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

Outline

- Introduction
- Paragon Architecture
- MultiTranslator Architecture
- Model Import Mechanism
- Examples
- Conclusion

Introduction

- Importance of modeling electrical and mechanical devices in same simulation
- How easy is it to model in SPICE ?
- Hardware Description Languages (HDLs) provide a convenient way for such applications
- Is language technology alone sufficient ?
  - Complex designs involve people from different scientific backgrounds
  - Not necessary to know all the languages
  - Need for Modeling Tools

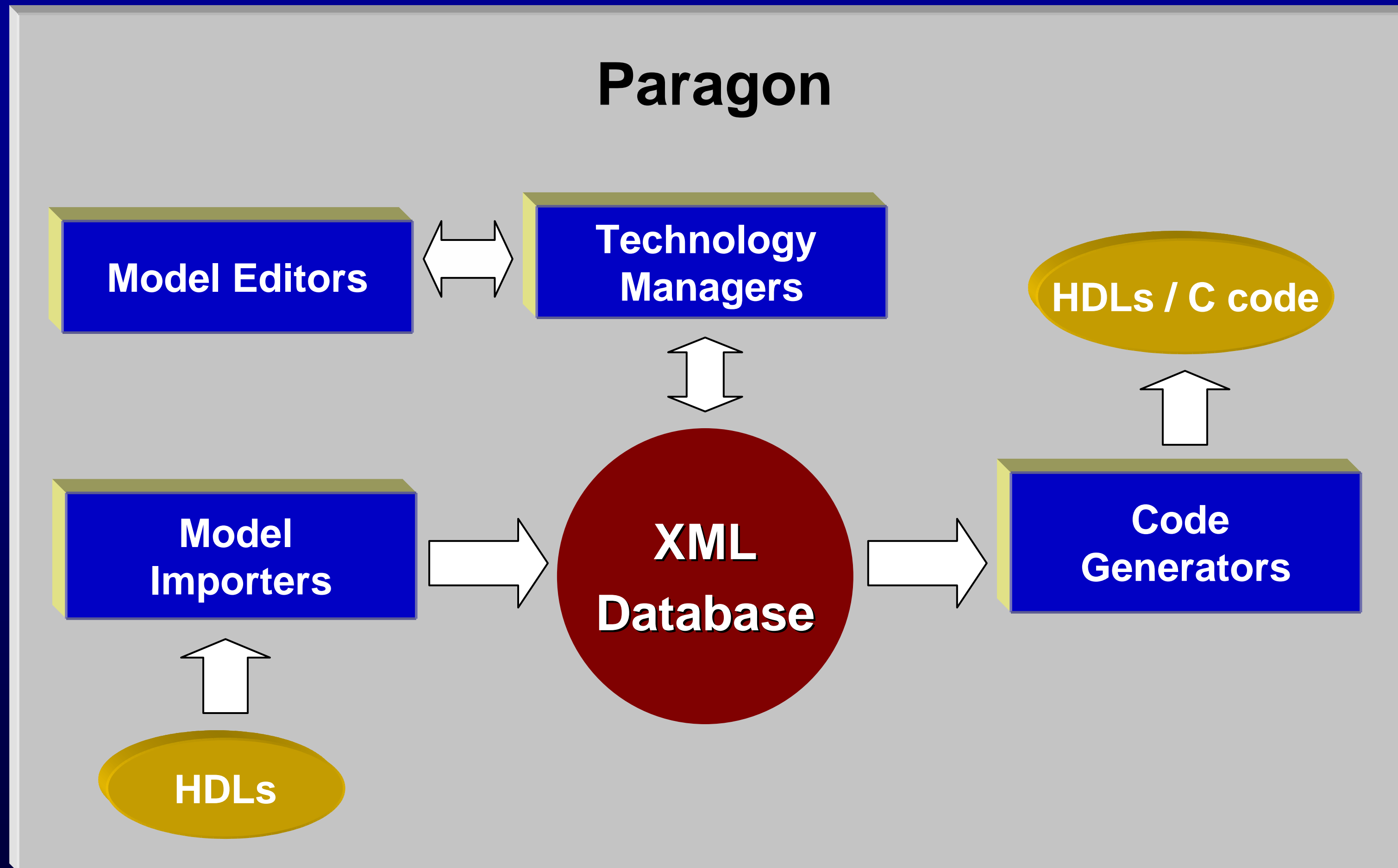
Introduction

- Motivation for Modeling Tools
  - HDL modeling is still programming
  - Error prone, tedious, time consuming and difficult to debug
  - Efficient sharing of models among designers in their convenient environments is becoming a major requirement
  - Concentrate more on modeling and design aspects rather than on syntax and semantics
  - *Language-based* and *Language-independent* tools like Paragon address all the above issues

Objective

- Virtual Test Bed (VTB)
  - Developed at the University of South Carolina
    http://vtb.ee.sc.edu/
  - Efficient mixed-technology simulation and visualization tool
  - Advanced power systems for navy platforms
  - Provides a C++ based interface rather than HDLs
- MultiTranslator (MT)
  - Developed at the Taganrog State University of Radio Engineering (TSURE)
  - Translates models written in ACSL, Modelica and etc.
- Effort to add new models to the VTB environment

