A Component-Based Power System Model-Driven Architecture

Izudin Dzafic, Mevludin Glavic, *Member, IEEE*, and Sejid Tesnjak

***Abstract*—This letter describes an approach of applying the model-driven development in power systems. A component-based model-driven architecture, that gives full flexibility of the automation in source code generation, is introduced. A design pattern to code generation is described.**

***Index Terms*—Code generation, component based approach, model-driven development, symbolic computation.**

I. Introduction

The commercial use of electricity began in late 1970s of the 19th century [1]. Since then, power systems around the world undergone technical changes that mostly can be attributed to the advances achieved in applied mathematics, software engineering, computer science, telecommunications, etc. More than 100 years of experience in planning, operating, and controlling power systems resulted in tremendous expert knowledge in the field. Unfortunately, there is often unjustified gap between advances in the mentioned fields and their applications in power systems. This letter addresses the issue within the context of recent advances in software engineering and introduces a component-based model-driven architecture (MDA) for power system steady-state analysis and optimization software applications development. The architecture is built in the spirit of the object management group's (OMG) recently announced MDA initiative, which offers conceptual framework for defining a set of standards in support of model-driven development (MDD) [2], [3]. The main idea of this letter is to connect expert knowledge in power systems with the ongoing software development initiatives and make this later available to power system experts that usually lack software development knowledge.

II. MDD Essentials

The MDD essentials include: automation, standardization, source code efficiency, and scalability.

Automation is by far the most effective technological means for boosting productivity and reliability. The MDD potentials for automation include: automatically verifying models on a computer (e.g., by executing them) and automatically generating complete programs from models (as opposed to just code skeletons and fragments).

Standardization provides a significant encouragement for further progress because it codifies best practices, enables and encourages reuse, and facilitates internal working between complementary tools.

Efficiency. One of the very important issues about MDD is how the automatically generated code's efficiency compares with the handmade code. Code efficiency can be decomposed into two separate areas: performance and memory utilization.

Scalability. MDD is intended for large-scale applications. An important metric of concern here is compilation time that can be divided into two separate parts: full system generation time and turnaround time for small incremental changes. The later is much more important because of its greater impact on productivity. Namely, small changes are far more frequent during development than full system recompiles.

Manuscript received March 18, 2004. Paper no. PESL-00147-2003.

I. Dzafic is with Siemens AG, Power Transmission and Distribution, Nuremberg, Germany (e-mail: izudin@dzafic@siemens.com).

M. Glavic is with the Electrical Engineering and Computer Science Department, the University of Liège, Liège, Belgium.

S. Tesnjak is with the Electrical Engineering Department, University of Zagreb, Zagreb, Croatia.

Digital Object Identifier 10.1109/TPWRS.2004.836178

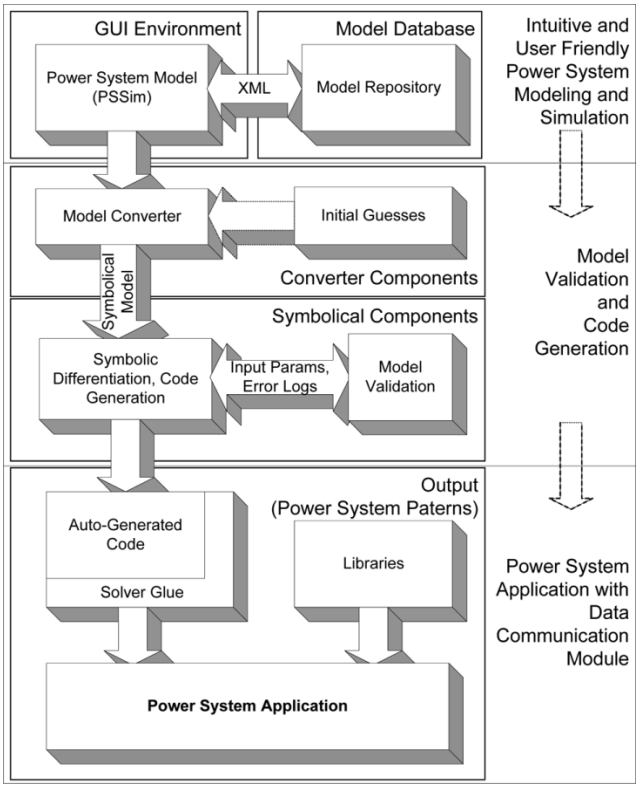

Fig. 1. The main components of the power system MDA.

