DAE Tools Software

Introduction

D.D. Nikolić

Updated: 1 April 2016

DAE Tools Project, `http://www.daetools.com`

[d][a][e]
Tools Project
Model the world freely

Intro

What is DAE Tools?

Process MODELLING, SIMULATION, and OPTIMISATION software

- Areas of application:
  - Initially: chemical process industry (mass, heat and momentum transfers, chemical reactions, separation processes, thermodynamics, electro-chemistry)
  - Nowadays: MULTI-DOMAIN
- Free/Open source software (GNU GPL)
- CROSS-PLATFORM (GNU/Linux, MacOS, Windows)
- MULTIPLE ARCHITECTURES (32/64 bit x86, ARM, ...)

What is DAE Tools? (cont'd)

- DAE Tools IS NOT:
  - A modelling language (such as Modelica and gPROMS)
  - An integrated software suite of data structures and routines for scientific applications (such as PETSc)
- DAE Tools IS:
  - A higher level structure – an architectural design of interdependent software components providing an API for:
    - Model development/specification
    - Activities on developed models (simulation, optimisation, ...)
    - Processing of the results
    - Report generation
    - Code generation and model exchange

What can be done with DAE Tools?

- Simulation
  - Steady-State
  - Transient
- Optimisation
  - Non-Linear Programming (NLP) problems
  - Mixed Integer Non-Linear Programming (NLP) problems
- Parameter estimation
  - Levenberg–Marquardt algorithm
- Code-generation, model-exchange, co-simulation
  - Modelica, gPROMS, Matlab, Simulink
  - Functional Mockup Interface (FMI)
  - C99 (for embedded systems)
  - C++ MPI (for distributed computing)

Types of systems that can be modelled

Initial value problems of implicit form, (described by systems of linear, non-linear, and (partial-)differential algebraic equations).

- Continuous with some elements of event-driven systems (discontinuous equations, state transition networks and discrete events)
- Steady-state or dynamic
- With lumped or distributed parameters (finite difference, finite volume and finite element methods)
- Only index-1 DAE systems at the moment

Why modelling software?

In general, two scenarios:

- Development of a new product/process/...
  - Reduce the time to market (TTM)
  - Reduce the development costs (no physical prototypes)
  - Maximise the performance, yield, productivity, purity, ...
  - Minimise the capital and operating costs
  - Explore the new design options with no risks
- Optimisation of an existing product/process/...
  - Increase the performance, yield, productivity, purity, ...
  - Reduce the operating costs, energy consumption, ...

Why yet another modelling software?

Currently available options:

1. Modelling languages (domain-specific or multi-domain): Modelica , Ascend , gPROMS , GAMS , Dymola , APMonitor
2. General-purpose programming languages:
   - Lower level third-generation languages such as C, C++ and Fortran: PETSc , SUNDIALS , Assimulo
   - Multi-paradigm numerical languages: Matlab , Mathematica , Maple , Scilab , and GNU Octave

Why yet another modelling software? (cont'd)

The advantages of the Hybrid approach betwen modelling and general-purpose programming languages:

1. Support for the runtime model generation
2. Support for the runtime simulation set-up
3. Support for complex runtime operating procedures
4. Interoperability with the third-party software packages (i.e. NumPy/SciPy)
5. Suitability for embedding and use as a web application or software as a service
6. Code-generation, model exchange and co-simulation capabilities

Not a modelling language (cont'd)

- Allows easy interaction with other software libraries (two-way interoperability with other software, embedding in other software etc.)
- Developed in c++ with Python bindings (Boost.Python)

Object-oriented modelling

- Everything is an object (models, parameters, variables, equations, state transition networks, simulations, solvers, ...)
- Models are classes derived from the base daeModel class (inheriting the common functionality)
- Hierarchical model decomposition allows creation of complex, re-usable model definitions
- All Object Oriented concepts supported (such as multiple inheritance, templates, polymorphism, ...) that are supported by the target language (c++, Python), except:
  - Derived classes always inherit all declared objects (parameters, variables, equations, ...)

