

Generic Modeling and Analysis Environment Design for Shipboard Power System

Jian Shi*, Ranjit Amgai*, Sherif Abdelwahed*, Abhishek Dubey[†], Josh Humphreys*, Mohamed Alattar*, and Rui Jia*

*Department of Electrical and Computer Engineering
Mississippi State University, Starkville, MS, USA

[†]Institute for Software Integrated Systems
Vanderbilt University, Nashville, TN, USA

Abstract—This paper proposes a new modeling and simulating environment based on the principles of Model Integrated Computing (MIC) that facilitates the design and analysis of shipboard power system and other similar distributed and comprehensive power systems. The conventional simulation platforms such as Matlab[®], Simulink[®], PSSE[®] and VTB[®] among many require the designers to have explicit domain specific knowledge of the syntactic, semantic and presentation information of the desired domain within the tools. This, however, severely slows down the design and analysis process, and cross domain and cross-tool operations remain challenging. Our approach focuses on the design of a modeling environment that provides generic support for different application domains (related to shipboard power systems), but yet explicit enough to capture the properties of a domain in terms of modeling concepts, composition principles and operation constraints. The proposed Shipboard Power System Modeling Environment design concept is demonstrated and verified through an open source toolkit Generic Modeling Environment (GME) with two case studies.

I. INTRODUCTION

Shipboard power system (SPS) includes complex interrelated dynamics and controls. The main purpose of the system design is to supply constant power to essential components including propulsion, weaponry, navigation and communication systems. The stability and reliability of the system under different scenarios, especially in the events of battle, is very critical to the mission of SPS design. Careful modeling and simulation to analyze such contingent scenarios is critical for evaluating the survivability of the SPS. In addition, complete ship building demands the inclusion of multi-disciplinary engineering fields including mechanical, electrical, communication and thermal dynamics.

Several conventional simulation platforms exists including Matlab, Matlab Simulink, PSSE, PSCAD, and VTB among many. These tools provide an ability to model and analyze specific portions of a complex and multi-disciplinary systems like shipboard power systems. However, there are some problems.

- The designer needs to have very explicit knowledge of the tool in order to develop the application fitting into the desired specifications and avoid syntactic mistakes, violating hierarchical component dependencies or other constraint violations.

- Although similar design concepts can still be used across different tools, it is still relatively time-consuming and expensive to transplant the application model due to the incompatibilities among different tools.
- Limited to the variety of tool-specific syntactic rules and constraints, it is hard to be expanded for development of future technologies system updates

Model-integrated concepts has been recently addressed [1] in power systems domain to integrate modeling tools from diverse domains. In this paper, a modeling approach based on Model Integrated Computing (MIC) [2],[3],[4] is proposed to support the conventional simulation environment. objective of the approach is to design a flexible and extensible model-integrated graphical framework that facilitates rapid evaluation of SPS under different testing scenarios across different domains and platforms.

The paper is organized as follows: Section II provides a brief review of the software infrastructure of the proposed design. In Section III, two case studies of the proposed approach are elaborated with the analysis of the simulation results. Finally, in Section IV, future work is discussed and the conclusions are drawn.

II. MODEL INTEGRATED COMPUTING AND GENERIC MODELING ENVIRONMENT

A. Model Integrated Computing (MIC)

The application development and schematic design for power systems has always been a challenge. Stringent time limit, high development and evaluation cost, complex interrelated components and availability are the factors that power engineers cannot solely rely on the physical test bed for system development[5]. Well stated models explicitly capture the structure, characteristics of the target system and the operation environment. On the other hand, models also provide a flexible and efficient approach to perform system design, performance analysis, verification and validation, and artifact synthesis capabilities. MIC approach adopts the model-based paradigm and provides a high-level, abstracted syntax and semantics representation for specifying and reasoning about different design aspects and system properties[2][3]. A

modeling language written using the grammar of this high level language can be tailored and be made domain specific[6] and can be made applicable to various control algorithms[7]. Alternatively, it also provides a mechanism for specifying an abstract integration language that can allow the consolidation of common semantics of various commercial analysis tools available for the domain. Then, the challenge remains: how to accelerate the modeling and analysis process? How to precisely capture the critical system specifications? How to extend the design to a variety of applications within different domains?

