

Deploying Modelica Models into Multiple Simulation Environments

Yuri Chernukhin^{*}, Maxim Polenov^{*}, Chandrasekhar Vemulapally^{**},
Eugene Solodovnik^{***}, H. Alan Mantooth^{**}, Roger Dougal^{***}

^{*} Department of Computer Engineering, Taganrog State University of Radio Engineering, Taganrog, Russia

^{**} Department of Electrical Engineering, University of Arkansas, Fayetteville, AR, USA

^{***} Department of Electrical Engineering, University of South Carolina, Columbia, SC, USA

ABSTRACT

A model import tool referred to as the MultiTranslator (MT), which is used by the Paragon⁺ modeling environment is described in this paper. The MultiTranslator is based on a standard XML format and also provides an interactive wizard to add and/or edit any information to the imported model. The approach described here also further demonstrates how Paragon's XML schema can act as a centroid for third party tools and make the models available in widely used languages like VHDL-AMS, Verilog-AMS and MAST. This mechanism is illustrated with importing a Modelica model into the Virtual Test Bed (VTB) simulation environment.

Keywords

Modelica, model import, XML, tools, HDLs

1. INTRODUCTION

The need for modeling different electrical and mechanical devices in the same simulations is gaining increasing focus as more sensor/actuator systems are deployed. The SPICE-based simulators lack a convenient way to develop either mixed-signal or mixed-technology models. Mixed-signal Hardware Description Languages (HDLs) [1], [20] are providing the flexibility of letting the designer write his/her own models. Knowing the fact that developing and maintaining behavioral models even in these HDLs is a time-consuming and error-prone process, making these models available in multiple simulation environments has become another major requirement in the contemporary design world [23]. The language technology alone is not sufficient to meet the challenges of the complexity of the design of real systems. This is due to the fact that complexity of such systems is now involving people from a variety of scientific backgrounds. It is very unlikely that each of these disciplines will have taught the use of the same languages or the same simulator.

With the aid of advanced modeling tools, models can be efficiently shared among designers in their

convenient environments for verification purposes. The objective of this paper is to illustrate the usefulness of tools like Paragon [2-4] and MultiTranslator [5], [6] where both use the same language-independent format to represent model information. The MultiTranslator is a software tool developed at the Taganrog State University of Radio Engineering (TSURE) intended to translate models written in Advanced Continuous Simulation Language (ASCL) [7], Modelica [8] and other modeling languages. Each translator is equipped with an individual grammar module, which can translate the model information into other formats. Among the existing MultiTranslator modules, the Modelica importer is described in this paper. The imported Modelica model is embedded and simulated in the VTB [9], [10] simulation engine to verify the conversion. Paragon is a language-independent modeling environment developed at the University of Arkansas, which has both the capabilities of automatic model importation and code generation of the imported models in various HDLs [23]. Compact semiconductor device models like BSIMSOI [12] and EKV [13] have been entered to generate Verilog-A and VHDL-AMS codes [14]. Paragon's own model import mechanism was first created to import MAST [15] models.

The VTB is a simulation environment developed at the University of South Carolina for prototyping of large-scale, multi-technical dynamic systems. It allows proof-testing of new designs prior to hardware construction. The application driving development of the VTB is advanced power systems for navy platforms. In particular VTB provides a wide range of possibilities for integrating dynamic models from other simulation environments.

2. MOTIVATION

The VTB is an efficient mixed-technology simulation and visualization tool. As in normal simulators, there is always a growing need for new models to be added to the existing library. The simulation kernel supports a C++ based interface for models rather than popular HDLs. Handling simulator specific information in models also complicates the procedure of bringing new models into VTB. To fulfill this need, Paragon was chosen as a companion tool to bring new models into the VTB environment. The model importer extracts all the model information and removes all language specific constructs to save in Paragon's internal format. Once a model is rendered in this internal format, it can

⁺ This work is sponsored by the Office of Naval Research under Subaward No. USC 01-636 and by the Semiconductor Research Corporation (SRC) under Grant No. 2005-HJ-1287

be viewed pictorially and better understood with the help of graphical editors. The biggest advantages of the model importing mechanism are to: a) harness the code generation capability of Paragon to generate code for the same model in multiple forms, b) as a teaching/training aid to others needing to understand the internal workings of a previously coded model, c) the creation of a more object-oriented model such as that found in Paragon, and d) automatic generation of model documentation (analogous to model data sheets).

