Research in Model-Based Product Development at PELAB in the MODPROD Center

Presentation at MODPROD'2015
PELAB – Programming Environment Laboratory
Department of Computer and Information Science
Linköping University
2015-02-03

Peter Fritzson
Examples of Complex Systems in Engineering

- Robotics
- Automotive
- Aircraft
- Mobile Phone Systems
- Business Software
- Power plants
- Heavy Vehicles
- Process industry
Architecture of Integrated Model-Based Product Development

Open Standards – Modelica (HW, SW) and UML (SW)
Open Source Modelica Consortium

Open Source Effort
The OpenModelica Open Source Environment
www.openmodelica.org

- Advanced Interactive Modelica compiler (OMC)
  - Supports most of the Modelica Language
  - Modelica and Python scripting
- Basic environment for creating models
  - OMShell – an interactive command handler
  - OMNotebook – a literate programming notebook
  - MDT – an advanced textual environment in Eclipse

- OMEdit graphic Editor
- OMDebugger for equations
- OMOptim optimization tool
- OM Dynamic optimizer collocation
- ModelicaML UML Profile
- MetaModelica extension
- ParModelica extension

The OpenModelica Open Source Environment www.openmodelica.org
OSMC – International Consortium for Open Source Model-based Development Tools, 46 members Feb 2014

Founded Dec 4, 2007

Open-source community services
  - Website and Support Forum
  - Version-controlled source base
  - Bug database
  - Development courses
  - www.openmodelica.org

Code Statistics

Industrial members
  - Bosch Rexroth AG, Germany
  - Siemens PLM, California, USA
  - Siemens Turbo, Sweden
  - CDAC Centre, Kerala, India
  - Creative Connections, Prague
  - DHI, Aarhus, Denmark
  - EDF, Paris, France
  - Equa Simulation AB, Sweden
  - Fraunhofer IWES, Bremerhaven
  - Fraunhofer FIRST, Berlin
  - Frontway AB, Sweden
  - IFP, Paris, France

University members
  - Linköping University, Sweden
  - UC Berkeley, USA
  - TU Berlin, Insti UEBB, Germany
  - FH Bielefeld, Bielefeld, Germany
  - TU Braunschweig, Germany
  - Univ Calabria, Italy
  - Danish Technical Univ, Denmark
  - TU Dortmund, Germany
  - TU Dresden, Germany
  - Université Laval, Canada
  - Georgia InstiTechnology, USA
  - Ghent University, Belgium

- GTI, USA
- ISID Dentsu, Tokyo, Japan
- ITI, Dresden, Germany
- Maplesoft, Canada
- Rickardo Inc., USA
- STEAG, Dehli, India
- TLK Thermo, Germany
- Sozhou Tongyuan, China
- VTI, Linkköping, Sweden
- VTT, Finland
- Wolfram MathCore, Sweden
OpenModelica Environment
General Tool Interoperability & Model Exchange Functional Mock-up Interface (FMI)

The FMI development is result of the MODELISAR 29-partner project
- FMI development initiated by Daimler
- Improved Software/Model/Hardware-in-the-Loop Simulation, of physical models and of AUTOSAR controller models from different vendors for automotive applications with different levels of detail.
- Open Standard
- 14 automotive use cases for evaluation
- > 50 tool vendors are supporting it

Engine with ECU  Gearbox with ECU  Thermal systems  Automated cargo door  Chassis components, roadway, ECU (e.g. ESP)

functional mockup interface for model exchange and tool coupling
courtesy Daimler
FMI in OpenModelica

- Model Exchange implemented (FMI 1.0 and FMI 2.0)
- FMI 2.0 Co-simulation is under development. A prototype of FMI 2.0 including tool co-simulation is available
- Ongoing work to support full FMI 2.0
Two OpenModelica Technical Milestones Achieved 2014

• Update to a **new** OpenModelica Compiler frontend flattening architecture
  • Achieved in OM 1.9.1 release, October 2015
  • Needed to handle some difficult cases of flattening, e.g. for the Fluid library
  • Work on this during approximately 2 years