The concept of MIC is to facilitate the environment designers, by enabling the definition of the syntax and semantic specifications in a way that yields a better overall experience during building and simulating practice of complex applications[6]. For the implementation of MIC, a two level development process is employed[3][5][8]. Software or system engineers operate on the meta-level for specifying and configuring a specific domain; while domain engineers work on the application level to create the application model and analyze the performance. Figure 1 demonstrates the structure of a typical MIC design work-flow.

Meta-level is a domain-independent abstraction that defines a domain specific environment in terms of modeling concepts, component relations, model-composition principles and constraints. In other words, meta-level is the specification of modeling paradigms of system configurations. It contains the base knowledge of rules and constraints of a specific domain and the corresponding representations.

Application level provides an environment for application model customizations. The objective is to let the environment designers build the model, synthesize executable applications, and analyze the simulation results on the application level. Principles of application operations are based on the semantic representations and paradigms defined in the meta-level. With changes and updates applied to the system, designers can easily modify the model and re-synthesize application files.

A model interpreter is used to convert the knowledge captured in the application models to other useful artifacts[9]. For example, it can be used to generate executable code and configurations files. Upon user's request, attributes and relationships of system components will be acknowledged and synchronized to an executor, which is normally provided by the specific domain. The interpreter will then invoke the executor and generate output in the form of data files, graphs, etc[4].

B. Generic Modeling Environment (GME)

Developed by the Institute of Software Integrated System at Vanderbilt University, Generic Modeling Environment (GME) is a configurable tool that provides a generic solution for model design and application development for different domain-specific modeling environments[10](figure 2).

A set of generic concepts have been embedded in GME to facilitate the creation of sophisticated systems. Typical modeling concepts of GME include: aspects, attributes, hierarchy,

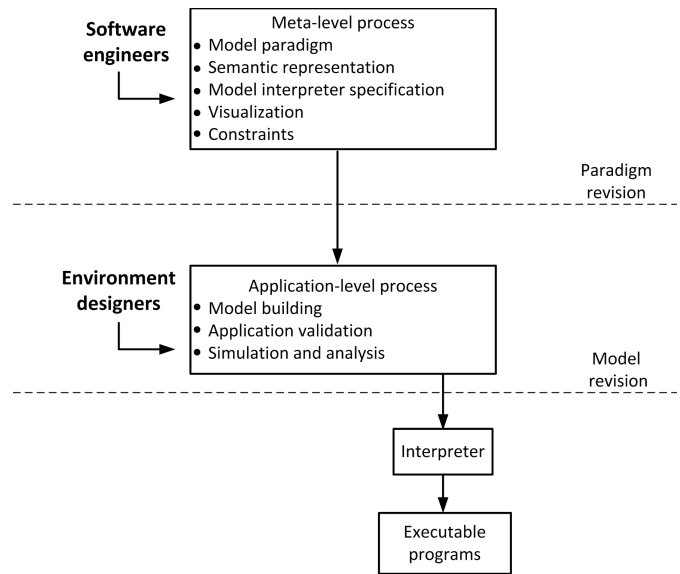


Figure 1. MIC Structure

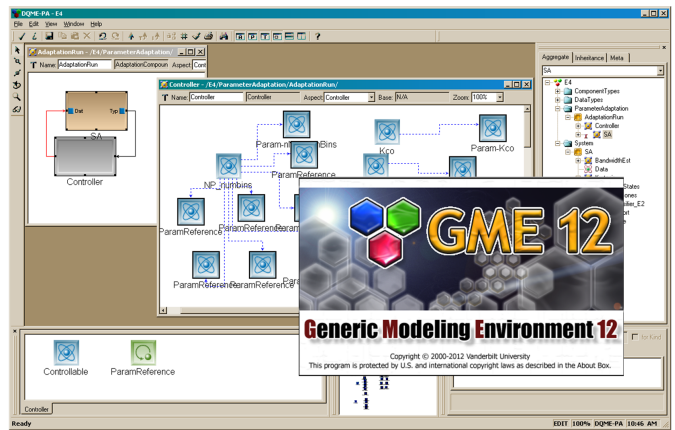


Figure 2. Generic Modeling Environment (GME)

set, reference, and constraints[11][4]. Within a GME project, model, atom, reference, connection and set are classified as first-class objects (FCOs).