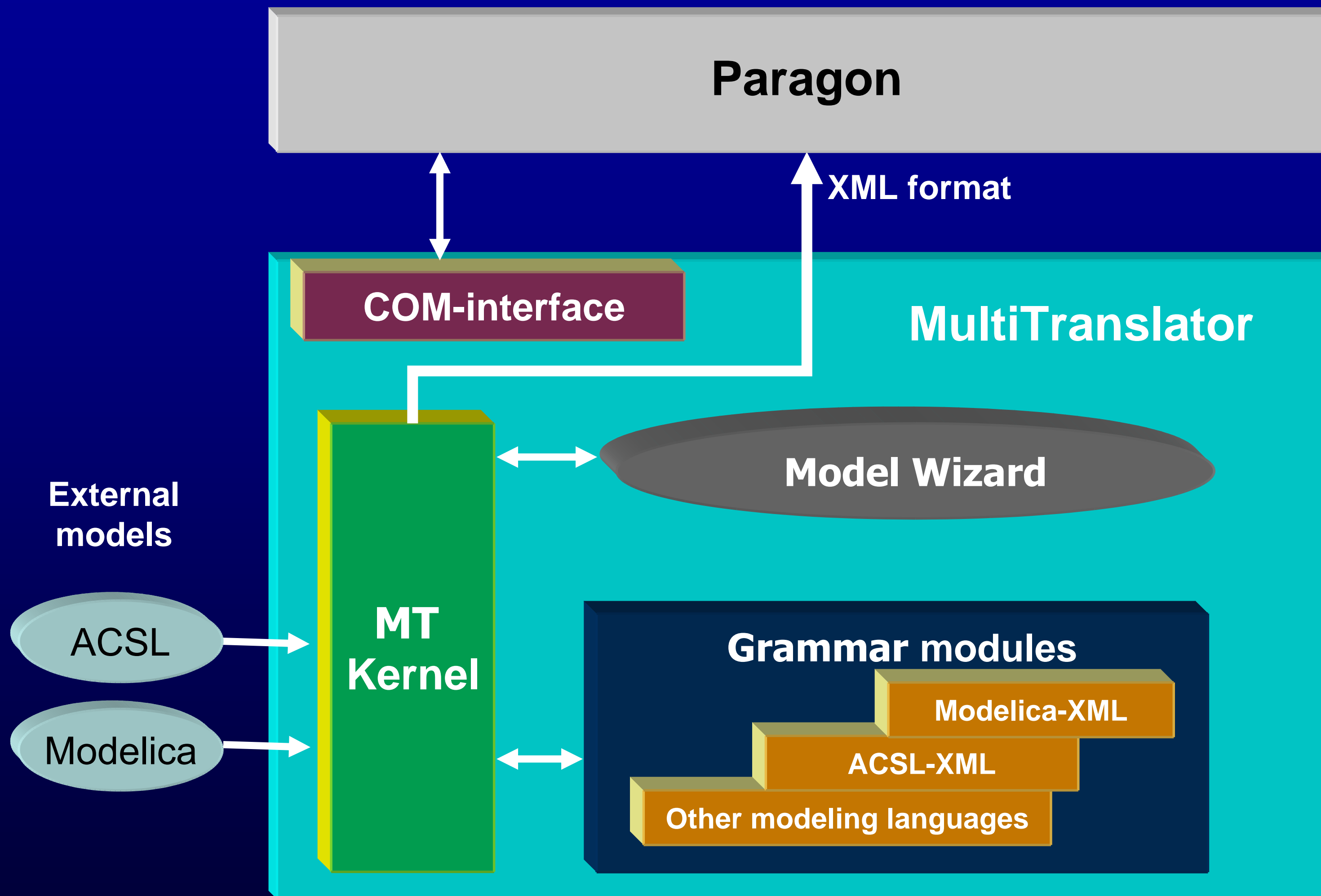
Paragon Architecture



Abstract Model Representation

- Model information saved in Common Modeling Interchange Format (CMIF)
- CMIF is basically XML/MathML
- Independent of specific language/simulator
- Able to capture constructs of VHDL-AMS, Verilog-AMS and MAST
- XML is used because it is
  - extensible
  - simple, flexible and structured
  - open-source and standardized
  - enables easy sharing of models

