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

Presentation at MODPROD'2011
PELAB – Programming Environment Laboratory
Department of Computer and Information Science
Linköping University
2011-02-08

Peter Fritzson
Examples of Complex Systems in Engineering

- Robotics
- Automotive
- Aircraft
- Mobile Phone Systems
- Business Software
- Power plants
- Heavy Vehicles
- Process industry
Vision of unified modeling framework for model-driven product development from platform independent models (PIM) to platform specific models (PSM)
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
- Basic environment for creating models
  - OMSHELL – an interactive command handler
  - OMNotebook – a literate programming notebook
  - MDT – an advanced textual environment in Eclipse
  - ModelicaML UML Profile
  - MetaModelica transforms
  - OMEdit graphic editor

© Peter Fritzson
OSMC – International Consortium for Open Source Model-based Development Tools, 32 members Dec 2010

Founded Dec 4, 2007

Open-source community services

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

Industrial members (18)

• ABB Corporate Research, Sweden
• Bosch Rexroth AG, Germany
• Siemens Turbo Machinery AB, Sweden
• CDAC Centre for Advanced Computing, Kerala, India
• CEIT Institute, Spain
• Creative Connections, Prague, Czech Republic
• Frontway AB, Sweden
• Equa Simulation AB, Sweden
• Evonik Energy Services, India
• IFP, Paris, France
• InterCAX, Atlanta, USA
• MOSTforWATER, Belgium
• MathCore Engineering AB, Sweden
• Maplesoft, Canada
• TLK Thermo, Germany
• VI-grade, Italy
• VTT, Finland
• XRG Simulation, Germany

University members (14)

• Linköping University, Sweden
• Hamburg University of Technology/TuTech, Germany
• FH Bielefeld, Bielefeld, Germany
• Technical University of Braunschweig, Germany
• Technical University of Dortmund, Germany
• Université Laval, modelEAU, Canada
• Griffith University, Australia
• Politecnico di Milano, Italy
• Mälardalen University, Sweden
• Technical University Dresden, Germany
• Telemark University College, Norway
• Ghent University, Belgium
• Ecoles des Mines, CEP, Paris, France
• University of Ljubljana, Slovenia

© Peter Fritzson
New OpenModelica Connection Editor
OMEdit

• Supports MSL 3.1
• Easy to use
• Rather stable
• Implemented in C++ Qt library
New Interactive Simulation with OpenModelica

Simulation Control

Examples of Simulation Visualization

Plot View

Requirements Evaluation View

Domain-Specific Visualization View
New OpenModelica Optimization Subsystem
OMOptim

• Parameter optimization
• Currently using genetic optimization algorithms in OMOptim 0.9.
New Web Server Based Teaching/Learning Subsystem
OMWeb

- Runs on ordinary web server
- Teachers provide modeling exercises
- Students send in solutions
- Running of simulations through server
- Lila EU project
New DrControl Interactive Electronic Book

Examples

Theory

Questions

Experimentation

Models

Example:
Consider a pendulum modeled in state space form as

\[
\begin{align*}
\dot{x} &= y \\
\dot{y} &= -g/l x \\
\end{align*}
\]

where \( x \) is the angle and \( y \) is the angular velocity, \( l \) is the length of the pendulum and \( g \) is the gravitational acceleration.

Assuming a cost function \( Q = 10 \cdot 1000 \), find the feedback law to the pendulum example from above.

Questions:

1. Question

Plot by OpenModelica

model stateSpaceNoise
StateSpaceWithNoise stateSpace;
Modelica.Blocks.Sources.Exponentials ref(
outMax=4, riseTime=1, riseTimeConst=1, 
fallTimeConst=0.2, offset=0, startTime=-1);
initial equation
stateSpace.x[1]=1;
stateSpace.x[2]=0;
equation
connect(ref.y, stateSpace.u[1]);
end stateSpaceNoise;
Research

Modeling-Language Design

Modeling Support Environments
Important Questions

• Design of modeling languages for modeling complex (physical) systems (Modelica) including precise semantics, extensibility, etc.