- *Atoms*: the basic, elementary object, which cannot contain any objects inside
- *Models*: the comprehensive object that can contain other objects and inner structures
- *Hierarchy*: the containment relationships between objects. Every object must have one parent and the parent must be a model
- *Aspects*: the control unit of visibility that determines which part of the model is visible or hidden
- *Connection*: expresses the relationship between objects within the same model. In order to make a connection, the connected objects must be visible to each other, i.e. in the same aspect
- *Reference*: expresses relationships between objects in different system levels or different systems

- *Set*: relationships among a group of objects under the same folder with the same aspect
- *Attributes*: in order to capture information that has no graphical representations, FCOs are affiliated with attributes. The common available attributes are test, integer, double, boolean and enumerated
- *Constraints*: rules made specifically for model-composition and attribute specification.

For a given domain, the paradigm is created as a composition of the modeling concepts as they determine the capacity of a specific application.

Apart from the effectiveness and expandability introduced with MIC, GME also offers a user-friendly graphical design interface. Developing a system model, especially a system with sophisticated components and hierarchical composition like ship-board power system, is an error-prone process. However, instead of the typical, textual representation, GME has offered designers a better option of a more expressive and readable system representation that is visible in the GUI. In this way, the tedious code-based design becomes an easier, more straightforward and more visualized process.[12]

In summary, GME is a comprehensive toolkit that integrates the meta-model editor, meta-model interpreter, application model editor, domain specific codes generator and simulation execution environment.

III. CASE STUDY

This research work focuses on evaluating and analyzing the static and dynamic performances of a simplified SPS. In order to achieve this goal, toolboxes that are specifically designed for power system analysis and control are carefully studied and chosen. Typical toolboxes like Matpower, Power System Toolbox (PST), Power Analysis Toolbox (PAT), Voltage Stability and Analysis Toolbox (VSAT) and Power System Analysis Toolbox (PSAT) have been widely utilized to perform different power system analysis on a variety of simulation platforms. Common functions supported include the calculation of power flow, continuation power flow, optimal power flow, and small signal stability analysis and time domain simulation. In the following case studies, GME will be integrated with Matpower and PSAT toolbox individually to create the design environment.

A. Case study I: Integration with MatPower

For the first case study, Matpower[13], which is primarily used to solve power flow and optimal power flow problems, is integrated with the design environment. The basic procedure for running a simulation in Matpower follows: i), Preparing of the input data matrices that defines all the relevant system parameters; ii), Invoking the main function to perform power flow or optimal power flow calculation, and iii), Displaying the results and saving simulation data in predefined structures and directories.

Modeling of Matpower is based on the standard steady-state power flow analysis models[14]. Equations describing system components and connections are represented in the

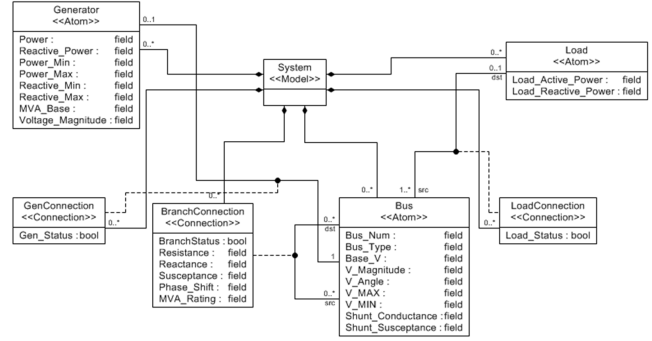


Figure 3. Meta_Model Design

form of matrices in the Matlab structure. The common fields of the Matlab structure consist of bus, branch, generator and generator costs for optimal power flow analysis. Among them, branches include all the transmission lines, transformers and phase shifters information, and modeled as a standard pi transmission line model with series impedance and charging capacitance. Generators are modeled as complex power injection at a specific bus with an active part and a reactive part, and loads are modeled constant consumption of active and reactive power at a bus. After the specifications of the Matpower *struct* or *case* file, commands like *runpf* and *runopf* are invoked to execute the analysis. The solver of Matpower is relying on the Matlab extension (MEX) files.