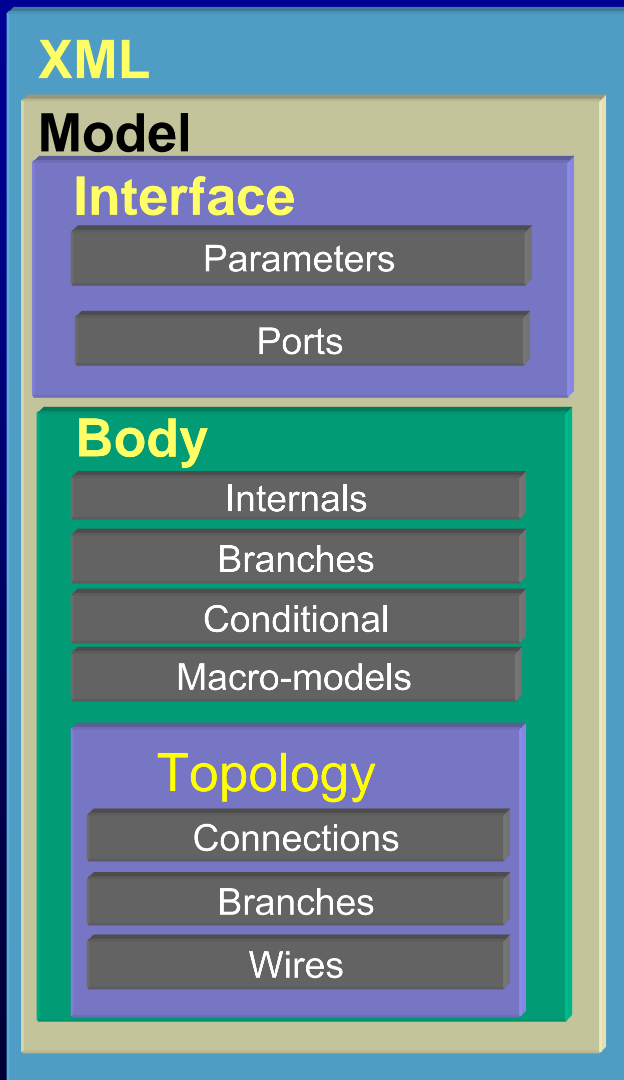
Paragon-MT

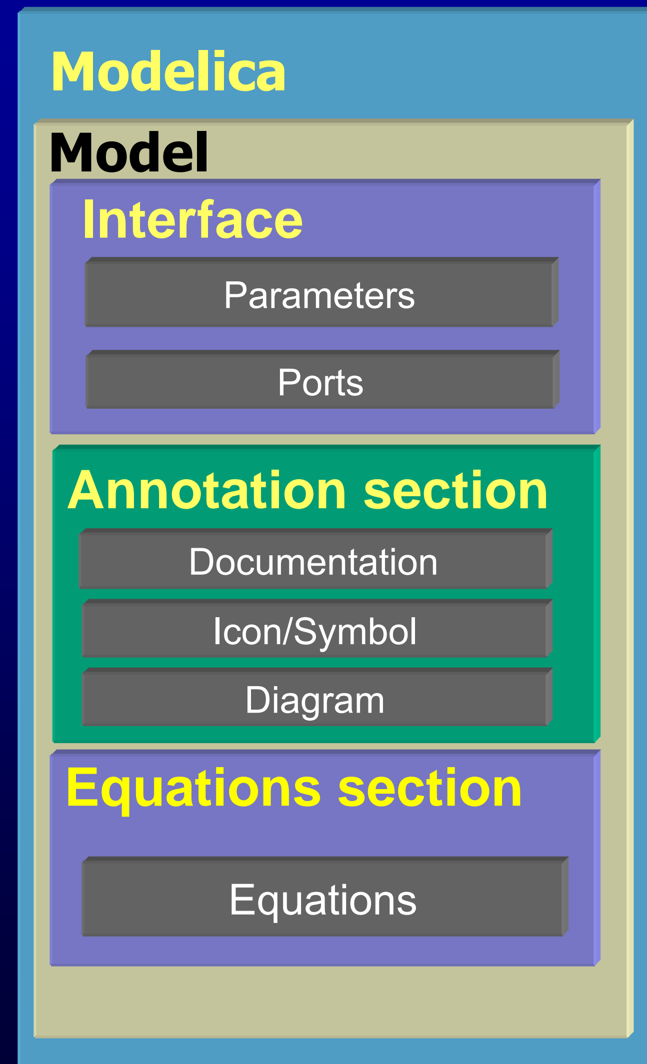


MultiTranslator Architecture

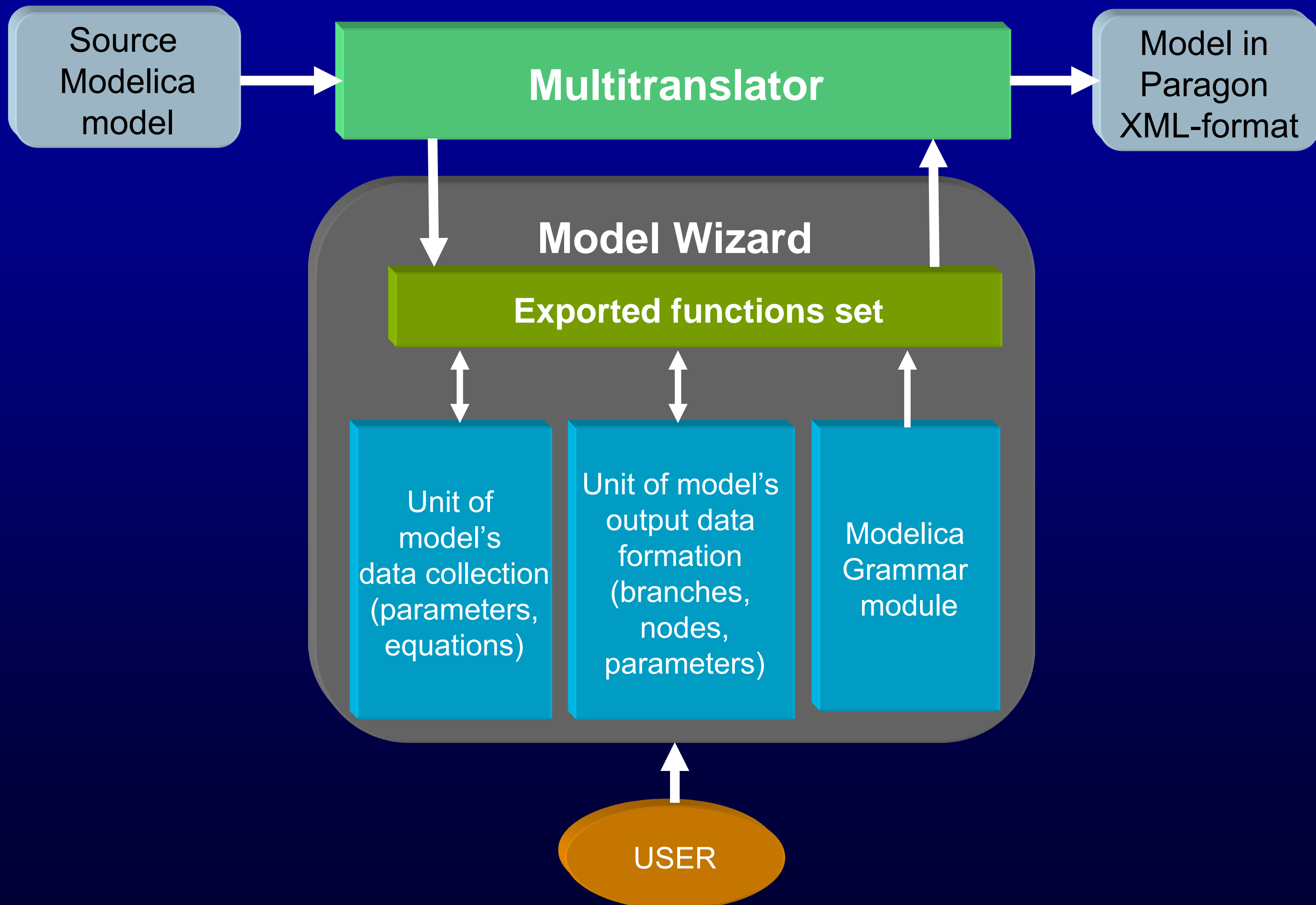
- MT acts a plug-in tool for importing Modelica models into Paragon
- MT utilizes various grammar modules
  - grammar rules and actions are defined
- Model Wizard is a part of the tool
  - allows the user to interact with the tool
  - additional information can be added
  - Connection point, parameters and branch information can be altered
- Integration is implemented using Component Object Model Interface (COM)

XML format and Modelica template





MT's Model Wizard



Screen-shot of Model Wizard

Assign Branches

Branch 2 of 2

Add | Del | Prev | Next

Node From: SHAFT

Node To: Omas

Branch name: branch2

Branch equation: T=-J*der(w)+C0*i-b*w

Add | Edit | Del

Across variable: w

Through variable: T

Assigned Branches

Omas, SHAFT, T=-J*der(w..., EC1, EC2, V=L*der(i)...

Cancel

Found terms

Nodes

EC1 analog_conservative_port electrical
EC2 analog_conservative_port electrical
SHAFT analog_conservative_port mechanical_angular_speed
Omas analog_conservative_node mechanical_angular_speed