Equation-oriented (acausal) modelling

- Equations given in an implicit form (as a residual)

$$F(\dot{x}, x, y, p) = 0$$

- Input-Output causality is not fixed:
  - Increased model re-use
  - Support for different simulation scenarios (based on a single model) by specifying different degrees of freedom
- For instance, equation given in the following form:

$$x_1 + x_2 + x_3 = 0$$

can be used to determine either x_1, x_2 or x_3 depending on what combination of variables is known:

$$x_1 = -x_2 - x_3 \text{ or } x_2 = -x_1 - x_3 \text{ or } x_3 = -x_1 - x_2$$

Separation of models definition from operations on them

- The structure of the model (parameters, variables, equations etc.) given in the model classes (*daeModel*, *daeFiniteElementModel*)
- The runtime information in the simulation class (*daeSimulation*)
- Single model definition, but:
  - One or more different simulation scenarios
  - One or more optimization scenarios

Hybrid continuous/discrete systems

- Modelling of continuous systems with some elements of event-driven systems
  - Discontinuous equations
  - State transition networks
  - Discrete events

Code generation

- Model export from DAE Tools to other DSL/modelling/programming languages
  - Modelica
  - c99

Model Exchange

- Support for Functional Mock-up Interface for Model Exchange and Co-Simulation (FMI): https://www.fmi-standard.org
- FMI – a tool independent standard to support both model exchange and co-simulation of dynamic models using a combination of xml-files and compiled C-code
- Still in experimental phase

Model reports

- Automatic model documentation
- XML + MathML format
- XSL transformation used to to generate HTML code and visualize reports
- Two types:
  - Model description report (contains model definition)
  - Runtime report with all values and equations expanded (contains definition of the simulation)

Model reports (cont'd)

Parameters

Name	Units	Domains	Description
Q_b	Wm^{-2}		Heat flux at the bottom edge of the plate
Q_t	Wm^{-2}		Heat flux at the top edge of the plate
ρ	kgm^{-3}		Density of the plate
c_p	$JK^{-1}kg^{-1}$		Specific heat capacity of the plate
λ_p	$WK^{-1}m^{-1}$		Thermal conductivity of the plate

Variables

Name	Type	Domains	Description
T	temperature_t	x , y	Temperature of the plate, K
test	temperature_t		

Equations

HeatBalance :

$$\rho \cdot c_p \cdot \frac{\mathrm{d}T(x,y)}{\mathrm{d}t} - \lambda_p \cdot \left(\frac{\partial^2 T(x,y)}{\partial x^2} + \frac{\partial^2 T(x,y)}{\partial y^2} \right) = 0 \; ; \forall x \in (x_0, x_n), \forall y \in (y_0, y_n)$$

Heat balance equation. Valid on the open x and y domains

BC_{bottom} *:*

$$((-\lambda_p)) \cdot \frac{\partial T(x,y)}{\partial y} - Q_b = 0 \; ; \forall x \in [x_0, x_n], y = y_0$$

Boundary conditions for the bottom edge

BC_{top} *:*

$$((-\lambda_p)) \cdot \frac{\partial T(x,y)}{\partial y} - Q_t = 0 \; ; \forall x \in [x_0, x_n], y = y_n$$

Boundary conditions for the top edge

[d][a][e] Tools Project
Model the world freely

Model reports (cont'd)

BC_{right} :

$$\frac{\partial T(x,y)}{\partial x} = 0 ; x = x_n, \forall\, y \in (y_0, y_n)$$

Boundary conditions for the right edge

Expanded into:

$$\frac{3 \cdot tutorial1.T(25,1) - 4 \cdot tutorial1.T(24,1) + tutorial1.T(23,1)}{tutorial1.x[25] - tutorial1.x[23]} = 0$$

Equation is: Linear

$$\frac{3 \cdot tutorial1.T(25,2) - 4 \cdot tutorial1.T(24,2) + tutorial1.T(23,2)}{tutorial1.x[25] - tutorial1.x[23]} = 0$$

Equation is: Linear

$$\frac{3 \cdot tutorial1.T(25,3) - 4 \cdot tutorial1.T(24,3) + tutorial1.T(23,3)}{tutorial1.x[25] - tutorial1.x[23]} = 0$$

Equation is: Linear

$$\frac{3 \cdot tutorial1.T(25,4) - 4 \cdot tutorial1.T(24,4) + tutorial1.T(23,4)}{tutorial1.x[25] - tutorial1.x[23]} = 0$$

Equation is: Linear

$$\frac{3 \cdot tutorial1.T(25,5) - 4 \cdot tutorial1.T(24,5) + tutorial1.T(23,5)}{tutorial1.x[25] - tutorial1.x[23]} = 0$$

Equation is: Linear

$$\frac{3 \cdot tutorial1.T(25,6) - 4 \cdot tutorial1.T(24,6) + tutorial1.T(23,6)}{tutorial1.x[25] - tutorial1.x[23]} = 0$$

Equation is: Linear

$$\frac{3 \cdot tutorial1.T(25,7) - 4 \cdot tutorial1.T(24,7) + tutorial1.T(23,7)}{tutorial1.x[25] - tutorial1.x[23]} = 0$$