1) *Step.1 Create the meta-model*: Based on the Matpower format requirement, the main components in the system can be summarized into Generators, Buses and Loads (see figure 3). Generator blocks contain most of the attributes for the *gen* matrices in the MatPower data file, the Bus blocks contain most of the attributes for the *bus* matrices, and the Load blocks contain the active and reactive power data for the *bus* matrices. There are also three types of connections in the system, branch connections, generator connections, and load connections. The Branch Connection contains data for the *branch* matrices including the destination and source bus of the connection. The Generator Connection includes data for which bus a generator is connected to, along with the generator status for the *gen* matrices. The Load Connection includes the information about the bus number a certain load is connected to and a status attribute indicating the connection status. Once all the components and attributes are settled in the meta model, the next step is to create the application model and code the interpreter.

2) *Step.2 Create the application model*: The application model is created to mimic the medium voltage AC baseline models of shipboard power system developed by the Electric Ship Research and Development (ESRDC)[15]. The fundamental topology includes four turbo-generators connected to a ring-bus which supplies two propulsion motors and four zonal loads. Other components like energy storage system or high-level pulsed load are not included for the simplicity. The main focus on the application model design is to evaluate the

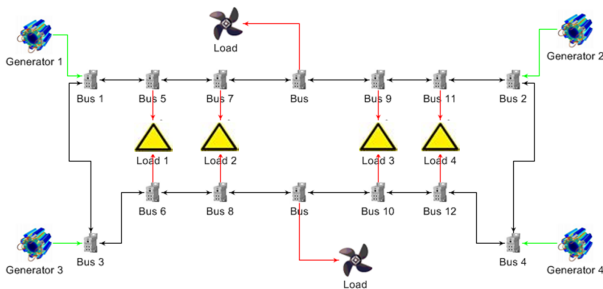


Figure 4. Application_Model Design

static state optimal power flow within the system, thus the control units and dynamic components within the system are also removed. The demonstration of application model is as shown in figure 4.

3) *Step.3 Interpreter design:* To develop the interpreter, the very first procedure is to collect data from the design interface, and save them in the correct locations. The next step is to ensure that all the constraints are satisfied. The final process in the interpreter is to create the needed m-file and running the Matlab engine to use the files to get the results back. The interpreter is programmed in Visual C++ environment. It uses a dynamically allocated two-dimensional array to save the data. Sample code below shows a section of codes that collects *Bus* data entities from the application model. *getAttribute()* is one of the main functions used to collect attributes of particular names from the system.

```
//Bus_Data_Collection
...
if ((* it) -> getObjectMeta().name() == "Bus"){
bus_array [0][ buses]=
(* it) -> getAttribute("Bus_Num") -> getIntegerValue();
bus_array [1][ buses]=
(* it) -> getAttribute("Bus_Type") -> getIntegerValue();
bus_array [4][ buses]=
(* it) -> getAttribute("Shunt_Conductance") -> getRealValue();
...
}
```

4) *Step.4 Define constraints within the interpreter:* The constraints as shown in the sample script below are described and implemented from a list of constraints decided by the users. Many of constraints are checked using a set of flags and counters during the actual gathering of data. Other constraints, such as the unique bus number, are checked using multiple *for* loops and checking the values from the data collected earlier. If any of the constraints is true, indicating a constraints violation; an error flag is set true and the flag will prevent the file generation as well as the Matlab engine from running. In addition, when there is an error on building the model, an error message will be shown on the console of GME describing the error for the user. If no error has occurred, the interpreter will continue with the file generation.

```
//Generator_Connection_Check
...
if (gen_conn_check < total_gens){
error = true;
Console::Out::WriteLine("Error: Generator(s) not Connected.");
}
```

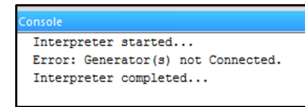


Figure 5. Demonstration of Error Messages in the Console

```
}
else if (gen_conn_check > total_gens){
error = true;
Console::Out::
WriteLine("Error: Multiple connections from a single Generator.");
}
```

The following constraints are implemented in the interpreter to ensure the model is created correctly:

- The total available generation power should be greater or equal to the total power demand
- Every Load can only be supplied from *port* bus or *starboard* bus
- Each Bus in the system must have a unique *Bus_Num* attribute
- Each Generator should be connected to the bus at one and only one spot
- To ensure the continuity of the power flow only one branch can be switched off at one time
- Bus voltage magnitude should be within *V_Max* and *V_Min*
- Generator active power and reactive power supply should be within their own limits

If any of these constraints is broken, one of the error messages from Figure 5 will show up in the GME console.