3. PARAGON'S XML AS A CENTROID

Paragon utilizes a generic XML schema, which enables the capture of information specific to model data. The use of XML, which is open source and a standardized format, allows easy data interchange and formatting. Many standard language translation tools like Extensible Style Sheet Language Transformations (XSLT) [24] can be used to manipulate the data and convert into necessary target format. In addition to the HDL code generation functionality, Paragon can also be used in combination with third party tools like ADMS [25], [26] and MCAST [27] to generate low level C code for target simulators. ADMS and MCAST are model compilers, which are based on Abstract Syntax Trees (AST) that read compact models described in high level languages like VHDL-AMS/Verilog-AMS and automatically generate C code that can be linked with existing circuit simulators like Spectre and SPICE3. One can imagine the amount of time, effort and complexity involved in writing and debugging huge models such as those in the BSIM family in these languages before actually deploying in simulations. The usefulness and capacity of high level advanced modeling tools like Paragon was illustrated by generating BSIM3-SOI model (version 2.2) and validating against the built-in Spectre model in [14]. Another example of Paragon's capacity is illustrated by the VBIC implementation as described in [28]. In each case, the Verilog-A code generated by Paragon was fed into ADMS to compile it into C for the compiled model interface of Spectre.

In order to address model debugging issues and to reduce simulation time, Paragon has a rich set of analysis and utility methods such as AST and Model Checking. The AST is created for each model to identify and represent the inter-relationships between different time-varying variables and constants of the model. The AST is analyzed to determine functional and time dependencies in the model, which enables the generation of efficient and readable code. The model import mechanism also utilizes a combination of the above mentioned utilities and distinguishes between sequential and simultaneous blocks of the model. It also identifies discontinuities and undefined variables before actually saving the model to the database, while the code generators are designed to generate proper code (e.g. generation of break statements in VHDL-AMS). All of these features differentiate the model importer

from basic language translators. So far, continuous time models have been the focus for model importing while the research effort is ongoing to include event-driven behavior. The newly developed Modelica Importer is discussed in this paper. Modelica is an object oriented modeling language, which is designed to allow convenient component-oriented modeling of complex physical systems (e.g., systems containing mechanical, electrical, electronic, hydraulic, thermal, control, electric power or process-oriented subcomponents). As there is a requirement for bringing Modelica mixed-technology and power device models into the VTB simulation environment, Paragon acts as a hub from where multiple languages can be generated.

4. TOOLS FOR IMPORTING MODELICA

Paragon was designed to create a user-friendly modeling environment, which would alleviate the need of mastering various HDLs [16]. The present design utilizes XML and MathML [17], [18] as the internal format. The extensible nature of XML easily facilitates addition of new features and phenomena, for example, multi-physical systems (thermal, optical, mechanical, etc). Paragon's XML template can be best described in a diagram shown in Fig. 1 below.

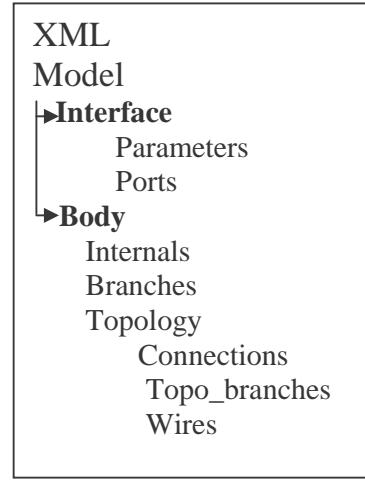


Fig. 1. Paragon's internal XML template.

The entire model information is encapsulated in the *Model* parent tag. The *Model* tag in turn is sub-divided into two tags *Interface* and *Body*, where the former represents the model parameters and connection points information and the latter incorporates the model implementation details. Parameter processing and ranges of validity information is saved in the *parameter* tag whereas the name, type and nature of connection points are saved in the *port* tags. The model can be viewed pictorially by a set of branches where each branch is associated with through and across variables. The *Topology* tag gives the branch connections and wiring information of the model.