• How to engineer complex engineering systems of both hardware and software in a consistent and safe manner?

• Compilation of models for efficient (real-time) execution on multi-core architectures

• Traceability from requirements to models to implementation
Context of Using of Semantics for Checking
Current Semantics and MetaModelica/Modelica 4 Work

• Defining **Core language MKL** (Modeling Kernel Language) and Core MetaModelica for expressing **base semantics**

• Other **constructs** should be **expressible** in the Core language

• Defining **extensible meta-modeling/meta-programming language primitives** (e.g. MetaModelica)

• Define compiler and core library in MetaModelica, migrated to Modelica 4
Integrated Hardware-Software Modeling

ModelicaML
UML Profile for Modelica

SysML-Modelica Integration
Using ECLIPSE as Integration Platform

- OpenUP/Basic
- Capacity Sub-Process Areas
- UML-Modelica Plug-in
- OpenModelica MDT

- ECLIPSE Process Framework (EPF)
  Composer Specific components
- Graphical Modeling Framework
- ECLIPSE Modeling Framework
- ECLIPSE Rich Client Platform (RCP) Runtime
- Java runtime
- C/C++ runtime
- OpenModelica runtime
- MetaModelica runtime
ModelicaML – UML Profile for Modelica
1st Generation (2006-2008)

• Extension of SysML subset
• Features:
  • Supports Modelica constructs
  • Modelica generic class modeling
  • Modelica syntax in definitions
  • Equation-based modeling
  • Simulation modeling
ModelicaML UML Profile, 2nd Generation
SysML/UML to Modelica OMG Standardization
(with Wladimir Schamai, 2009 – Now)

• ModelicaML is a UML Profile for SW/HW modeling
  • Applicable to “pure” UML or to other UML profiles, e.g. SysML

• Standardized Mapping UML/SysML to Modelica
  • Defines transformation/mapping for executable models
  • Being standardized by OMG

• ModelicaML
  • Defines graphical concrete syntax (graphical notation for diagram) for representing Modelica constructs integrated with UML
  • Includes graphical formalisms (e.g. State Machines, Activities, Requirements)
    • Which do not exist in Modelica language
    • Which are translated into executable Modelica code
  • Is defined towards generation of executable Modelica code
  • Current implementation based on the Papyrus UML tool + OpenModelica
ModelicaML: Graphical Notation

Structure

Requirements

- **Requirement**
  - id: 001
  - text: The level of liquid in a tank shall never exceed 90% of the tank height.
  - specificType: !Tank

- **Requirement**
  - id: 002
  - text: The volume of the tank shall be 0.3 m³.
  - specificType: ![TankConnectedPL tank1]

Behavior

- **Condition Algorithm**
  - limit value algorithm
  - p₁₅m = p₁₅m;
  - else:
    - p₁₅m = p₁₅m;
  - end if

- **Condition Algorithm**
  - monitoring the level
  - monitoring the level

© Peter Fritzson
Example: Representation of System Structure
Example: Representation of System Behavior

- **State Machine of the Controller**
  - States: *off*, *monitoring the level*, *controlling the level*
  - Transitions:
    - From *off*: *not powered* to *monitoring the level*
    - From *monitoring the level*: *clnval > 0.1* to *controlling the level*
    - From *controlling the level*: *clev < 0.1* to *monitoring the level*
  - **Conditional Algorithm (Activity Diagram)**

- **State Machine of the Tank**
  - States: *empty*, *partially filled*, *overflow*
  - Transitions:
    - From *empty*: 
      - *p < pmax* to *empty*
      - *p = pmax* to *empty*
      - *else* to *partially filled*
    - From *partially filled*: 
      - *h > 0.001* to *partially filled*
      - *h = 0.001* to *overflow*
Example: Representation of System Requirements

Textual Requirement

- **Max level of liquid in a tank**
  - `maxLevel ModelicaReal`
  - `tank_height ModelicaReal`

- **Volume of the tank**
  - `tank_volume ModelicaReal`
  - `design_value ModelicaReal`

Formalized Requirement

- Monitoring the level, no violation
- Violated once or several times, continue monitoring
- Monitoring
- Violated
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 tank2 is violated