5) *Step.5 Synthesis of configuration information:* Sample script below shows the generated m-file. Once the constraints are all satisfied, loops need to be created to traverse through the arrays to print out the data in the correct locations. The Interpreter also creates the command file to run the case file. The bus data figures and the branch data figures are saved in a file for further examination. Once the data file is ready, the interpreter will invoke the Matlab engine.

```
// Configuration_File_Synthesis
...
fprintf ( matlab_file , "function mpc = case_1");
fprintf ( matlab_file , "mpc.version = '2'");
fprintf ( matlab_file , "mpc.baseMVA = 10");
fprintf ( matlab_file , "mpc.bus = [ \n");
//Print_Bus_Array
for(counter1 = 0;counter1 < total_buses;counter1++){
for(counter2 = 0;counter2 < 11;counter2++){
if (counter2 == 6)
fprintf ( matlab_file , "\t1");
if (counter2 == 9)
fprintf ( matlab_file , "\t1");
fprintf ( matlab_file , "\t%.2f", bus_array [counter2][ counter1 ]);
}
}
fprintf ( matlab_file , "; \n");
...
}
```

6) *Step.6 Invoke Matpower solver to execute the generated models:* Matlab engine contains a series of API functions which supports C/C++, Fortran among many[16]. These functions are used to invoke Matlab engine and execute Mat-

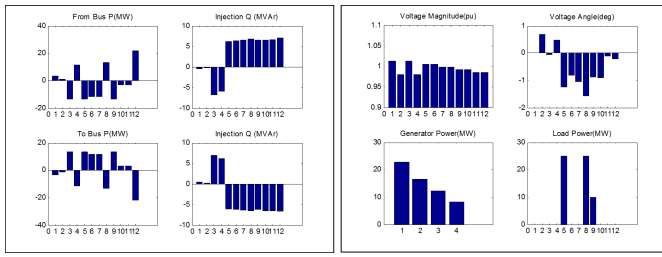


Figure 6. Optimal Power Flow Results

lab scripts directly within the other programming environments. Data (variables, arrays, matrices, etc.) can be transferred between the C++ workspace and Matlab workspace bidirectionally. Designers can directly call Matlab functions in C++ instead of coding the complicated functions by hand.

```
//Invoke_Matlab_Engine_and_Solve
...
Engine *ep; // define Matlab engine pointer .
char MatlabPath[100]; // current Matlab Path
char p[6000]; //Matlab return buffer
int n=6000; //Matlab return buffer size
ep=engOpen(NULL);
engOutputBuffer(ep, p, n); //push the Matlab output into the buffer
TCHAR NPath[MAX_PATH]; //current C++ project path
GetCurrentDirectory (MAX_PATH, NPath);
strcpy (MatlabPath, "cd ");
strcat (MatlabPath, NPath);
engEvalString (ep, MatlabPath); //change the Matlab project path to the C++ path
engEvalString (ep, "Matpower"); //execute the m-file
Console::Out::WriteLine(p);
...
```

7) *Step.7 Display of the simulation results:* The simulation results are presented in figure 6.

The purpose of integration with MatPower case study is to provide a tangible approach demonstrating the utility of the proposed environment. However, Matpower has its own limitations. In the second case study, we integrate PSAT with more functionality including dynamics and larger component libraries.

B. Case study II: Integration with PSAT

This case study exploits the potential of the proposed generic modeling concept through well tested open source tool PSAT[17]. Static and dynamic analysis algorithms defined within PSAT supports the corresponding models including power flow data, switches, loads, measurements, loads, machines, controls, FACTS, and user defined models[18]. PSAT provides graphical user interface (GUI) with a customizable Simulink-based library to assist the system design, in addition to command line interface. In this case, PSAT is explored through the command line interface rather than existing GUI to access the global structure, modify component parameters, set desired options, and directly invoke PSAT engine to execute configuration files. For the second case study, our main purpose is to expand the design environment to another domain specific toolkit with similar data structures and specification formats. It suggests that the interpreter design, data collection procedure, constraints settings, and configuration synthesis

process will be of very close patterns from the previous design in case study two. Thus, the meta-model creation and the application design are the emphasis with the demonstration of more functionality, more designer interface flexibility, and more detailed results analysis capabilities.

