A Design Pattern for Power System Steady State Analysis Software

GONG Cheng Ming **LI Lei** **XU Xiu Zhi**
NARI Technology Devlopment Co. Ltd. Suzhou Power Supply Company
gongchengming@sgepri.sgcc.com.cn lilei3@sgepri.sgcc.com.cn nickxxz@163.com

Abstract
This paper addresses a design pattern for power system steady-state analysis software. With the addressed design pattern, objects of the network model, analysis problems and their solvers are identified and isolated. By using this design pattern, efficiency of the development and flexibility of the software can be improved greatly.

Keywords
design pattern, power system analysis software

1. INTRODUCTION

The power system steady-state analysis software, from the basic power flow program to very complicated optimization tool, plays important role for today's power system planning and operation. Many efforts have been put on the algorithms and their implementation. Typical problems have been thoroughly discussed e.g. in [Bergen et al, 2000], [Abur et al, 2004] and [Zhu, 2009]. Some design methodologies, including modern Object-Oriented methodologies [Neyer el al, 1990] [Zhou, 1996] [Soman et al, 2002], have also been introduced for the development of the software. But existing work only focused on one or several problems, without any pattern being summarized for the common design of the steady-state analysis software. Reference [Qiu et al, 2012] introduced the application of design patterns in a set of commercial software for simplifying the design of user interface, data source, etc., but it less relates to the core analysis aspect.
Patterns originated as an architectural concept [Alexander et al 1977]. Kent Beck and Ward Cunningham began applying patterns to programming and presented their results at the OOPSLA conference [Beck et al, 1989]. After the book [Gamma et al, 1994] was published, design patterns gained popularity in computer science.
In software engineering, a design pattern is a general reusable solution to a commonly occurring problem within a given context in software design. A design pattern is not a finished design that can be transformed directly into source or machine code. It is a description or template for how to solve a problem that can be used in many different situations. Object-oriented design patterns typically show relationships and interactions between classes or objects, without specifying the final application classes or objects that are involved.
This paper introduces a design pattern for the power system steady-state analysis software. This design pattern isolates the Model, the Problem and the Solver. The internal components of Model and Solver are well decomposed and the relationships of the internal components and external dependencies are analyzed. With this design pattern, it becomes easier to compose very complicated analysis software.

2. OVERALL ANALYSIS

2.1 Isolation of Model

Though there are different problems for power system steady-state analysis, these problems are based on the common power system stead-state model. It's natural to isolate the Model with the responsibilities
1) to represent the static properties of the power grid,
2) to represent the dynamic states of the power grid and
3) to calculate how the static properties react to different dynamic states.

Different problems and their solvers will share the common Model.

2.2 Isolation of Problem

Different power system steady-state analysis problems are distinguished by the different given conditions. For example,
1) conditions for power flow problem are the setting values for each bus,
2) conditions for power system state estimation are a set of measurements
3) conditions for optimal power flow problem are the objective and constraints.

It's helpful and convenient to isolate the Problem with the responsibility to precisely describe what the problem is.

2.3 Isolation of Solver

The purpose of any steady-state analysis function is to solve the state variables for given Model and Problem. The core part of the analysis software is surely the Solver and its dedicated responsibility is to efficiently find the correct solutions.

2.4 Relationships of Model, Problem and Solver

Based on the above analysis, relationship of the Model, Problem and Solver can be organized as figure 1. It means that he Model and the Problem are independent, and the Solver depends on both the Problem and the Model.

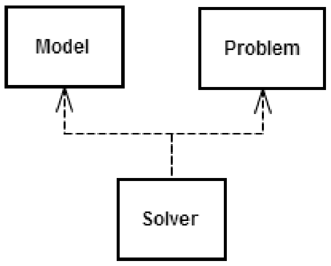

Fig.1 Relationships of Model, Problem and Solver

3. DECOMPOSITION OF THE MODEL

According to the responsibilities of the Model analysed in the previous section, a Model can be decomposed as three components.

1) The Static Property, which represents the topological structure and physical properties of the Model. This component is mainly composed of a set of variables which describe
   a) number and IDs of buses in a grid,
   b) number and IDs of the branches and injections in a grid,
   c) connections of the branches and injections,
   d) parameters of buses, branches and injects.
2) The Dynamic State, which can determine a unique state of the grid. The complex bus voltages are commonly selected as state variables for the steady-state analysis.
3) The Operator, which can give information on the Model and on how the grid reacts to given dynamic states. This part is mainly composed of a set of methods which are responsible to calculate
   a) the bus admittance matrix (Y matrix),
   b) the bus impedance matrix (Z matrix),
   c) the power flows of branches and bus injections,
   d) the first-order derivatives of power flows to state variables,
   e) the second-order derivatives of power flows to state variables,
   f) etc.

Among the above three components, the Static Property and the Dynamic State are independent. The Operator depends on the Static Property and the Dynamic State. Relationships of the components are shown in figure 2.

