developing the whole power system analysis application is around 73%. This is because there are many components are shared to perform the different power system analyses.

C. *Application Performance*

The performance test is conducted to reveal the difference on execution time between component software and non-component software. In order to do this, a component based software and non-component software with the same algorithm and object-oriented codes were developed. The component-based application is prepared by reusing the power flow and mathematical solver components, which were delivered in the library files. On the other hand, these files are included into the non-component power flow application. In this test, the execution time for solving load flow is taken for both balanced NR and FD power flow methods.

The 118-bus system and 300-bus system were used as the test system. The performance of the proposed CBD application is compared with a non-component application (structural) for both NR and FD methods. The results of the execution time are reported in Table 2 and Table 3. It is clear that the component software application consume a slightly more time compared to the non-component application. The additional time required for the CBD application is less than 10% for large systems.

TABLE 1
REUSABILITY OF COMPONENTS IN THE POWER SYSTEM APPLICATION

Component	Power System Analysis Type					
	BPF	TSA	UPF	HPF	FCA	URD
DB	X	√	√	√	√	√
SLD	X	√	√	√	√	√
AM	X	√	√	√	√	
SLS	√	√	√	√	√	
NR	X	√	√	√	√	√
FD	X	√	√	√	√	√
TRA		X				
MES		X				
UPF			X		√	√
HPF				X		
FC					X	
CHART		√		√		
URD						X
% REUSE	17	78	83	88	88	83
	AVERAGE 72.83%					

Note: X *indicates the components developed from scratch*
√ *indicates components reused*

Where:

DB	Data Base	TRA	Trapezoidal Solver
SLD	Single Line Diagram	MES	Modified Euler Solver
AM	Admittance Matrix	BPF	Balanced Power Flow
SLS	Sparse Linear Solver	UPF	Unbalanced Power Flow
NR	Newton Raphson	HPF	Harmonic Power Flow
FD	Fast Decoupled	FC	Fault Calculation
TSA	Transient Stability	CHART	Charting Components
URD	Unbalanced Radial Distribution		

TABLE 2
NEWTON RAPHSON LOAD FLOW EXECUTION TIME

Data test system	CPU time (in second)	
	Non-component	Component
IEEE-118	0.027344	0.039062
IEEE-300	0.289062	0.312500

TABLE 3
FAST DECOUPLED LOAD FLOW EXECUTION TIME

Data test system	CPU time (in second)	
	Non-component	Component
IEEE-118	<0.001	<0.001
IEEE-300	<0.001	0.01

The component application consumed slightly more time because the component application needs to read library files, which are separated from the application files when they are executed. Therefore, time is consumed in the process of transferring parameters message to and from the functions of the library. However in the non-component application, calling function takes place in the same memory block of the application, which made the CPU time slightly less than the case in the component-based application.

VI. CONCLUSION

The paper has presented the application of CBD and OOP for developing power system analysis application software. The different components have been integrated into power system analysis software. The OOP and component technology produces a high degree of reusability, flexibility, and maintainability of a power system analysis application. A measure of the components reuse for developing the power system analysis application has shown that the component reuse is about 73%. The components also can be replaced at any time with other best component from third party. Therefore the maintenance of the power system analysis application requires only local updating of components rather than global updating of applications. For example, in the future, the proposed software can be added with other power system analyses such as optimization, power quality analysis, or updating the graphical user interface. The performance of the application is also tested, that there is no much difference between component and non-component system in terms of execution time.

REFERENCES

[1] D. de Champeaux, D. Lea and O. Faure, “Object-Oriented System, Assison-Wesley, 1993.

[2] I. Joyner, A C++ Critique, second edition 1992. Available at: http://www.literateprogramming. com/c++critique.pdf

[3] Szyperki C., “Components Software-Beyond Object Oriented Programming”, Reading: Addison-Wesley, 1998

[4] Khalid M. Nor, T. A. Gani, H. Mokhlis, “The Application of Component Based Methodology in Developing Visual Power System Analysis Tool”, Proceeding of the 22nd conference on IEEE PES PICA, Sydney 2001.

[5] Lu, F.Y,; Chen S,” Using component technology in power system simulations”, IEEE Power Engineering Society Winter Meeting, 2002., Vol. 1, No.1, Jan. 2002, pp. 684 - 689

[6] Dzafic, I, Glavic, M., Tesnjak, S., “A Component-Based Power System Model-Driven Architecture”, Letter, IEEE Transactions on Power Systems, Vol. 19, No. 4, Nov. 2004 , pp. 2109 – 2110

[7] Khalid M. Nor, H. Mokhlis, T. A. Gani, “Reusability techniques in load-flow analysis computer program”, IEEE Transactions on Power Systems, Vol. 19, No. 4 , Nov. 2004, pp. 1754 – 1762

[8] M. Abdel-Akher, Khalid M. Nor, A-H. A. Rashid, “Improved three-phase power-flow using sequence components”, approved for publication in IEEE Trans. on Power Systems ID TPWRS-00409-2004.R1

[9] A.F. Neyer, F.F. Wu and K. Imhof, “Object Oriented Programming for Flexible Software: Example of A Load Flow”, IEEE Transaction on Power Systems, Vol. 5, No. 3, August 1990.

[10] Jun Zhu and L.Lubkeman, “Object-Oriented Development of Software Systems for Power System Simulation”, IEEE Trans. (Power App. Sys), Vol. 12, No. 2, May 1997.

[11] Ivar Jacobson, “Object-Oriented Software Engineering” Addison-Wesley Publishing Company. 1992.

[12] J. W Demmel et. al, “A Supernodal Approach to Sparse Partial Pivoting”, SIAM Journal on Matrix Analysis and Applications Vol 20 , No 3. pp.720 -755, 1999. The Library available at: http://crd.lbl.gov/~xiaoye/SuperLU/