1) *Create the Meta-Model:* The set of rules for meta-model development follows the similar concept as defined for the first case study. However, the inclusion of the system dynamics in PSAT requires more information to reflect the system characteristics. Thus the meta-model needs to be extended with more entities and more detailed parameter specifications. Every power system analysis tool can have their unique data requirements based on the unique design philosophy. While the basic information requirement from power system modeling remains same, upgrading the system utilizing particular tool operations or adding different functionalities is a common scenario for meta-model update.

Interactions between generators, transmission line, load and associated controls, as well as those components themselves, are supplemented through meta-model as shown in figure 7. Generators are abstracted by fourth order model and IEEE type 1 automatic voltage regulator (AVR). Load models are described in rich details including constant power, ZIP, and exponential recovery in the meta-model. As testing the vulnerability of SPS with various fault scenarios could greatly strengthen the stability and survivability, this concept is captured in meta-model as fault atoms with various parameters including fault interval, resistance and reactance magnitude. Breakers are made available with switching intervals to trip the faulted buses. Transformer, cable, bus, and other components remain same as defined in first case study. Thus generated library from the meta-model for this case is shown in figure III-B2. The Running_modes setting in the library, as shown in figure 9, allows application model designer to specify one of the operations among power flow, continuous power flow, time domain and small signal stability analysis (eigenvalue analysis). Main settings of the system simulation such as frequency, power base ratings, starting and finishing times, can also be modified by designers at any time through the Simulation_parameter setting.

2) *Create the Application Model:* Application model remains almost same across different tools; however, inclusion of different features can encourage application developer to add corresponding functionalities based on the needs. The application model as shown in figure 11 is similar to the SPS model presented in Case Study I but, dynamics associated analysis and evaluations can be accessed by the environment designers in this case.

3) *Verification of Simulation Results:* Time-Domain simulation results of bus voltages changing before and after a three phase fault are presented in Figure 12. The system was in stable steady state operation before the fault is applied at $t=16$ seconds. However, after the fault, system voltage oscillates and becomes unstable. Reconfiguration action is not taken into account. This is a simple demonstration of a variety of utilities provided in the design interface. Designers can choose