• **Deployment** of **bootstrapped** OpenModelica Compiler
  • Achieved in November 2014. All development switched to this compiler
  • Bootstrapping = OMC compiler used to develop and compile itself
  • Advantages in terms of better programmability, maintenance, debugging and performance analysis, modularity and current/future performance increases
  • OpenModelica 1.9.2, February 2015, first release based on this platform
  • Work on this during approximately 9 years, 2005 to 2014
OpenModelica Bootstrapped Compiler Platform

- First Modelica compiler to model and compile itself!
- More efficient simulation!
- More expressive modeling language
- Enabling better debugging and performance analyzer

- Potential Platform portability
  - Now compiling to C Code
  - Have Java prototype code generator – could move compiler to run under JVM
  - Have C# prototype code generator – could move compiler to run under C# run-time
Research

Modeling-Language Design

Model-based Optimization

Model-Based Fault Analysis

Multi-Core based Simulation

Model Debugging

Modeling Support Environments
Modeling Language and Tool Research

• How can a **modeling language be designed** with **precise semantics** to avoid errors?
• Can the language be made **extensible**?
• Can it model itself (**meta-modeling**)?

• How should a user-supportive modeling **environment** be designed?

• **Current/recent work:**
  • MetaModeling in Modelica
  • Fault analysis
  • Bootstrapping
Research on Verified Code Generation from Subset Modelica

Special Tool Chain

Bernhard Thiele (talk later at MODPROD)
ModelicaML
UML Profile for Modelica

Verification of Requirements

Talk later at MODPROD by Wladimir Schamai

Automating Model Composition
for Design Verification – Work MODRIO Project
**Example: Simulation and Requirements Evaluation**

- Req. 001 is instantiated 2 times (there are 2 tanks in the system).
- Tank height is 0.6m.
- Req. 001 for the tank 2 is violated.
- Req. 001 for the tank 1 is not violated.
vVDR – virtual Verification of Design Requirements
vVDR Roles and Tasks (Wladimir Schamai PhD thesis Nov 2013)

- 1. **Select requirements** to be **verified** using simulations (or logic tools)
- 2. **Formalize** textual requirements

- Support Requirements Analyst in selecting requirements
- 3. **Create system design models** to be verified against requirements
- Support System Tester in deciding which requirements are to be verified using which test cases
- Support System Tester in linking requirement properties to design model properties

- 4. Create **test modes** including **test cases** and decide which requirement are to be verified using which test cases
- 5. **Link requirements** to design models
- 6. **Simulate and observe** requirement violations
- 7. **Analyze** simulation **results** and create a Simulation **Summary Report**
Parallel Execution
Compilation to MultiCore
Integrating Parallelism and Mathematical Models
Three Approaches

• **Automatic Parallelization of Mathematical Models**
  - Parallelism over the numeric solver method.
  - Parallelism over time.
  - **Parallelism over the model equation system**
    - ... with fine-grained task scheduling

• **Coarse-Grained Explicit Parallelization Using Components**
  - The programmer partitions the application into computational components using strongly-typed communication interfaces.
    - Co-Simulation
    - Transmission-Line Modeling (TLM)

• **Explicit Parallel Programming**
  - Providing general, easy-to-use explicit parallel programming constructs within the *algorithmic* part of the modeling language.
    - ParModelica, NestStepModelica
Automatic Scheduling and Task Clustering
Efficient parallel execution of code from models


Original task system of Four Bit Binary Adder model

1122 Tasks 1360 Edges

Task system after clustering for level scheduler

18 Tasks 72 Edges
function parvar
  Integer m = 1024;
  Integer A[m];
  Integer B[m];
  parglobal Integer pm;
  parglobal Integer pn;
  parglobal Integer pA[m];
  parglobal Integer pB[m];
  parlocal Integer ps;
  parlocal Integer pSS[10];
end parvar;

//copy to device
B := A;
pA := A;

//copy from device
B := pA;
pB := pA;