 Req. 001 for the tank1 is not violated
Parallel Execution
Compilation to MultiCore
Towards High-Level Parallel Modeling and Simulation

- Simulations are time-consuming
- Moore’s ”Law”: (since 1965)
  - #devices per chip area doubles every 18 months
  - CPU clock rate also doubled every 18 months – until 2003, then: heat and power issues, limited ILP, ...
  → superscalar technology has reached its limits, only (thread-level) parallelism can increase throughput substantially

- The consequence:
  Chip multiprocessors (+ clusters)
  - Multi-core, PIM, ... (for general-purpose computing)

- Need parallel programming/modeling/parallelization
  - Automatic parallelization
  - Explicit parallel modeling and parallel programming
Towards Easy-to-Use Modeling & Simulation using Parallel Computers

Modeling using drag’n’drop

Parallel Simulation Code

Translation

Parallel Execution

Visualization

© Peter Fritzson
Integrating Parallelism and Mathematical Models

Three Approaches

- **Automatic Parallelization of Mathematical Models (ModPar)**
  - 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.
    - NestStepModelica
Modelica Simulations – Parallelization Approach

• Simulation = solution of (hybrid) DAEs from models
  \[ g(\dot{X}, X, Y, t) = 0 \]
  \[ h(X, Y, t) = 0 \]

• In each step of numerical solver:
  • Calculate \( \dot{X} \) in g (and Y in h)

• Parallelization approach: perform the \textbf{calculation} of \( \dot{X} \) in parallel
  • Called \textit{parallelization over the system}.

• Drawback: Numeric solver might become bottle-neck
Example – Task Graphs and Parallelized Application

Clustered Task Graph

Thermofluid Pipe Application
Modified Approach – Inlining and Pipelining

• Try to keep **communication as close as possible**

• Only **communicate in one direction** inside a time step.

• **Solver Inlining** – distribute the solver across all the processors

• Some **parallelism** across the **method** – parallel evaluation of Runge-Kutta step
Task Merging vs Approach with Pipelining/Inlining

- Use a graph rewriting system to merge tasks into larger tasks, based on latency and bandwidth.
- Some tasks are duplicated to avoid communication within a step.

- Try to keep communication as close as possible.
- Only communicate in one direction inside a time step.
- Solver Inlining – distribute the solver across all the processors.
Measurements Pipelined
(100000 steps Flexible Shaft Model, Fall 2008)

Task-merging, MPI, SGI Altix

Pipelined, Pthreads, SGI, Intel Xeon

Relative speedup vs. Number of processors for SGI Altix 3700 Bx2:
- Speedup increases with the number of processors.
- Maximum speedup observed for 10 processors.

Relative speedup vs. Number of processors for Intel Xeon:
- Speedup increases linearly with the number of processors.
- Consistent speedup across different processor counts.

Relative speedup vs. Number of processors for SGI Altix 3700 Bx2:
- Speedup decreases as the number of processors increases beyond a certain point.
- Maximum speedup observed for 10 processors.

Relative speedup vs. Number of processors for Intel Xeon:
- Speedup increases linearly with the number of processors.
- Consistent speedup across different processor counts.
Speedup Measurements on NVIDIA (nov 2009)
Modelica Model, Generated Code, Function of Problem Size

- **New Platform**
  NVIDIA Fermi 2050
  2 Teraflop peak

- **In Use from January 2011**
New Scalable OpenModelica Parallel Code Generator

- Work in Progress
- Both **task** parallelism and **data** parallelism
- Handling non-expanded **arrays** efficiently
- Planned use of TLM-partitioning for more parallelism

- Generating **OpenCL** code for platform independence
- **Template** based code generator
The OpenModelica Compiler
Implemented mainly in MetaModelica and C/C++; 114 packages
Summary of MODPROD Research in PELAB

- Modeling language design (semantics, type systems, meta-modeling, extensibility)
- Modelica-SysML integration
- Requirements traceability
- Compilation to multi-core platforms
- Compilation and performance measurements for real-time simulation

Contact: peter.fritzson@liu.se