Add | Edit | Del

Equations

V=L*der(i)+R*i+C0*w
T=-J*der(w)+C0*i-b*w

Variables: T, V, i, w, J, R, L, w0, V0, h

Common Equations: C0=V0/w0

Add | Edit | Del

OK

List of terminals and nodes of the model

Equation for the current branch

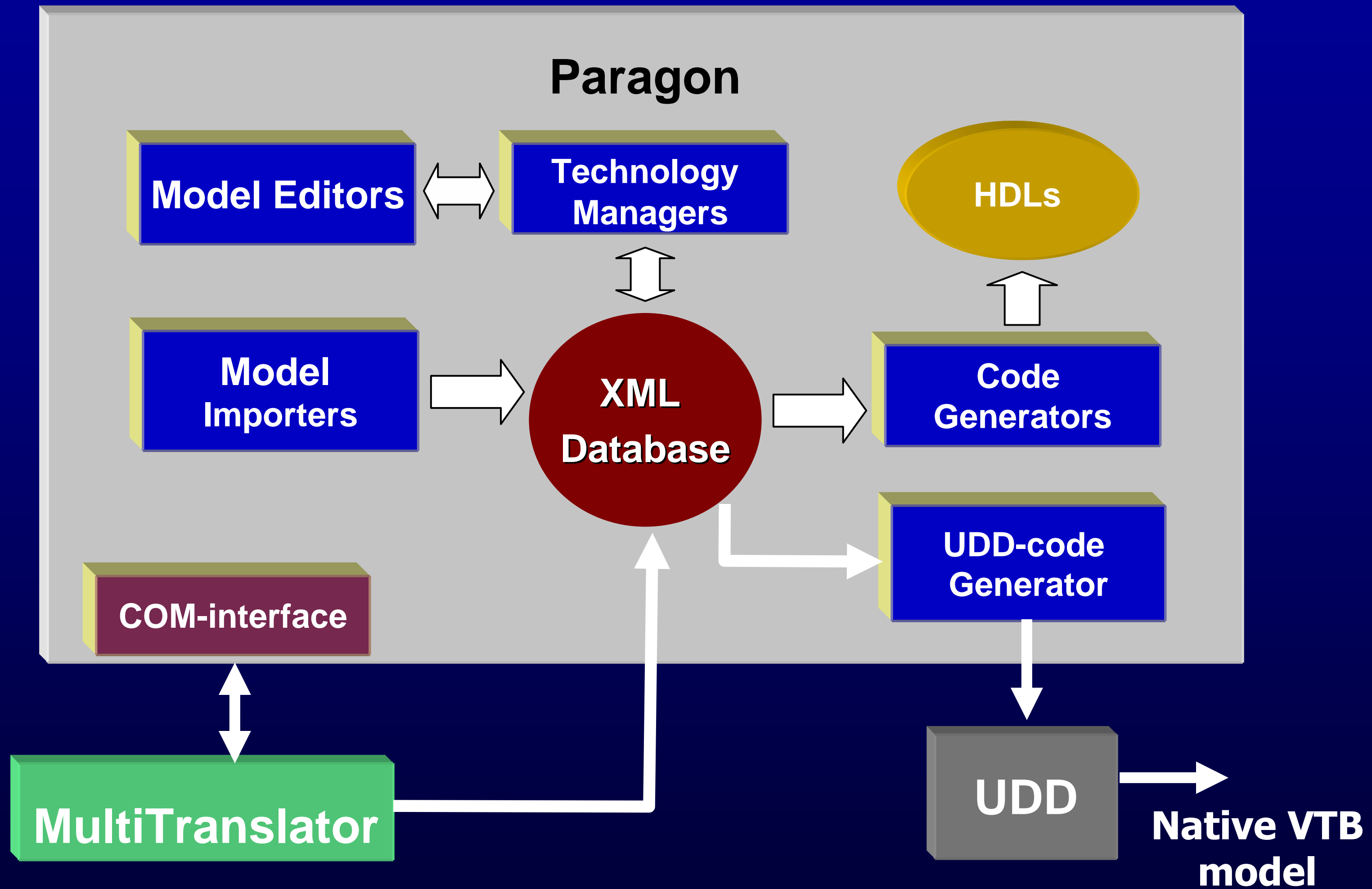
List of all branch equations

Display of branches

Model variables

General Model equations

Modelica Model Import Mechanism



Exporting to VTB

- UDD – User Defined Device
- Generated C++ code has to be compiled to a dynamic linked library (*dll*)
- Symbol can be created for further instantiation in the VTB environment

Modelica to UDD

```
model DCMotor "DC Motor"
  Real Tq=10.0 "Torque applied at the shaft";
  Real V=20.0 "Voltage across A & B";
  Real i "Current through terminal A";
  Real w "Angular rotor speed";
  Real J=0.5 "The rotor moment of inertia";
  Real R=0.4 "Resistance";
  Real L = 0.0025 “Self-inductance”;
  Real w0=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;
```

Modelica model

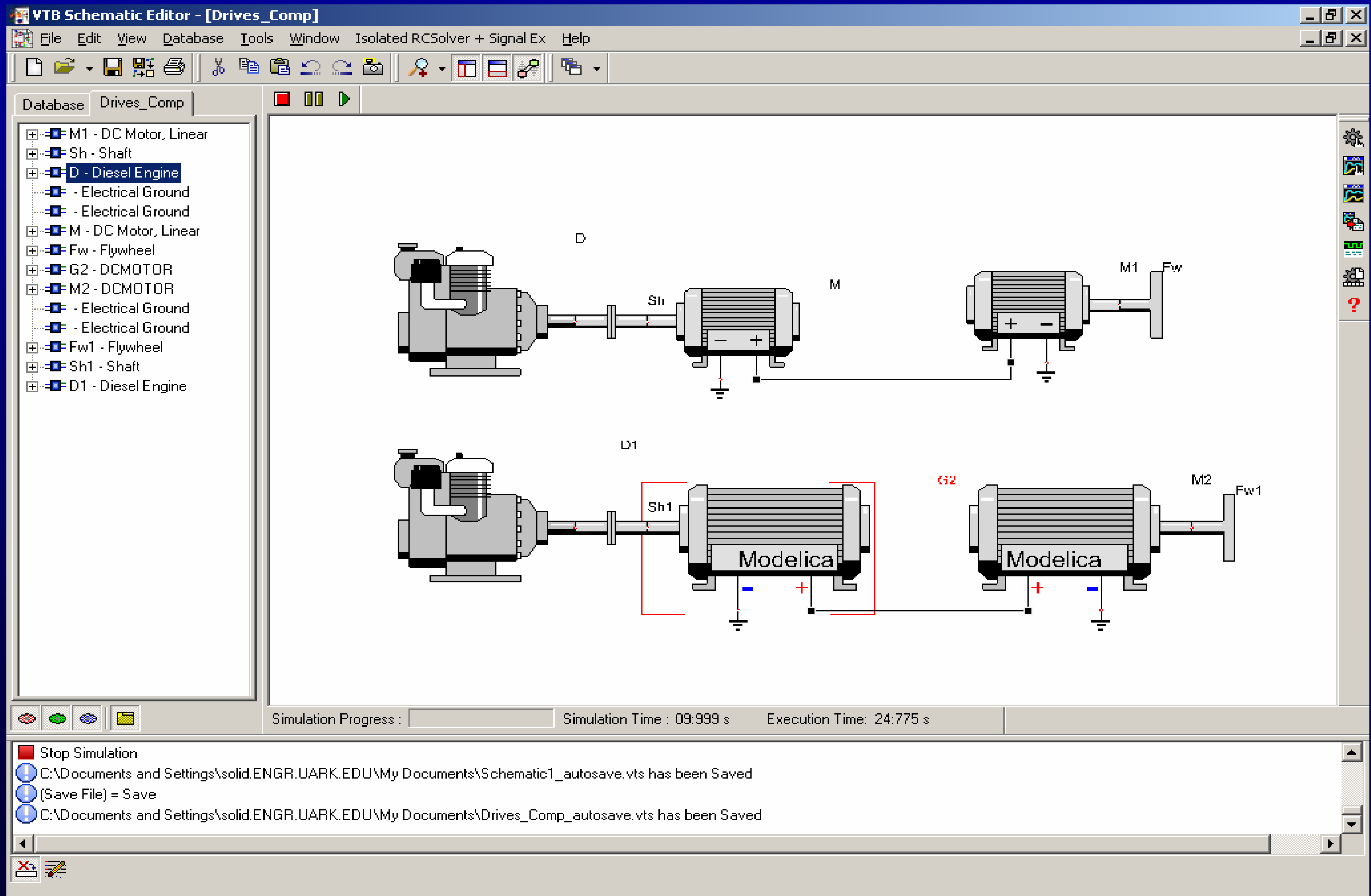
```
<?xml version="1.0" ?>
        <model name="DCMotor" version="1.1">
         <comment value="DC Motor generated by
             MultiTranslator" />
        <interface>
        <parameter default="125.664" default_enabled=""
             name="w0" nature="real" process_parameter=""
             unit="">
        <comment value="Rated angular speed" />
        </parameter>
        . . .
        <port mode="inout" name="node1" nature="electrical"
             type="terminal">
        <comment value="" />
        <terminal number="0" />
        </port>
         . . .
        </interface>
        <body name="Architecture:Ideal">
          <internal_node name="0_mas" type="mechanical" />
        <branch from="node1" name="branch1" to="node2">
          <quantity default="" name="i" nature="through"
             type="current" unit="ampere" />
          <quantity default="" name="V" nature="across"
             type="voltage" unit="volt" />
        <equation string="i=integ(V-R*i-V0/w0*w)/L">
        . . .
        </equation>
          </branch>
        . . .
        </topology>
        </body>
        </model>
```