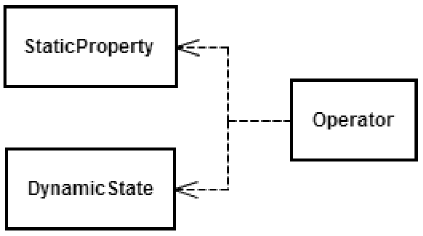

Fig. 2 Decomposition of Model and relationships of the components

4. DECOMPOSITION OF THE SOLVER

A Solver is responsible to get the solution for given Model and Problem. It's normal that multiple solvers are available and often necessary for one Problem. For example, Newton solver and fast-decoupled solver are available for power flow problem. Based on the principle of decomposing the interface and implementation, it's proper to isolate a Solver Interface from the Solver as the base of different Concrete Solver.

A Solver is normally configurable with a set of parameters, such as convergence threshold, iteration limitation etc. The Solver Parameter has its own explicit responsibility so it can also be isolated.

The relationships of Concrete Solver, Solver Interface and Solver Parameter are shown in figure 3. It's clear that the Solver Interface depends on the Solver Parameter, and the Concrete Solver inherits from the Solver Interface.

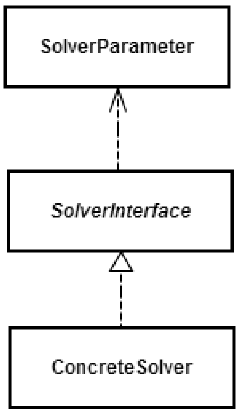

Fig. 3 Decomposition of Solver and relationships of the components

After the isolation of Solver Interface and Concrete Solver, the dependencies of the Solver on the Model will be allocated to different components. Firstly, Solver Interface only depends on Static Property and Dynamic State in Model. Secondly, Concrete Solver depends on Operator in the Model. Figure 4 shows the detailed relationships of the components in Model and Solver.

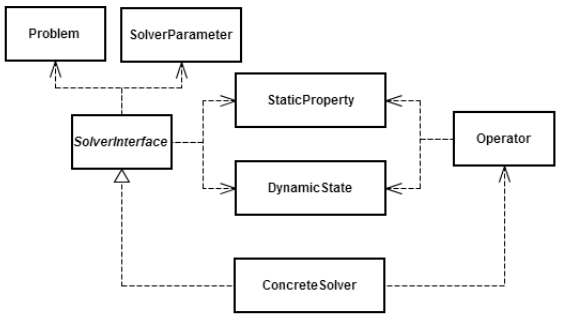

Fig. 4 Detailed relationships of components in Model and Solver

5. EXTERNAL DEPENDENCIES

The Operator and different Concrete Solver are possible to share common data structures and lower-level solvers. The responsibilities of these data structures and lower-level solvers are to represent specialized mathematical concepts and to solve standardized mathematical problems. Sparse matrix and direct linear solvers are examples of this kind of data structures and lower-level solvers. These data structures and lower-level solvers are often available from third-parties, so they can be regarded as External Dependency. Figure 5 shows the relationships of Operator, Concrete Solver and External Dependency.

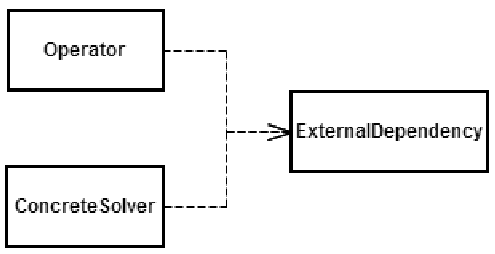

Fig. 5 Relationships of Operator, Concrete Solver and External Dependency

Many of the power system steady-state analysis problems are finally converted to problem of solving sparse linear equations. Sparse linear solvers, normally direct solvers [Pandit et al, 2001], will be used by most of the Concrete Solvers. Special techniques, such as reducing non-zero fill-ins during factorization, are required to solve the sparse linear problems. It’s reasonable to utilize the mature solvers from the mathematics area. Existing solvers of this kind include SuperLU [Li, 2005], etc. Detailed information on available direct sparse solvers can be found in a survey [Li, 2011] and the book [Davis, 2006]. The Sparse Solver is normally designed based on abstract sparse matrix [Pandit, 2001] [Davis, 2006], and the abstracted Sparse Matrix class can be shared by the Operator for the basic sparse matrix representation and operation. When the Sparse Matrix and the Sparse Solver are isolated, the relationships in figure 5 are specialized as figure 6.

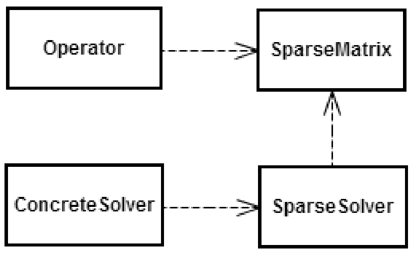

Fig. 6 Dependency of sparse matrix and sparse solver

Some other common data structures and solvers may also be shared by the Operator or different Concrete Solvers. They can also be isolated as external dependencies like the Sparse Matrix and the Sparse Solver.