The general template of a Modelica model is shown in Fig. 2. The structure of a Modelica-model can be divided into three sections, each having its own significance. When compared against the Paragon's XML format, section 1 contains the information of model parameters, ports and internal variables. Both the parameters and ports information are saved in the *Interface* tag, whereas the internal variables are saved in the *Body -> Internals* section. In a similar way section 2 carries all the model topology information, which is saved in the *Topology* and *Branch* tags. Section 3 represents the characteristic equations governing the model behavior and this information is saved in the *equation* tags of the Paragon format. All the model expressions and equations are expressed in MathML, which is an XML application by itself for describing mathematical notations.

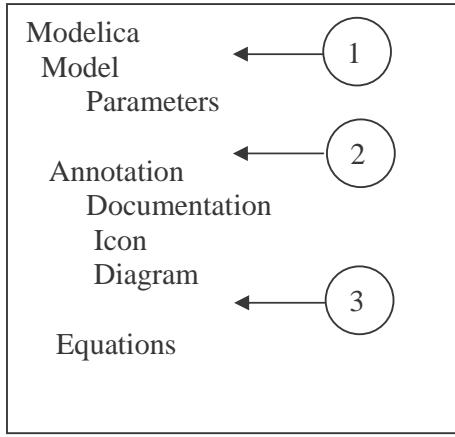


Fig. 2. General Modelica template structure.

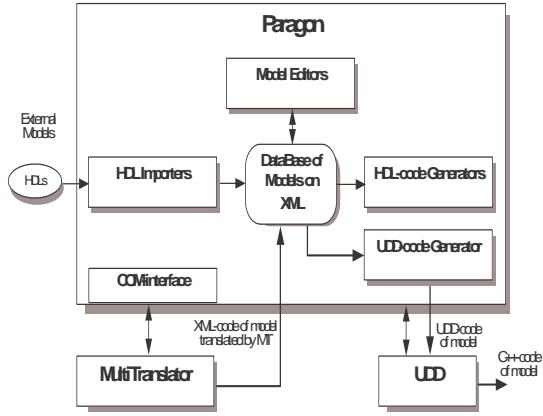


Fig. 3. Block diagram of the model importer in Paragon.

4.1. Implementation

The MultiTranslator utilizes a grammar module for translating modelica models into the target XML description. The block diagram in Fig. 3 shows the

overall process of the model import mechanism. The MultiTranslator acts as a plug-in tool for importing Modelica models into Paragon, whereas the native VTB C++ code is generated by an automatic code generation module called UDD (User Defined Device) [19]. Further the generated C++ code is compiled to a Dynamic Linked Library (dll) and then loaded into the VTB model library. The integration of Paragon and the MultiTranslator is implemented using the client-server Component Object Model (COM) interface. The MultiTranslator acts as a server and responds to the function calls made by Paragon to import the Modelica model.

The MultiTranslator's grammar module contains the description of the Modelica language grammar constructs and a set of rules. These rules define the actions to take place based on the input to generate the correct XML output. A section of the Modelica grammar module defining two rules is shown in Fig. 4. Every rule is followed by a keyword *rule* and actions to be performed are defined within the body of the *rule* section. Multiple actions can be performed based on the input in the *variant* section. The example given below demonstrates how mathematical operators are converted to equivalent XML format. The *<mo>* tag is the *presentation* tag used in MathML to represent mathematical operators. After parsing the Modelica model, the grammar module converts the model to Paragon's format on a query-driven basis.

```

rule <"Add_Minus_Op">
{
    variant
    {
        term "+<mo>+</mo>" {strParameter = strParameter +
            "<mo>+</mo>"}
    }
    variant
    {
        term "-<mo>-</mo>" {strParameter = strParameter +
            "<mo>-</mo>"}
    }
}
rule <"Power_Op">
{
    variant
    {
        term "<^>" {strParameter = strParameter +
            "<mo>^</mo>"}
    }
    variant
    {
        term "<Primary><^><Primary>" {strParameter = strParameter +
            "<mo>*</mo> + <mo>*</mo> ^ <mo>*</mo>"}
    }
}