III. A Power System MDA

The architecture, illustrated in Fig. 1, consists of a GUI part for power system modeling [4] (models are kept in model repository), a set of model converter components, and a set of components [5] that utilize symbolic models, test them, and auto-generate the code.

All user efforts are focused to the system modeling through the GUI part and model validation through extensive experimentation with a symbolic model (the symbolic component has embedded user-friendly editor that allows description of a formal model in natural form). For each specific problem of interest there must be at least one model converter implemented. Converters encapsulate initial guesses for variables and put initial information together with the model. Converters transform graphical model into symbolic one that is further converted to the source code by specially designed set of components.

As is shown in Fig. 2, the code generator is implemented as object-oriented parser with embedded symbolic differentiation capabilities. The Reverse Polish Notation Parser class (*RPNParser*) and the *EvaluatorSparseMatrix* class have *SerializeCode* method that is used to generate code in a pre-selected location. Class *NRSolver* is used to verify models. It actually interprets commands created by the *RPNParser*. Auto-generated source code consists of two parts [6]: application framework code and code needed for Jacobian and model evaluation inside the Newton-Raphson (N-R) solver.

The N-R code is implemented as a C++ class. This class has two virtual member functions *EvaluateJacobian* and *EvaluteModel*. Auto-

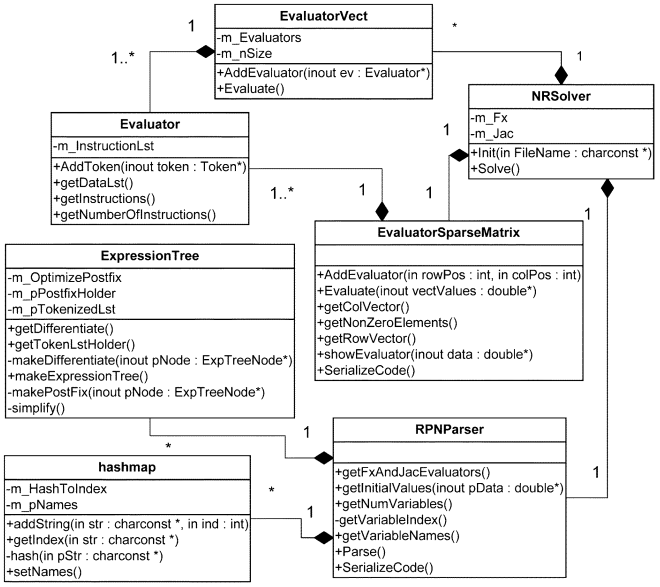

Fig. 2. Class diagram of symbolic code generator.