//copy device to device
pm := m;
n := pm;
pn := pm;
end parvar;

Modelica + OpenCL = ParModelica

Memory Regions | Accessible by
Global Memory   | All work-items in all work-groups
Constant Memory | All work-items in all work-groups
Local Memory    | All work-items in a work-group
Private Memory  | Private to a work-item
ParModelica Matrix Multiplication using *Kernel function*

Gained speedup

- Intel Xeon E5520 CPU (16 cores) 26
- NVIDIA Fermi-Tesla M2050 GPU (448 cores) 115

Speedup comparison to sequential algorithm on Intel Xeon E5520 CPU

### Speedup

<table>
<thead>
<tr>
<th>Parameter M (Matrix sizes MxM)</th>
<th>Speedup (GPU M2050)</th>
</tr>
</thead>
<tbody>
<tr>
<td>64</td>
<td>0.0625</td>
</tr>
<tr>
<td>128</td>
<td>0.125</td>
</tr>
<tr>
<td>256</td>
<td>0.25</td>
</tr>
<tr>
<td>512</td>
<td>0.5</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameter M (Matrix sizes MxM)</th>
<th>Speedup (CPU E5520) (Parallel)</th>
</tr>
</thead>
<tbody>
<tr>
<td>64</td>
<td>0.137</td>
</tr>
<tr>
<td>128</td>
<td>0.17</td>
</tr>
<tr>
<td>256</td>
<td>0.438</td>
</tr>
<tr>
<td>512</td>
<td>2.36</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameter M (Matrix sizes MxM)</th>
<th>Speedup (CPU E5520) (Serial)</th>
</tr>
</thead>
<tbody>
<tr>
<td>64</td>
<td>0.093</td>
</tr>
<tr>
<td>128</td>
<td>0.741</td>
</tr>
<tr>
<td>256</td>
<td>5.875</td>
</tr>
<tr>
<td>512</td>
<td>58.426</td>
</tr>
</tbody>
</table>

Simulation Time (second)

- CPU E5520 (Serial) 0.093 0.741 5.875 58.426 465.234
- CPU E5520 (Parallel) 0.137 0.17 0.438 2.36 17.66
- GPU M2050 (Parallel) 1.215 1.217 1.274 1.625 4.057
New Scalable OpenModelica Parallel Code Generator

- Ongoing work
- Both **task** parallelism and **data** parallelism
- Handling non-expanded **arrays** efficiently
- Includes use of **TLM-partitioning** for more parallelism

- OPENMP or Pthreads based parallelization prototype for equation-based models in OpenModelica

- ParModelica - Generating **OpenCL** code for platform independence
- **Template** based code generator
Model Debugging
and Performance Analysis
Equation-Based Model Dynamic Debugging

Martin Sjölund

martin.sjolund@liu.se
Need for Debugging Tools
Map Low vs High Abstraction Level

• A **major part** of the total **cost** of software projects is due to testing and debugging

• US-Study 2002:
Software errors cost the US economy **annually ~ 60 Billion $**

• **Problem:** Large Gap in Abstraction Level from **Equations** to **Executable Code**

• Example error message (hard to understand)

  Error solving nonlinear system 132
  
  time = 0.002
  residual[0] = 0.288956
  x[0] = 1.105149
  residual[1] = 17.000400
  x[1] = 1.248448
  ...

...
Model Compiler Translation Phases Extended with Debugging

- Include debugging support within the translation process

<table>
<thead>
<tr>
<th>Debugging Translation Process Additional Steps</th>
<th>Normal Translation Process</th>
</tr>
</thead>
<tbody>
<tr>
<td>Save element position</td>
<td>Modelica Source Code</td>
</tr>
<tr>
<td>Save element origin (model and position)</td>
<td>Modelica model</td>
</tr>
<tr>
<td>Save equation elements origin (model and position)</td>
<td>Flat Model</td>
</tr>
<tr>
<td>Save the optimizer transformations changes</td>
<td>Sorted equations</td>
</tr>
<tr>
<td>Save all the available origin information</td>
<td>Optimized sorted equations</td>
</tr>
<tr>
<td>Executable with all the available origin information</td>
<td>C Code</td>
</tr>
<tr>
<td>Simulation with run-time debugging functionality</td>
<td>Executable</td>
</tr>
</tbody>
</table>