XML representation

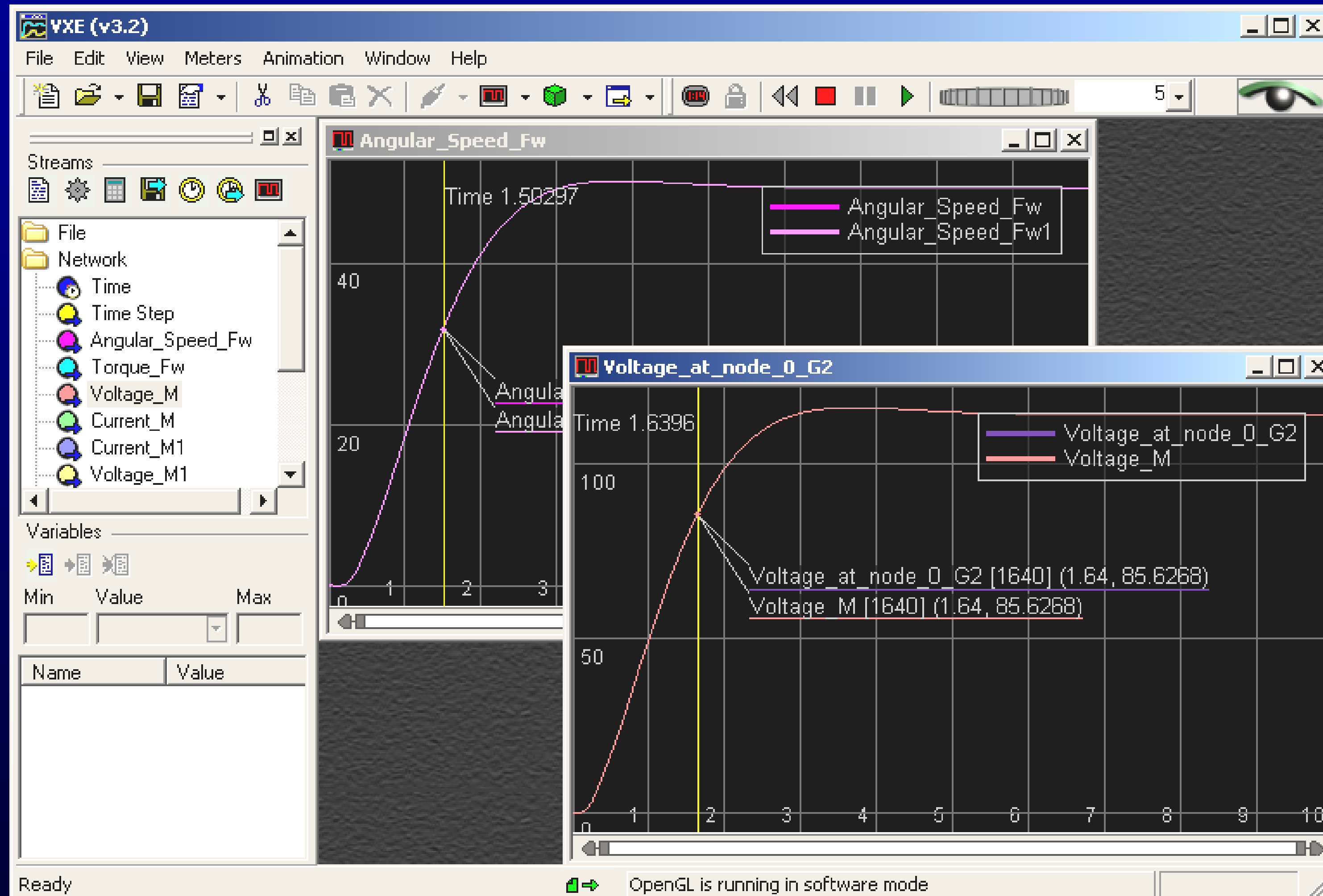
```
! UDD input Model file of DCMotor generated by Paragon
! This is a machine generated code.
! Generated on Wed, 13 Apr 2005 11:32:03 AM
Name: DCMotor
Nodes: 3
Terminals: 3
USE Model DCMotor0
Model DCMotor0
{
Tq=(((-(J*(diff(w))))+((V0/w0)*i0))-(b*w))
pari=((INTEG(((v0-v1)-(R*i0))-((V0/w0)*w)))/L)
pari
-pari
Tq
}
```

UDD model file generated by Paragon

Results (Drive System)



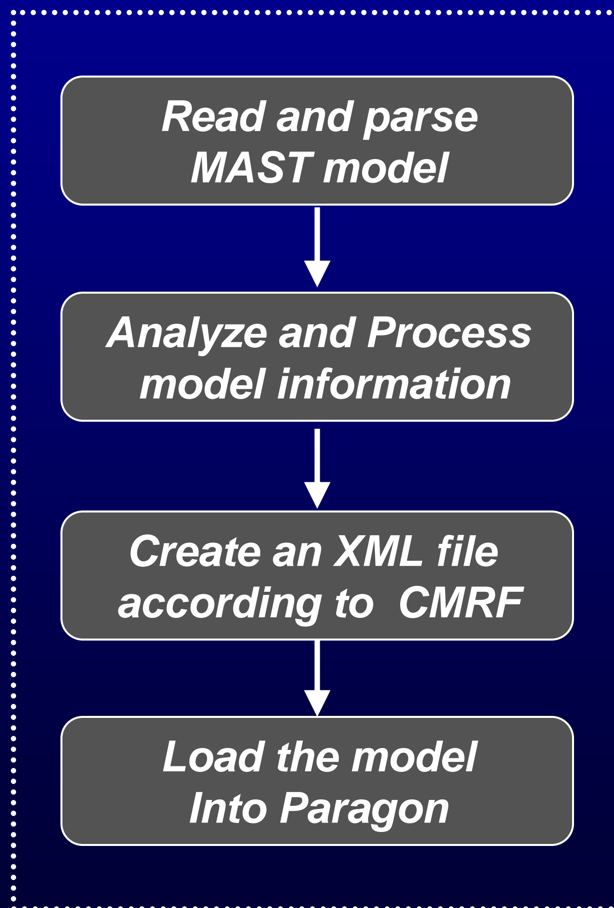
Results (Drive System)

