Industrial Applications of OpenModelica

Challenges and Pitfalls

Industrial Automation Power Generation, ABB AG
Agenda

Use cases of Modelica

Importance of Standards

Example: ABB OPTIMAX PowerFit

Challenges and Pitfalls with OpenModelica
Use cases of Modelica and usability of OpenModelica

Major use case: Engineering Design Simulation

Engineering Design Simulation (-)
- Ambitious expert users
- Hard: cf. License cost for best in class tool vs. Engineering cost for additional hours spent with OpenModelica

Education (+)
- Don’t require all Modelica features
- Licensing enables flexible use, e.g. in class rooms, on campuses, ...
- Openness is advantageous for deep investigations

Generate controller code for advanced model-based applications (++)
- Openness, including also development process and testing, improve code quality
- Licensing enables flexible use, e.g. installation on customer machines
- Can customize yourself to industrial needs (e.g. C++ runtime)
Industrial applications: Mitigate risks with standards

Modelica and FMI

Traditional approach
- Each tool defines its own proprietary interfaces and formats
- Possible problems show up late
- Tool changes or even upgrades become very expensive

Exploitation of standards
- Modelica saves investments into model development
- FMI saves investments into runtime software development

Low barrier for start using a new Modelica tool, like OpenModelica
ABB Dynamic Optimization – standardized model-based applications

Basis for several OPTIMAX application, e.g. Soot BlowMax and PowerFit

Modelica for application engineering (www.modelica.org)

FMI (Functional Model Interface) for deployment (www.fmi-standard.org)
ABB OPTIMAX PowerFit

General System Architecture

- Optimal schedules
- Unit commitment
- Balancing group optimization
- Balancing power calls
- Combined heat and power optimization
- Multi-Energy site optimization
- Demand response
- Autonomous operation of distribution grids

Interfaces

OPTIMAX® PowerFit
Central control and optimization

Real-time interfaces

Technical Units

Forecasts
Delivery Commitments
Plant Schedules
Plant data

Energy Management / Trader

Availabilities
Diagnosis

Accounting
Reporting

TSO Grid Services

DSO Automation

GPRS / UMTS / LTE

Firewall / Router / VPN
Combined Grid Solution

New 400 MW interconnector utilizing existing or to be build offshore wind power collector grids
Implementation of a Master controller for Interconnector Operation

Summary

Highlights

- Designed and implemented by ABB
- Based on ABB OPTIMAX PowerFit
- Optimal power flow calculation using KF grid model with state estimation
- Geo-redundant servers
- IEC 60870-5-104 communication
Use of OpenModelica

Challenges and Pitfalls

Challenges

- Push advanced model-based control applications by making them accessible in a graphical tool
- Good: OpenModelica becomes increasingly applicable (features, speed, crashes, localization)
- Good: OpenModelica license permits widespread use
- Good: OpenModelica permits own contribution to push limits in cutting edge applications
- Missing: encryption of libraries

Pitfalls

- OpenModelica often appears as “do it yourself” tool for industrial applications
  - C runtime for academic use vs. C++ runtime for industrial use vs. embedded C runtime for R&D
  - Incomplete implementation of Modelica standard, e.g. synchronous features, inline integration, ...
- Implementation of OpenModelica appears inefficient
  - See effort required to provide a new feature, like 64bit or replaceable models
## Sources for inefficiency of MetaModelica

String conversion – good languages have a common convention; what about MetaModelica?

<table>
<thead>
<tr>
<th>Modelica</th>
<th>MetaModelica (some examples)</th>
</tr>
</thead>
<tbody>
<tr>
<td>String()</td>
<td>intString()</td>
</tr>
<tr>
<td></td>
<td>ExpressionDumpTpl.dumpExp</td>
</tr>
<tr>
<td></td>
<td>DAEDump.dumpOperatorStringString()</td>
</tr>
<tr>
<td></td>
<td>DAEDump.dumpExtDeclStr</td>
</tr>
<tr>
<td></td>
<td>DAEDump.derivativeCondStrString()</td>
</tr>
<tr>
<td></td>
<td>...</td>
</tr>
</tbody>
</table>

| C++               |                                                                    |
| to_string()       | Integer: shortened type + String                                   |
|                   | Exp: dump + type name                                              |
|                   | Operator: dump + type name + String                                |
|                   | ExternalDecl: dump + shortened type name + String                 |
|                   | DerivativeCond: type name + String                                 |

| Julia             |                                                                    |
| string()          |                                                                    |
Sources for inefficiency of MetaModelica

Array access – good languages use array access operator; what about MetaModelica?

<table>
<thead>
<tr>
<th>Modelica</th>
<th>MetaModelica</th>
</tr>
</thead>
<tbody>
<tr>
<td>arr[index]</td>
<td>Dangerous.arrayGetNoBoundsChecking(arr, Integer(index))</td>
</tr>
</tbody>
</table>

C/ C++

arr[index]

Julia

arr[index]
Sources for inefficiency of MetaModelica

Function signatures – means to distinguish function inputs from outputs; about MetaModelica?

<table>
<thead>
<tr>
<th>Modelica</th>
<th>MetaModelica</th>
</tr>
</thead>
<tbody>
<tr>
<td>input arg</td>
<td>protected function addSimVar</td>
</tr>
<tr>
<td>output ret</td>
<td>input SimCodeVar.SimVar simVar;</td>
</tr>
<tr>
<td></td>
<td>input SimVarsIndex index;</td>
</tr>
<tr>
<td>C/ C++</td>
<td></td>
</tr>
<tr>
<td>f(const reference)</td>
<td>algorithm</td>
</tr>
<tr>
<td></td>
<td>Dangerous.arrayUpdateNoBoundsChecking(simVars, Integer(index),</td>
</tr>
<tr>
<td></td>
<td>simVar::Dangerous.arrayGetNoBoundsChecking(simVars,</td>
</tr>
<tr>
<td></td>
<td>Integer(index)));</td>
</tr>
<tr>
<td>Julia</td>
<td></td>
</tr>
<tr>
<td>f!(reference)</td>
<td>// simVars[index] := simVar :: simVars[index];</td>
</tr>
<tr>
<td></td>
<td>end addSimVar;</td>
</tr>
</tbody>
</table>
Conclusions

**OpenModelica** cannot yet compete in the field of engineering design simulations

- Increasingly applicable in special cases though

**OpenModelica** is very attractive generation of controller code for advanced applications

- Openness, including also development process and testing, improve code quality
- Can customize yourself to industrial needs (e.g. C++ runtime)

**Unsolved:** need of industrial users to largely “do it yourself”

- C runtime vs. C++ runtime vs. embedded C runtime
- Increase dev productivity – better exploit software technologies developed after 1989 (C89)?
- First work on Modelica and FMI standards, proprietary additions with second prio