Simulation
Example Symbolic Transformations with Compiler Debug Trace

- Complicated to understand source of some errors
- Efficient trace of transformations – low overhead

Example: 0 = y + der(x * time * z);    z = 1.0;

(1) substitution:
  y + der(x * (time * z))
  =>
  y + der(x * (time * 1.0))

(2) simplify:
  y + der(x * (time * 1.0))
  =>
  y + der(x * time)

(3) expand derivative (symbolic diff):
  y + der(x * time)
  =>
  y + (x + der(x) * time)

(4) solve:
  0.0 = y + (x + der(x) * time)
  =>
  der(x) = ((-y) - x) / time
Properties of Transformation Trace

- Most equations have very **few** transformations on them
- Most of the interesting equations have a few
  - Still rather readable
- Some extra care to handle Modelica variable aliasing
- Very **efficient** implementation, max 1% overhead

### MSL 3.1 MultiBody DoublePendulum

<table>
<thead>
<tr>
<th># Ops</th>
<th>Frequency</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>457</td>
<td>Parameters</td>
</tr>
<tr>
<td>1</td>
<td>89</td>
<td>Dummy eq &amp; know var</td>
</tr>
<tr>
<td>2</td>
<td>720</td>
<td>Alias vars</td>
</tr>
<tr>
<td>3</td>
<td>479</td>
<td>Alias vars</td>
</tr>
<tr>
<td>4</td>
<td>124</td>
<td>Alias after simplify</td>
</tr>
<tr>
<td>5</td>
<td>25</td>
<td>Alias after simplify</td>
</tr>
<tr>
<td>6</td>
<td>99</td>
<td>Alias after simplify</td>
</tr>
<tr>
<td>7</td>
<td>55</td>
<td>Scalar eq</td>
</tr>
<tr>
<td>8</td>
<td>37</td>
<td>...</td>
</tr>
<tr>
<td>9</td>
<td>110</td>
<td>...</td>
</tr>
<tr>
<td>10</td>
<td>72</td>
<td>...</td>
</tr>
<tr>
<td>11</td>
<td>12</td>
<td>...</td>
</tr>
<tr>
<td>12</td>
<td>25</td>
<td>...</td>
</tr>
<tr>
<td>13</td>
<td>35</td>
<td>...</td>
</tr>
<tr>
<td>14</td>
<td>3</td>
<td>Known constant after many replacements</td>
</tr>
<tr>
<td>21</td>
<td>27</td>
<td>World object (3x3 matrix with many occurrences of aliased vars)</td>
</tr>
</tbody>
</table>
Integrated Static-Dynamic OpenModelica Equation Model Debugger

Efficient handling of Large Equation Systems

Showing equation transformations of a model:

Mapping dynamic run-time error to source model position
Example – Detecting Source of Chattering (excessive event switching) causing bad performance

equation
\[ z = \text{if } x > 0 \text{ then } -1 \text{ else } 1; \]
\[ y = 2 \cdot z; \]
Error Indication – Simulation Slows Down


Cancel Simulation

OMEdit - Debugging.Chattering.ChatteringEvents1 Simulation Output

stdout | info | Chattering detected around time
0.50000005..0.500000995001 (100 state events in a row with a total time delta less than the step size 0.002). This can be a performance bottleneck. Use -lv LOG_EVENTS for more information. The zero-crossing was: x > 0.0 Debug more
Transformations Browser – EngineV6 Overview
(11 116 equations in model)
Browsing Equation Transformation Chains
Closeup of EngineV6 Equations