```

Fig. 4. Example of rules defined in the Modelica grammar module

4.2. Model Wizard

The Model Wizard is a part of the MultiTranslator tool, which allows the user to interact with the tool while importing a Modelica model. This wizard allows the user to make changes when the model is being imported into Paragon, such as augmenting the model with additional information, while still keeping the Modelica model

unaltered. The screenshot of the model wizard is shown in Fig. 5 illustrating with a DC motor model example given below. In the example shown, the model consists of a set of variables and two equations describing the model behavior. Though this is legal in Modelica, in HDLs connection points are needed to instantiate the model in a netlist. In this case, this wizard will give the user the flexibility of entering connection points, their natures and branches between connection points. The wizard also helps the user to visualize the model graphically and correlate the equations to respective branches in the model.

```
model DCMotor "DC Motor"
  Real Tq=10.0 "Torque applied at the shaft";
  Real V=20.0 "Voltage across terminals A and B";
  Real i "Current through terminal A";
  Real w "Angular rotor speed";
  Real J=0.5 "The rotor equivalent moment of inertia";
  Real R=0.4 "Resistance";
  Real L = 0.0025 "Self-inductance";
  Real wo=125.664 "Rated angular speed";
  Real V0=115.0 "Rated voltage";
  Real b=0.196 "Constant of tough friction";
equation
  V=L*der(i)+R*i+ V0/w0*w;
  Tq=J*der(w)+ V0/w0*i-b*w;
end DC Motor;
```

Once the model is imported into the Paragon modeling environment, native VTB C++ models can be generated through the UDD code generation module. This UDD code generation module expects the model information to be in a specific format before generating native VTB models. For example, UDD always expects equations having through variables as a function of across variables. The Paragon's internal code generation and analysis tools are capable of converting the model information into UDD compatible format. The UDD file for the DC motor model is shown in Fig. 6.

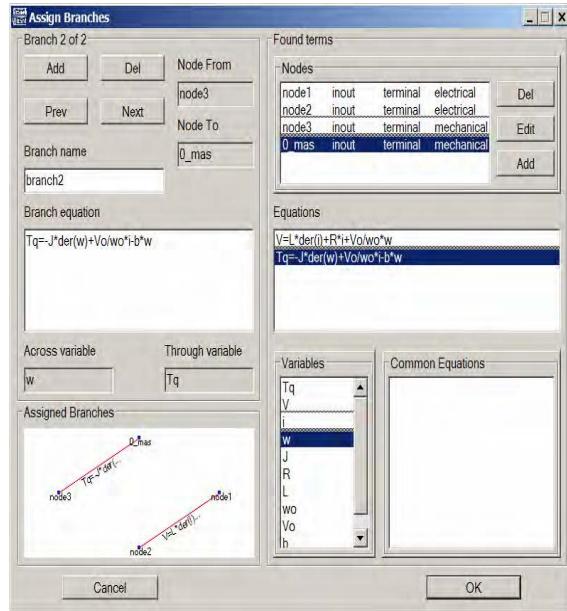


Fig. 5. Screenshot of the MultiTranslator's Model Wizard showing DC motor model example

```
! UDD input Model file of DCMotor generated by Paragon
! This is a machine generated code.
! Generated on Wed, 13 Apr 2005 11:32:03 AM
```

```
Name: DCMotor
Nodes: 3
Terminals: 3
USE Model DCMotor0

Model DCMotor0
{
  Tq=(((J*(diff(w)))+(V0/w0)*i0)-(b*w))
  pari=((INTEG(((v0-v1)-(R*i0)-(V0/w0)*w))/L)

  pari
  -pari
  Tq
}
```