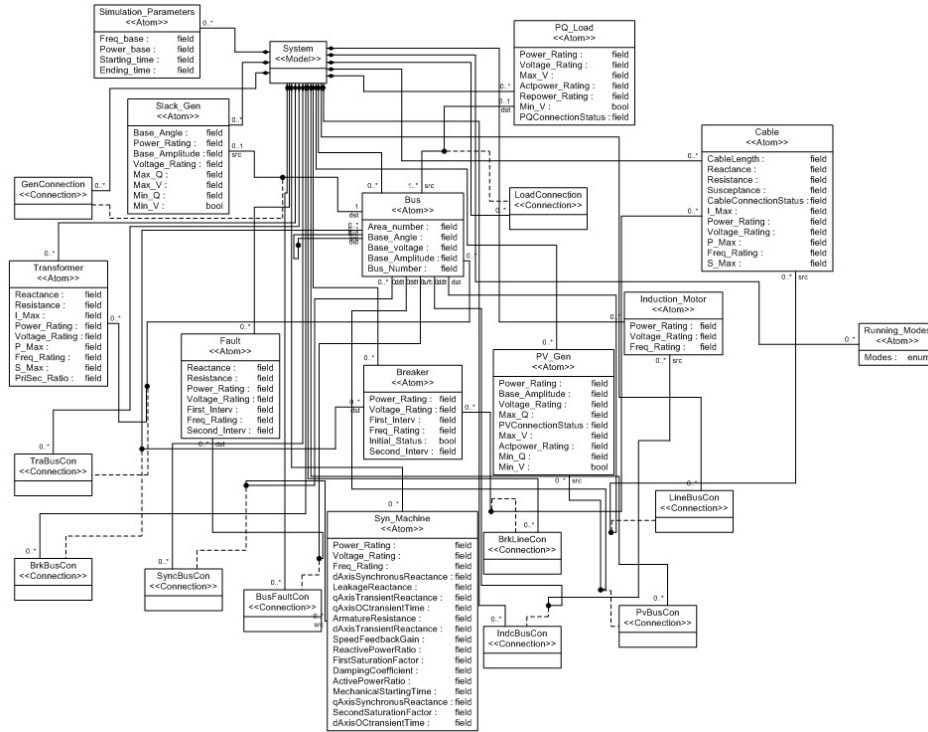


Figure 7. Meta-Model Design for Case_2

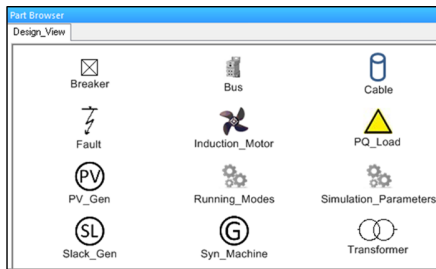


Figure 8. Entities Library Generated From Meta-Model

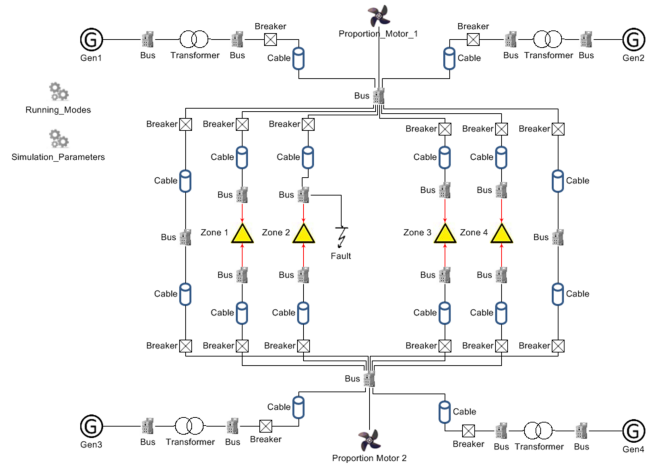


Figure 11. Application Model for Case_2

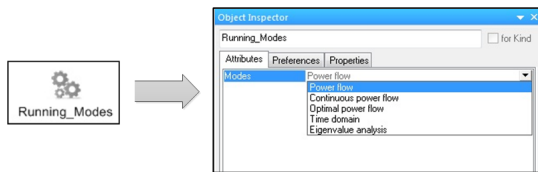


Figure 9. Running_Mode Settings

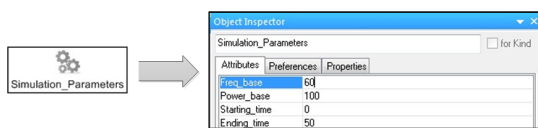


Figure 10. Simulation_Parameter Settings

different types of analysis to evaluate the comprehensive performance of the application model.

IV. CONCLUSION AND FUTURE WORK

In this paper, the concept of Model Integrated Computing is presented to integrate with different power system toolboxes, in order to support the development and analysis of shipboard power system and other similar systems. GME toolkit is utilized to develop the meta-model and application model. Currently, Matpower and PSAT toolbox are supported by the proposed environment for synthesizing applications

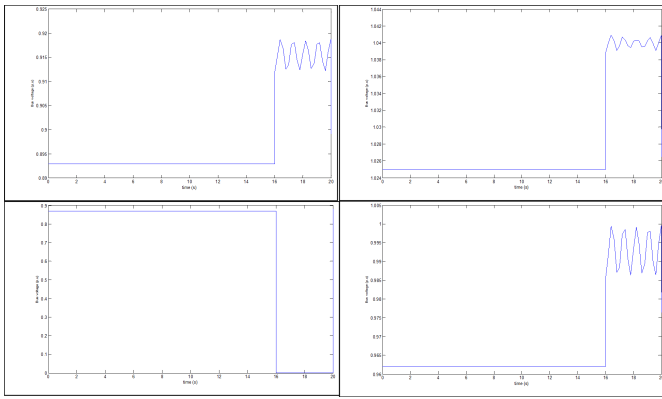


Figure 12. Bus Voltage response to fault

and performing various kinds of system analysis. In future, the current design will be updated and expanded to other computational platforms to exploit the full potential of the model-based approach.