<table>
<thead>
<tr>
<th>Defines</th>
<th>Depends</th>
</tr>
</thead>
<tbody>
<tr>
<td>Variable</td>
<td>Variable</td>
</tr>
<tr>
<td></td>
<td>- cylinder3.Rod.body.w_a[1]</td>
</tr>
</tbody>
</table>

**Equation Operations**

- substitute: {{cylinder3.B2.e[1] * cylinder3.B2.e[1] + (1.0 - cy...2.phi), 0.0 * 0.0 + (1.0 - 0.0 * 0.0) * cos(cylinder3.B2.phi)}}
- original: R_rel = Frames.planarRotation(e, phi_offset + phi, w); => flattened:
### Eqs Transformations Browser – Nonlinear System

<table>
<thead>
<tr>
<th>Ind</th>
<th>Type</th>
<th>Equation</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>regular</td>
<td>(assignment) p1 = dp_valve + p2</td>
</tr>
<tr>
<td>17</td>
<td>regular</td>
<td>(assignment) sqrt_dp = DIVISION(dp_valve, (dp_valve ^ 2.0 + ...))</td>
</tr>
<tr>
<td>18</td>
<td>regular</td>
<td>(assignment) w_pump = Kv * sqrt_dp</td>
</tr>
<tr>
<td>19</td>
<td>regular</td>
<td>(assignment) dp_pump = p1 - patm</td>
</tr>
<tr>
<td>20</td>
<td>regular</td>
<td>(residual, dp_pump + a1 * w_pump ^ 2.0 - dp0)</td>
</tr>
<tr>
<td>21</td>
<td>regular</td>
<td>(nonlinear)</td>
</tr>
</tbody>
</table>

- **Regular**: (assignment) p1 = dp_valve + p2
- **Regular**: sqrt_dp = DIVISION(dp_valve, (dp_valve ^ 2.0 + ...))
- **Regular**: w_pump = Kv * sqrt_dp
- **Regular**: dp_pump = p1 - patm
- **Regular**: dp_pump + a1 * w_pump ^ 2.0 - dp0

**Parameter**: t1 = 1.0
**Parameter**: h0 = cp * (T0 - Tref)
**Statement**: assert(Tref >= 0.0, "Variable Tref out of [min, max] interval: Tref >= 0.0 has value: "+ String(Tref, "g"));
**Statement**: assert(rho >= 0.0, "Variable rho out of [min, max] interval: rho >= 0.0 has value: "+ String(rho, "g"));
**Statement**: assert(T0 >= 0.0, "Variable T0 out of [min, max] interval: T0 >= 0.0 has value: "+ String(T0, "g"));
**Statement**: assert(T1 >= 0.0, "Variable T1 out of [min, max] interval: T1 >= 0.0 has value: "+ String(T1, "g"));
Performance Profiling
(Here: Profiling all equations in MSL 3.2.1 DoublePendulum)

- Measuring performance of equation blocks to find bottlenecks
  - Useful as input before model simplification for real-time platforms
- Integrated with the debugger so it is possible to show what the slow equations compute
- Suitable for real-time profiling (less information), or a complete view of all equation blocks and function calls
Debugging Modelica Algorithmic Code

- Based on GDB with domain-specific extensions
- Ability to break on conditions including the time variable in Modelica
- Debug algorithmic standard Modelica code
- Debug MetaModelica code, including tree and list data structures – allows efficiently debugging the compiler itself

- Two GUI implementations, one in OpenModelica MDT Eclipse Plug-in, one in OMEdit
OpenModelica MDT Algorithmic Code Debugger
Adding Breakpoints

[Image of a software interface for adding breakpoints, with options for File Name, Line Number, Enabled, Ignore Count, Condition (Expression), and Time (Value and comparison operators)].
Summary of MODPROD Research at PELAB

• Modeling language design (semantics, type systems, meta-modeling, extensibility)
• Model-based efficient optimization (multi-core)
• Fault estimation & traceability; Bayesian networks
• Modelica-UMLSysML integration
• Requirements traceability
• Debugging of models
• Compilation to multi-core platforms
• Compilation and performance measurements for real-time simulation