Fig. 6. UDD file generated by Paragon to generate native VTB models

5. RESULTS

The last step before exporting the models into the VTB simulation environment is to compile the generated C++ code to a *dll* and saving in the model library. The VTB symbol editor can be used to create a symbol for the new model for further instantiation in the simulations. The DC motor model was imported into Paragon and the equivalent VTB model was generated utilizing the UDD tool. A *drive* system consisting of a petrol engine, flexible shaft, single-phase generator, DC-motor and ventilator was created in the VTB environment. Two DC motors were used while one of them served as a generator. To verify the results, two drive systems were created, one with the components available in the VTB model library, whereas the other with models imported and generated using MultiTranslator, Paragon and UDD tools. Both the test setups are shown as a screenshot in Fig. 7. Both the mechanical and electrical characteristics like voltage values for the generators and angular speed for DC motors were plotted and compared in the results shown in Fig. 8. Many other models like gyrator, propeller, shaft and transformer were imported and verified against built-in VTB models. The model import mechanism was also very successfully implemented with MAST models.

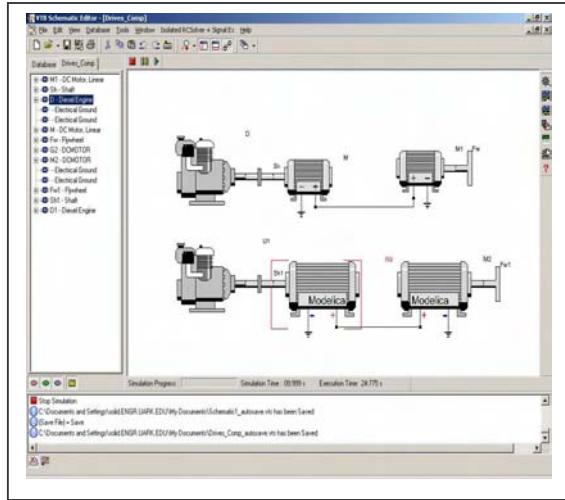


Fig. 7. Screenshot of the drive systems setup in the VTB simulation environment

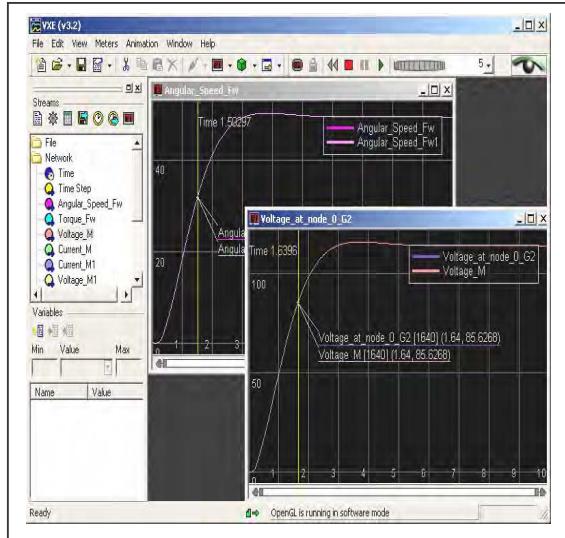


Fig. 8. Simulation results showing the voltage and angular speed values for the Generator and DC motor for drive systems

6. CONCLUSIONS

This paper describes an example of how a standard, open-source interface such as the XML format used in Paragon can be leveraged to deploy models of one language into multiple other possibilities. Consequently, this prevents models from becoming obsolete when designers switch to different design environments. Modelica models were imported into Paragon and then exported into C++ (via the UDD format) for the Virtual Test Bed mixed-technology system simulation environment.