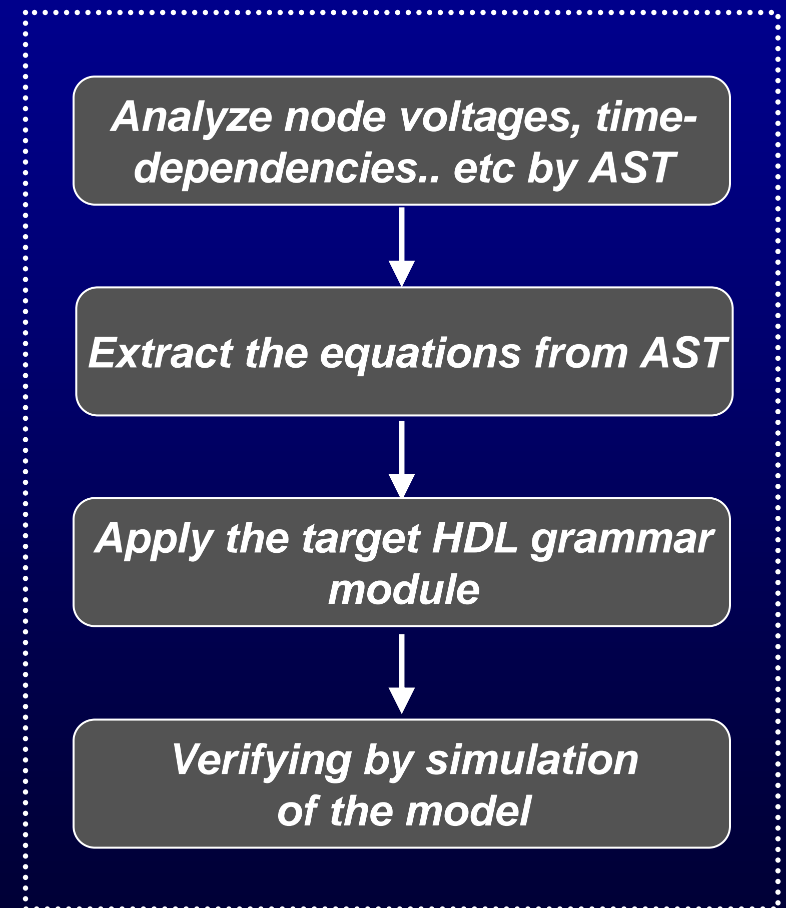


Paragon with other features

- MAST Importer
- VHDL-AMS, Verilog-A and MAST code generators
- Industry standard semiconductor device models like BSIMSOI, EKV were implemented
- Certify – Model Parameter Extraction tool
- Integrated with Model Compilers to generate SPICE code

MAST Reading Technology





MAST Importer (Diode)

MAST

```
template diode p n = model, ic
electrical p, n
struc { number n=1, is = 10f, tau = 0, cjo = 0, m = .5, fc = .5, vj = 1,
                rs = 0
} model = ()

external number temp
number ic= undef
{
<consts.sin
electrical pi
number wn vt, fcvj
val q qdiode
val i id
val v vd,vpn
parameters {
                vt = math_boltz*(temp+math_ctok)/math_charge
                fcvj = model->fc * model->vj
                if (model->n<=0)   wn = .1
                else wn = model->n
}
values {
vpn = v(p) - v(n)
vd = v(pi) - v(n)
id = model->is*(limexp(vd/(wn*vt)) - 1)
if (vd <fcvj)
{
qdiode = model->tau*id + model->vj*model->cjo*(1-(1-vd/model-
>vj)**(1-model->m))/(1-model->m)

}
```

VHDL-AMS

```
library IEEE;
use IEEE.math_real.all;
package diode_package is
subtype model_vj is real range 1.0 to real'high;
subtype model_cjo is real range 0.0 to real'high;
subtype model_is is real range 10.0e-15 to real'high;
subtype model_rs is real range 0.0 to real'high;
subtype model_tau is real range 0.0 to real'high;
subtype model_fc is real range 0.5 to real'high;
subtype model_m is real range 0.5 to real'high;
subtype model_n is real range 1.0 to real'high;
end package diode_package;

library IEEE;
use IEEE.math_real.all;
use IEEE.electrical_systems.all;
use work.diode_package.all;

entity diode is
generic (model_vj:work.complex_diode_package.model_vj:=1.0;
 temp:real;
 model_is:work.complex_diode_package.model_is:=10.0e-15;
 model_rs:work.complex_diode_package.model_rs:=0.0;
 model_n:work.complex_diode_package.model_n:=1.0;
 model_cjo:work.complex_diode_package.model_cjo:=0.0;
 ic:real:=real'low;
 model_fc:work.complex_diode_package.model_fc:=0.5;
 model_m:work.complex_diode_package.model_m:=0.5;
 model_tau:work.complex_diode_package.model_tau:=0.0);
port (terminal p: electrical; terminal n: electrical);
end entity diode;
```

MAST Importer (Diode)

MAST

```
else {
qdiode = model->tau*id +model->cjo*model->vj*(1-(1-model->fc)**(1-model->m))/(1-model->m) + (model->cjo/(1-model->fc)**(1+model->m))*(((1-model->fc*(1+model->m)))*(vd-fcvj)+model->m/(2*model->vj)*(vd**2 -fcvj**2))
}
}
control_section {
if (model->rs == 0) collapse (p,pi)
newton_step (vd,[0,0.1,2,0])
small_signal (rc,resistance,"series resistance",model->rs)
ss_partial(g_d,id,vd)
small_signal(req,resistance,"pi-n resistance, if g_d ~=0 then 1/g_d else inf)
small_signal(cj,capacitance,"pi-n capacitance",qdiode,vd)
initial_condition(vd,ic)
}
equations {
        i(pi->n) += id +d_by_dt(qdiode)
        if (model->rs ~= 0) i(p->pi) += v(p,pi)/model->rs
}
}
```

VHDL-AMS

```
architecture Arch1 of diode is
terminal pi: electrical;
--This function is used to avoid convergence problems
due to numerical overflow –
function limit_exp( x : real ) return real is
   variable abs_x : real := abs(x);
   variable result : real;
begin
  if abs_x < 100.0 then
    result:= exp(abs_x);
  else
    result:= exp(100.0)*(abs_x-99.0);
  end if;
  if x < 0.0 then
  result:= 1.0/result;
  end if;
  return result;
end function limit_exp;

function function_parameters(temp:real;
model_fc:work.complex_diode_package.model_fc;
model_vj:work.complex_diode_package.model_vj;
model_n:work.complex_diode_package.model_n) return
real_vector is
variable vt:real;
variable fcvj:real;
variable wn:real;
variable parameters_list:real_vector(0 to 2);
begin
vt:=(1.380658e-023*(temp+0.0))/1.60217733e-019;
fcvj:=model_fc*model_vj;
```

VHDL-AMS equivalent of *limexp* function in MAST

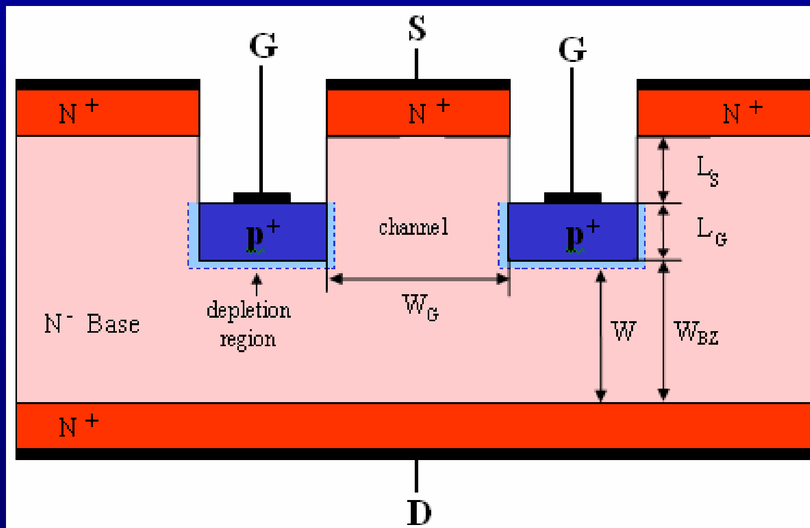
MAST Importer (Diode)