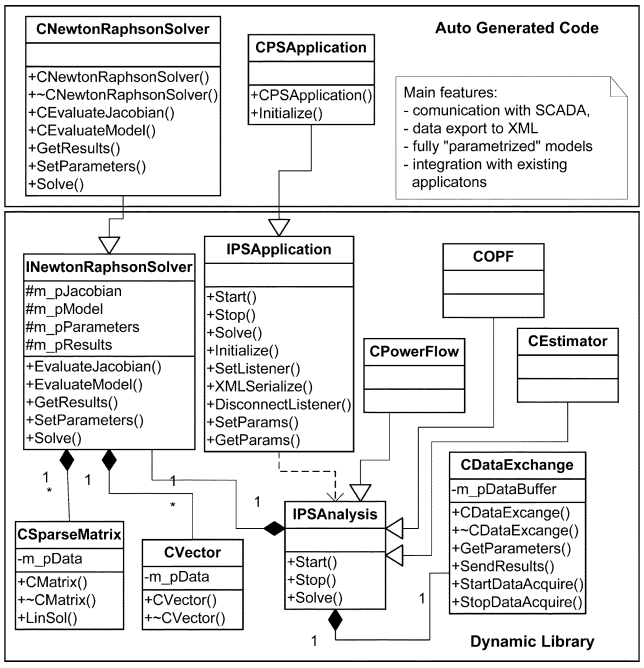

Fig. 3. Simplified class diagram of the auto-generated code.

generated code needs to generate constructors for the *CNewtonRaphsonSolver* and *CPSApplication* classes and source code for *EvaluateJacobian* and *EvaluteModel*. *CDataExchange* contains methods used in communication with underlying SCADA. Taking advantage of object-oriented programming, the main part of the power system application code can be compiled into a library. Thus, the generated source code is more compact and easier to maintain. Fig. 3 shows simplified class diagram of the object-oriented structure of auto-generated code. It is worth noticing that the auto-generated source code is implemented with loops unrolling technique and with reduced reference (address) pointing. As a result of this implementation, created source code is always slightly faster than the same carefully handcrafted code. Also, as Fig. 3 shows, design pattern of the auto-generated code enables very low time overhead in case of model changes. In this case only few auto-generated files should be recompiled. Since all class methods are referenced through the underlying abstract interfaces, there is no need to recompile other parts of the code. Thus, the *efficiency* and *scalability* are achieved. Memory efficiency is achieved through the usage of a sparse matrix library.

TABLE I
TIMING RESULTS OBTAINED ON AMD XP 2500+, 512 MB, WINDOWS XP

Case	Average Solution Time		Difference
	MDD	UWPFLOW	
IEEE118	7 ms	8 ms	14.3 %
IEEE300	43 ms	52 ms	20.9 %

IV. RESULTS

Execution time of auto-generated code has been compared with that of handcrafted C/C++ code for power flow analysis—UWPFLOW [7]. The results obtained using two standard test systems are given in Table I. Execution times do not include time needed for routines initialization.

Auto-generated code, for particular cases considered, is approximately 15%–20% faster than similar handcrafted code.

V. CONCLUSIONS AND FUTURE WORK

A component-based MDA, for power system software applications development, is presented. The work is underway to standardize some segments of the architecture according to the OMG's MDA specification. Components of the test architecture can be obtained upon request to the authors, without any charge.

REFERENCES

[1] P. Kundur, *Power System Stability and Control*. New York: McGraw-Hill, 1994.
[2] MDA Resources. Object Management Group. [Online]. Available: http://www.omg.org/mda/index.htm
[3] B. Selic, "The pragmatics of model-driven development," *IEEE Software*, vol. 20, no. 5, pp. 19–25, 2003.
[4] I. Dzafic and M. Glavic. (2003) PSSim: Windows-Based Application for Power System Analysis and Simulation,Ver. 1.5. [Online]. Available: http://www.simtel.net/pub/pd/74 914.html
[5] I. Dzafic, F. L. Alvarado, M. Glavic, and S. Tesnjak, "Component based approach to power system applications development," in *Proc. 14th Power System Computation Conf.*, Sevilla, Spain, 2002, Paper 33-1.
[6] I. Dzafic, S. Tesnjak, and M. Glavic, "Automatic object-oriented code generation to power system on-line optimization and analysis," in *21st Modeling, Identification, and Control*, Innsbruck, Austria, 2002, pp. 620–624.
[7] C. A. Canizares, Z. T. Faur, and F. L. Alvarado. UWPFLOW. [Online]. Available: http://iliniza/uwaterlo.ca/~claudio/software/pflow