7. REFERENCES

- [1] R.S. Cooper, *The Designer's Guide to Analog and Mixed-signal Modeling*, Avanti Corporation.
- [2] Lynguent, Inc, <http://www.lynguent.com/Company/company.html>
- [3] V. Chaudhary, M. Francis, X. Huang, H. A. Mantooth, "Paragon - A mixed-signal behavioral modeling environment," *IEEE Int. Conf. on Communications, Circuits, & Syst. (ICCCAS)*, vol. 2, pp. 1315-1321, Chengdu, China, June 2002.
- [4] M. Francis, V. Chaudhary, H. A. Mantooth, "Compact modeling of semiconductor devices using higher level methods," *IEEE 2004 International Symposium on Circuits and Systems*, vol. 5, pp. v-109 - v-112, May 2004.
- [5] MultiTranslator, Taganrog State University of Radio Engineering, <http://eng.tsure.ru/>
- [6] Y. Chernukhin, V. Guzik, M. Polenov, "Tools of external models translation and import for virtual simulation systems," Proceedings of 5th International Conference on Computer Simulation. vol.1, – St. Petersburg, Russia, June 2004.
<http://www.aegistg.com/ACSLCUT/products/SIM/sim.htm>
- [7] <http://www.modelica.org/>
- [8] Virtual Test Bed, University of South Carolina, <http://vtb.engr.sc.edu/>
- [9] R. A. Dougal, T. Lovett, A. Monti, E. Santi, "A multilanguage environment for interactive simulation and development controls for power electronics," *IEEE Power Electronics Specialists Conference*, vol. 3, pp. 1725–1729, June 2001.
- [10] E. Solodovnik, W. Gao, R. Dougal, "Automatic Model Generation in VTB: Phase domain Modeling of an Induction Machine," 34th Annual North American Power Symposium, Tempe, Arizona, Oct. 2002.
- [11] BSIM3SOI Source code and Documentation, <http://www-device.eecs.berkeley.edu/~bsimsoi/>
- [12] M. Bucher, C. Lallement, C. Enz, F. Théodoloz, F. Krummenacher, The EPFL-EKV MOSFET Model Equations for Simulation, Technical Report, Model Version 2.6, June 1997. Revision I, September, 1997, Revision II, July, 1998.
- [13] V. Chaudhary, M. Francis, W. Zheng, H. A. Mantooth, L. Lemaitre, "Automatic Generation of Compact Semiconductor Device models using Paragon and ADMS," *International Workshop on Behavioral Modeling and Simulation (BMAS)*, pp. 107-112, Oct. 2004.
- [14] MAST/Saber User Manual, Synopsys, Inc.
- [15] P. Mallick, M. Francis, C. Vemulapally, A. Austin, H. A. Mantooth, "Achieving language independence with Paragon," *International Workshop on Behavioral Modeling and Simulation (BMAS)*, pp. 149-153, Oct. 2003.

- [17] Extensible Markup Language (XML),
<http://www.w3.org/XML>
- [18] Mathematical Markup Language (MathML),
<http://www.w3.org/Math>
- [19] E. Santi, R. A. Dougal, A. Monti, "The VTB Environment for Virtual Prototyping of Dynamic Ship Systems," American Society of Naval Engineers Annual Meeting, Arlington, VA, Mar. 2003.
- [20] H. A. Mantooth, M. Fiegenbaum, *Modeling with an Analog Hardware Description Language*, Kluwer Academic Publishers, Norwell, MA, 1995.
- [21] P. Ashenden, G. D. Peterson, D. A. Teegarden, *The Systems Designer's Guide to VHDL-AMS*, Morgan-Kaufmann, San Francisco, CA, 2003
- [22] 1076.1-1999 IEEE Standard VHDL Analog and Mixed-Signal Extensions Language Reference Manual, IEEE Press, ISBN 0-7381-1640-8.
- [23] A. S. Kashyap, C. Vemulapally, H. A. Mantooth, "VHDL-AMS Modeling of Silicon Carbide Power Semiconductor Devices," *IEEE Workshop on Computers in Power Electronics (COMPEL)*, pp. 50-54, Aug 2004.
- [24] Extensible Stylesheet Language,
<http://www.w3.org/TR/xslt>
- [25] ADMS Model Compiler,
<http://sourceforge.net/projects/mot-adms>
- [26] L. Lemaitre, C. McAndrew, S. Hamm, "ADMS – Automatic Device Model Synthesizer," *Proc. IEEE Custom Int. Circ. Conf.*, pp. 27-30, 2002.
- [27] B. Wan, B. P. Hu, L. Zhou, C.-J. Shi, "MCAST – An abstract-syntax-tree based model compiler for circuit simulation," *IEEE Custom Integrated Circuits Conf. (CICC)*, pp. 249-252, Sept. 2003.
- [28] A. M. Francis, B. O. Woods, H. A. Mantooth, M. Vlach, L. Lemaitre, "A Methodology for Rapid Development and Simulator Integration of Compact Models," *Proc. of SRC TECHCON 2005*, 6 pgs., Oct. 2005.