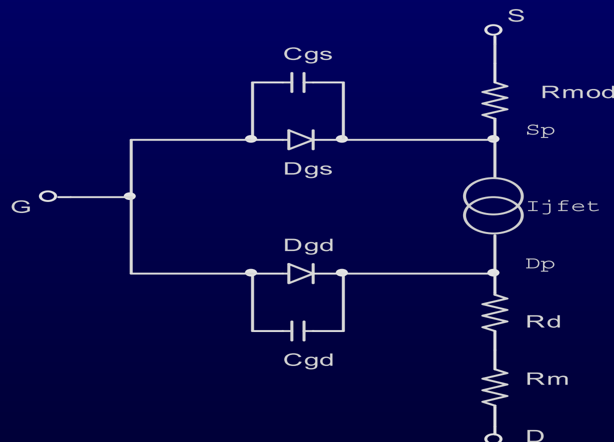
VHDL-AMS

```
if (model_n<0.0) then
wn:=0.1;
else
wn:=model_n;
end if;
parameters_list :=(vt,fcvj,wn);
return parameters_list;
end function function_parameters;
constant parameters_list:real_vector(0 to
2):=function_parameters( temp,model_fc,model_vj,model_n);
constant wn:real:= parameters_list(2);
constant vt:real:= parameters_list(0);
constant fcvj:real:= parameters_list(1);
----------- Branch Variable Declarations ---------
quantity vpn across p to n;
quantity newQty0 through pi to n;
quantity vd across pi to n;
quantity newQty2 through p to pi;
quantity newQty1 across p to pi;
------- Time-varying Quantities Declarations-------
quantity values_list:real_vector(0 to 1);
quantity new_variable1:real:=1.0;
function function_values(model_is:
work.complex_diode_package.model_is;
vd : real; wn : real; vt : real; fcvj : real;
model_tau : work.complex_diode_package.model_tau;
model_vj : work.complex_diode_package.model_vj;
model_cjo:work.complex_diode_package.model_cjo;
model_m:work.complex_diode_package.model_m;
```

```
model_fc:work.complex_diode_package.model_fc) return
real_vector is
variable id:real;
variable qdiode:real;
variable values_list:real_vector(0 to 1);
begin
id:=(model_is*((limit_exp(vd/(wn*vt)))-1.0));
if (vd<fcvj) then
qdiode:=((model_tau*id)+(((model_vj*model_cjo)*(1.0-((1.0-
(vd/model_vj))**(1.0-model_m))))/(1.0-model_m)));
else
qdiode:=(((model_tau*id)+(((model_cjo*model_vj)*(1.0-((1.0-
model_fc)**(1.0-model_m))))/(1.0-
model_m)))+(((model_cjo/(1.0-
model_fc))**(1.0+model_m))*((((1.0-
(model_fc*(1.0+model_m))))*(vd-
fcvj))+((model_m/(2.0*model_vj))*((vd**2.0)-(fcvj**2.0))))));
end if;
values_list :=(id,qdiode);
return values_list;
end function function_values;
begin
-------------- Simultaneous Equations -------------
values_list==function_values(model_is,vd,wn,vt,model_tau,mod
el_vj, model_cjo,model_m,model_fc,fcvj);
newQty0==values_list(0)+new_variable1'dot;
newQty2==newQty1/model_rs;
new_variable1==values_list(1);
end architecture Arch1;
```

MAST model (SiC JFET/SIT)






```
###### SiC SIT MAST MODEL######
###### Copyright (C) 2005 - MSCAD Lab - University of Arkansas
######------------------######
template jfet_dc d g s  =
vp,As,Ag,temp,Nb,Isgs,Isgd,rs,ngs,ngd,alpha,rmod,tau,wbz,fcsj,io,lambda,cap

electrical d,g,s                          #node type
#Model parameters (Temperature Dependant)
#      Room temperature value (300K)
number vp   =  6.0                        # Pinch-off Voltage
number As   =  0.01                       # Source Area /cm^2(Physical measurement)
number Ag   =  0.01                       # Gate Area /cm^2(Physical measurement)
number temp =  373                        # Temperature
number Nb   =  2e15                       # Base Doping Concentration
………………
### TEMPLATE BODY ###
{
#Initialize constants
<consts.sin
number muln = 947          # Electron Mobility Multiplier cm**2/Vs for 4H-SiC
number mulp = 108.1        # Hole Mobility Multiplier cm**2/Vs for 4H-SiC
…………….
#Declaring values
val mu mun                                #(cm**2/v/sec) electron mobility
val mu mup                                #(cm**2/v/sec) hole mobility
val pcm3 ni                               #(1/cm**3) intrinsic carrier concentration
…………….
values{
vgdp = v(g) - v(dp)
vgsp = v(g) - v(sp)
………………
#Current calculations:
c1  = (0.0031*(vgs**4))+(0.0279*(vgs**3))+(0.0769*(vgs**2))+(0.0521*vgs)+0.05
alph= (0.08-((c1*limexp(e1*vdpsp))))/(1.5*temp/300)
…………………
## Equations Section
equations {
i(d->dp)        += iddp               # Current from drain to internal node dp
i(dp->sp)       += idpsp              # Current flow within internal nodes-dp to sp
i(sp->s)        += irmod              # Current flow from internal node sp to source
i(g->dp)        += igdp + d_by_dt(qcgd) # Current from gate to internal node dp
i(g->sp)        += igsp + d_by_dt(qcgs) + d_by_dt(qcgs2)  # Current from gate to
internal node sp
                                          }
}
```

Imported MAST (JFET/SIT) model

ModLyng - Model Creator (jfet_dc)

File Edit Editors Tools Generate Import Window Help

Open Models
jfet_dc
Architectures
Arch1

General Model Information
Model Name: jfet_dc
Model Connection Points
d
g
s
Model Parameters
Ag
alpha
As
cap
fcsj
io
Isgd

Topology Editor

s, vgs+, +Cgs, vsps, irmod, +vgsp, newQty1, sp, g, vdpsp, idpsp, vgdp, dp, newQty0, Cgd, vddp, iddp, d

Equation Editor

Model Expression | Branch

```
vbi=2.8
epsilon=8.85e-14
episic=9.7
mulp=108.1
muln=947
Function function_values(vgs,temp,vdpsp,io,vgsp,vp,lambda,wbz,episic,vg
c1=(0.0031*(vgs^4))+(0.0279*(vgs^3))+(0.0769*(vgs^2))+(0.0521*vgs)+0.05
e1=((-0.0053*(vgs^4))-(0.0535*(vgs^3))-(0.1984*(vgs^2))-(0.3502*vgs)-0.
alph=(0.08-((c1*limexp(e1*vdpsp))))/(1.5*temp/300)
p1=1+0.05*(vgs)
idpsp=0
If (((vdpsp)>(0)))
        idpsp=io*(1+tanh((p1)*(vgsp-vp)))*tanh(alph*vdpsp)*limexp(lambd
Endif
If (((idpsp)<=(0)))
        idpsp=0
Endif
vt=1.380658e-23*temp/1.60217733e-19
wgd=sqrt(abs(2*episic*(vgdp+vbi)/1.60217733e-19*Nb))
wb=wbz-wgd
If (((wgd)>=(wbz)))
        wb=1e-3*wbz
Endif
wgs=sqrt(abs(2*episic*(vgsp+vbi)/1.60217733e-19*Nb))
If (((vgdp)>(0)))
```