One of targeted simulation platform is VTB developed by South Carolina University[19]. VTB contains an open simulation framework that supports the development of multi-scale and multi-disciplined systems, which makes it ideal for power system analysis. VTB also supports external reuse or repackaging utilizing the framework, solvers and components through .NET and .COM interfaces. Even, simulation scenarios and control algorithm needs to be tested in real time before it gets deployed for application. Dedicated power system simulation platform such as RTDS, with the ability to integrate with hardware is another potential extension to this work. RTDS is one of such platform that allows real time applications testing and faster simulations along with other benefits. The integration of VTB and RTDS to the present environment would further prove the extendibility and practicability of the proposed integrated modeling approach.

REFERENCES

- [1] J. Sztipanovits, G. Hemingway, A. Bose, and A. Srivastava, "Model-based Integration Technology for Next Generation Electric Grid Simulations," *Computational Needs for Next Generation Electric Grid*, pp. 4.11–4.44, 2011.
- [2] G. Karsai, J. Sztipanovits, A. Ledecz, and T. Bapty, "Model-integrated development of embedded software," *Proceedings of the IEEE*, vol. 91, no. 1, pp. 145–164, Jan. 2003.
- [3] J. Sztipanovits and G. Karsai, "Model-integrated computing," *IEEE Computer*, pp. 110–111, 1997.
- [4] A. Ledecz, M. Maroti, and P. Volgyesi, "The generic modeling environment," Vanderbilt University, Tech. Rep., 2004.
- [5] A. Dubey, X. Wu, H. Su, and T. Koo, "Computation platform for automatic analysis of embedded software systems using model based approach," *Third International Symposium on Automated Technology for Verification and Analysis(ATVA)*, pp. 114–128, 2005.
- [6] G. Karsai, J. Sztipanovits, and H. Franke, "Towards specification of program synthesis in model-integrated computing," *Proceedings IEEE ECBS*, pp. 226–233, 1998.
- [7] R. Amgai, J. Shi, and S. Abdelwahed, "Lookahead Control Based Framework for Power System Applications," *North American Power Symposium(NAPS)*, pp. 1–6, 2012.
- [8] W. Gao and S. Musunuri, "Hybrid Electric Vehicle Modeling and Analysis in Generic Modeling Environment," *2006 IEEE Vehicle Power and Propulsion Conference*, pp. 1–6, Sep. 2006.
- [9] A. Bakshi, V. Prasanna, and A. Ledecz, "MILAN: A model based integrated simulation framework for design of embedded systems," *ACM Sigplan Notices*, 2001.
- [10] GME Toolkit, "<http://www.isis.vanderbilt.edu/projects/gme>."
- [11] A. Ledecz, M. Maroti, and A. Bakay, "The generic modeling environment," *Proceedings of the IEEE International Workshop on Intelligent Signal Processing*, 2001.
- [12] P. Volgyesi and A. Ledecz, "Component-based development of networked embedded applications," *Proceedings. 28th Euromicro Conference*, pp. 68–73, 2002.
- [13] Matpower, "<http://www.pserc.cornell.edu/matpower/>."
- [14] R. D. Zimmerman, C. E. Murillo-sánchez, R. J. Thomas, L. Fellow, and A. M. Atpower, "MATPOWER : Steady-State Operations , Systems Research and Education," *IEEE Transaction on Power Systems*, vol. 26, no. 1, pp. 12–19, 2011.
- [15] Syntek, "DD(X) Notional Baseline Modeling and Simulation Development Report," *Internal Report*, Aug. 2001.
- [16] "Matlab Documentation: Call Matlab Engine." [Online]. Available: <http://www.mathworks.com/help/matlab/calling-matlab-engine-f>
- [17] PSAT, "<Http://www.uclm.edu/area/gsee/Web/Federico/psat.htm>."
- [18] F. Milano, "An Open Source Power System Analysis Toolbox," *IEEE Transactions on Power Systems*, vol. 20, no. 3, pp. 1199–1206, Aug. 2005.
- [19] Virtual Test Bed, "<http://vtb.engr.sc.edu/vtbwebsite/>."