6. CONCLUSION

By merging figure 6 and figure 4, the whole Model / Problem / Solver design pattern is formed. All the relationships of isolated components in this design pattern is shown in figure 7.

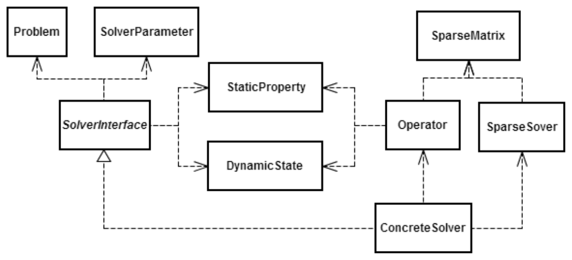

Fig. 7 Detailed structure of the Model / Problem / Solver design pattern

When the Model / Problem / Solver design pattern is applied to different steady-state analysis problems and their different solvers, the isolation of the components becomes much clearer. Figure 8 shows an example in which the Model / Problem / Solver design pattern is applied to two problems (AProblem and BProblem). For each problem, there are two concrete solvers (Sover1 and Solver2). It can be seen that common components are isolated for well sharing.

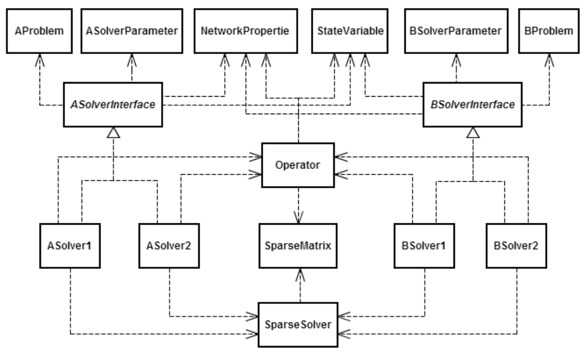

Fig. 8 Illustration of multiple problems and solvers

Also this Model / Problem / Solver design pattern has been applied for real product design in the author’s companies. Higher efficiency and better quality has been achieved by applying this design model.

There is still some further work on the addressed Model / Problem / Solver design pattern. The main work is to extend the design pattern for other non-steady-state problems such as short circuit analysis, dynamic analysis etc.

References

- Ali Abur, Antonio G ́omez Exp ́osito, 2007, Power System State Estimation: Theory and Implementation. Marcel Dekker, New York, 2004.
- Christopher Alexander, Sara Ishikawa, Murray Silverstein, 1977, A Pattern Language: Towns, Buildings, Construction. Oxford University Press, Berkeley, 1977.
- Kent Beck, Ward Cunningham, 1989, A Laboratory for Teaching Object Oriented Thinking. OOPSLA (Object-oriented Programming, Systems, Languages, and Applications) '89 Cnonference Proceedings, ACM SIGPLAN Notices, Vol. 24, No. 10, pp. 1-6, New Orleans, 1989.
- Arthur R. Bergen, Vijay Vittal, 2000, Power System Analysis, 2nd Edition. Prentice Hall, New Jersey, 2000.
- Timothy A. Davis, 2006, Direct Methods for Sparse Linear Systems. SIAM, Philadelphia, 2006
- Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides, 1994, Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, Boston, 1994.
- Xiaoye Li, 2005, An Overview of SuperLU: Algorithms, Implementation, and User Interface. ACM Transactions on Mathematical Software, Vol. 31, No. 3, pp. 302-325, 2005
- Xiaoye Li, 2011, Direct Solvers for Sparse Matrices. August 2011. [online]. Available: http://crdlegacy.lbl.gov/xiaoye/SuperLU/SparseDirectSurvey.pdf.
- F. Neyer, F. F. Wu, and K. Imhof, 1990, Object-Oriented Programming for Flexible Software: Example of A Load Flow. IEEE Transactions on Power Systems, Vol. 5, No. 3, pp. 689-695, 1990.
- Shubha Pandit, S. A. Soman, S. A. Khaparde, 2001, Design of Generic Direct Sparse Linear System Solver in C++ for Power System Analysis. IEEE Transactions on Power Systems, Vol. 16, No. 4, pp. 647-652, 2001.
- Weijia Qiu, Weimei Zou, Yongfeng Sun, 2012, Design Patterns Applied in Power System Analysis Software Package. 2012 International Conference on Industrial Control and Electronics Engineering, Xi'an, 2012.
- S.A. Soman, S.A. Khaparde, Shubha Pandit, 2002, Computational Methods for Large Sparse Power Systems: An Object Oriented Approach. Kluwer Academic Publishers, Norwell, 2002.
- Erzhuan Zhou, 1996, Object Oriented Programming, C++ and Power System Simulation. IEEE Transactions on Power Systems, Vol. 11, No. 1, pp. 206-215, 1996.
- Jizhong Zhu, 2009, Optimization of Power System Operation. Wiley-IEEE Press, Hoboken, 2009.