Generated VHDL-AMS model

ModLyng - Model Creator (jfet_dc)

File Edit Editors Tools Generate Import Window Help

- Open Models
  - jfet_dc
    - Architectures
      - Arch1

Topology Editor

s, vgs+, +Cgs, vsps, irmod, +vgsp, newQty1, sp, g, vdpsp, idpsp, vgdp, dp, newQty0, vddp, iddp, Cgd, d

- General Model Information
  - Model Name: jfet_dc
- Model Connection Points
  - d
  - g
  - s
- Model Parameters
  - Ag
  - alpha
  - As
  - cap
  - fcsj
  - io
  - Isgd
  - Isgs
  - lambda

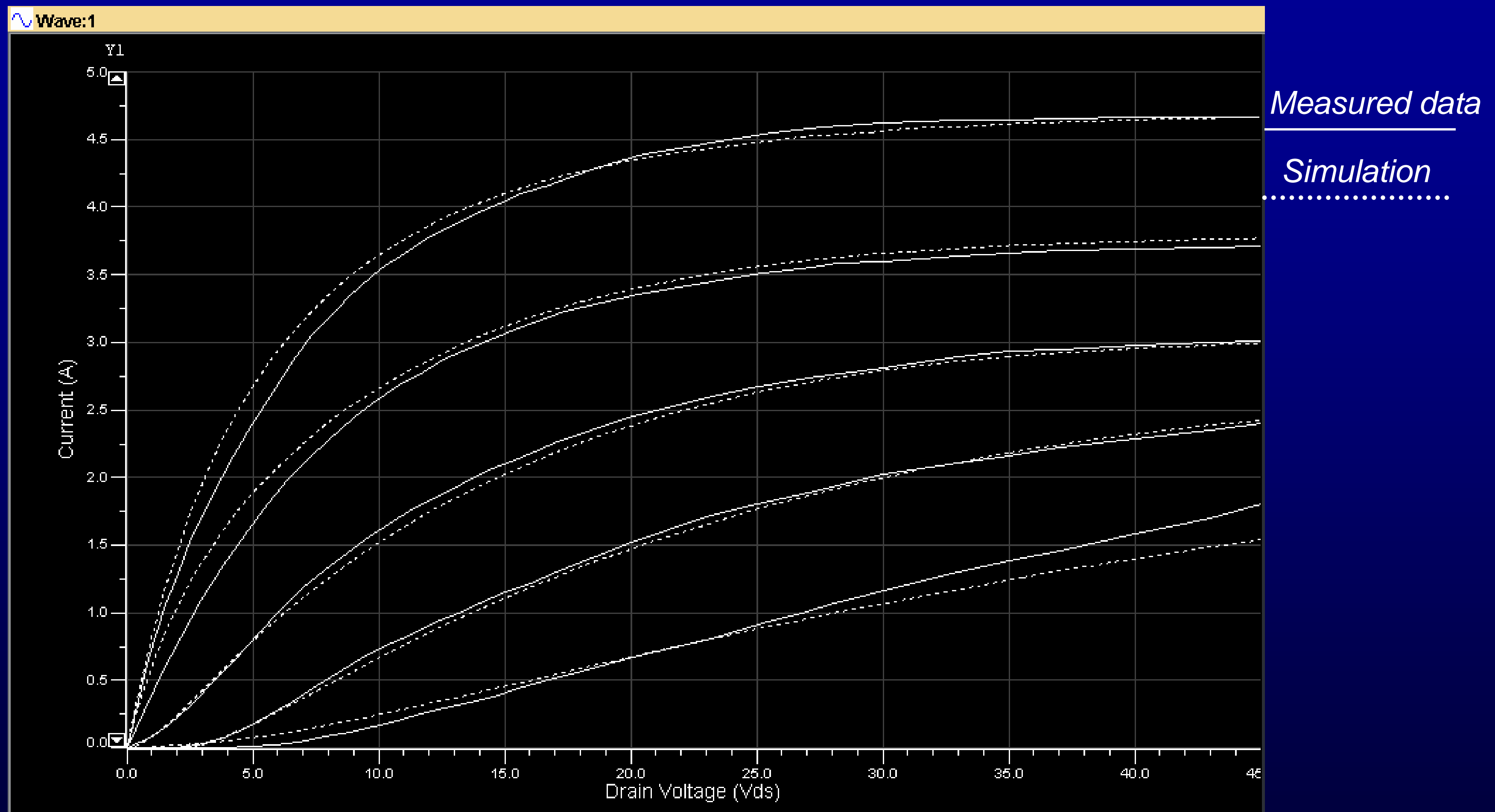
jfet_dc.vhd + (C:\Documents and Se...\Desktop\jfet_dc\output_vhdl) - GVIM1

File Edit Tools Syntax Buffers Window cppcomplete Plugin CVS Python Help

```
-- VHDL-AMS Model of jfet_dc generated by ModLyng
-- This is a machine generated code.
-- Generated on Sat, 17 Sep 2005 06:38:01 PM
library IEEE;
use IEEE.math_real.all;
use IEEE.electrical_systems.all;
use work.jfet_dc_package.all;
------ Model Interface and Parameters Declaration ------
entity jfet_dc is
generic (ngd:real:=2.0;
 tau:real:=0.0;
 Ag:real:=0.01;
 temp:real:=373.0;
 rs:real:=0.5;
 Nb:real:=2.0e15;
 cap:real:=0.0;
 wbz:real:=10.0e-4;
 fcsj:real:=1.0;
 Isgd:real:=1.0e-30;
 vp:real:=6.0;
 As:real:=0.01;
 io:real:=4.5e7;
 ngs:real:=2.0;
 alpha:real:=2.15;
 rmod:real:=0.5;
 Isgs:real:=1.0e-30;
 lambda:real:=0.0015);
port (terminal s: electrical; terminal d: electrical; terminal g: electrical);
end entity jfet_dc;

architecture Arch1 of jfet_dc is
terminal sp: electrical;
terminal dp: electrical;
-- This function is used to avoid convergence problems due to numerical overflow --
function limit_exp( x : real ) return real is
    variable abs_x : real := abs(x);
    variable result : real;
jfet_dc.vhd [+]                                     21,16          Top
-- INSERT --
```

Simulation Results (25 ^{o}C)



Measured data and Simulation results for 25 ^{o}C

Conclusions

- Benefits of the standard open-source XML format
- Prevents models becoming obsolete
- Modelica model was imported and then exported to the VTB environment through Paragon
- Modeling methodology described here enables easy implementation of compact semiconductor device models