Equation is: Linear

$$\frac{3 \cdot tutorial1.T(25,8) - 4 \cdot tutorial1.T(24,8) + tutorial1.T(23,8)}{tutorial1.x[25] - tutorial1.x[23]} = 0$$

Equation is: Linear

$$\frac{3 \cdot tutorial1.T(25,9) - 4 \cdot tutorial1.T(24,9) + tutorial1.T(23,9)}{tutorial1.x[25] - tutorial1.x[23]} = 0$$

Equation is: Linear

$$\frac{3 \cdot tutorial1.T(25,10) - 4 \cdot tutorial1.T(24,10) + tutorial1.T(23,10)}{tutorial1.x[25] - tutorial1.x[23]} = 0$$

Equation is: Linear

$$\frac{3 \cdot tutorial1.T(25,11) - 4 \cdot tutorial1.T(24,11) + tutorial1.T(23,11)}{tutorial1.x[25] - tutorial1.x[23]} = 0$$

Equation is: Linear

$$\frac{3 \cdot tutorial1.T(25,12) - 4 \cdot tutorial1.T(24,12) + tutorial1.T(23,12)}{tutorial1.x[25] - tutorial1.x[23]} = 0$$

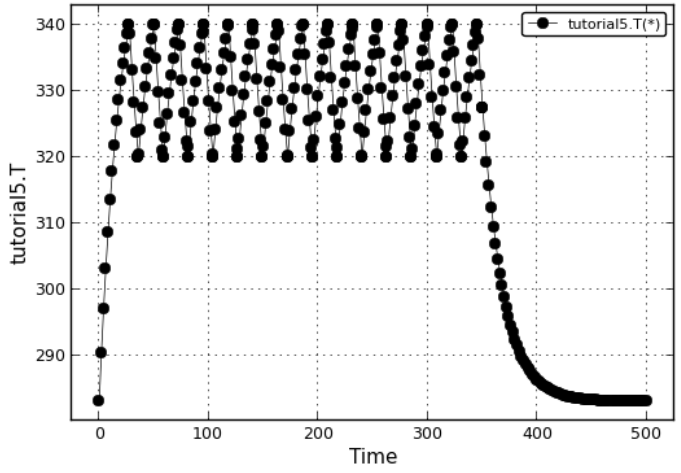
[d][a][e] Tools Project
Model the world freely

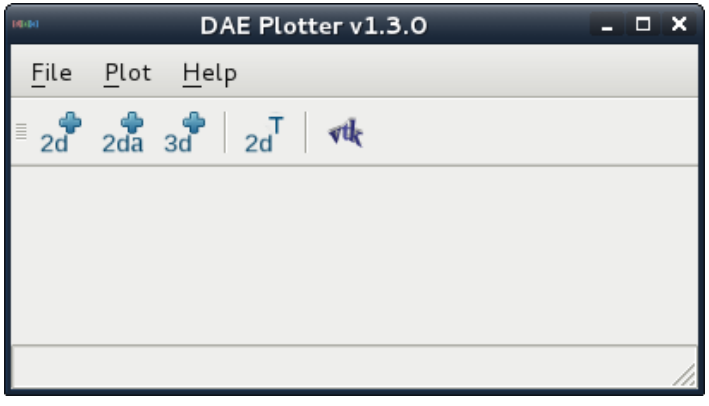
Multi-domain

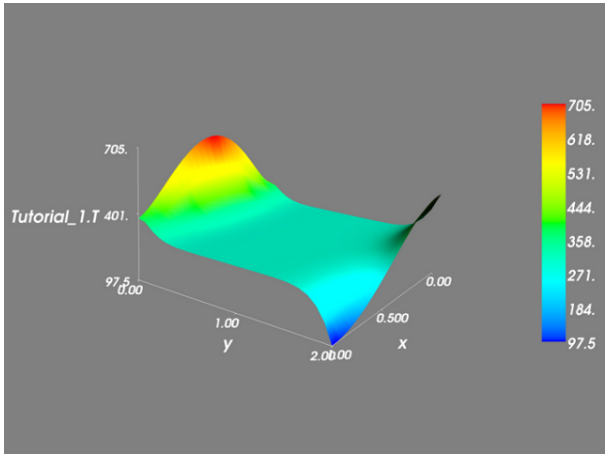
- From chemical processing industry to biological neural networks
- DAE Tools is not a DSL but defines the basic modelling concepts such as models, parameters, variables, various types of equations (ordinary, differential, partial differential, discontinuous), state transition networks etc. that can be used as building blocks for a specific domain
- Example: a reference implementation simulator for NineML (xml-based modelling language for describing networks of spiking neurons)
- The key concepts from NineML are based on DAE Tools concepts: Neurone, Synapse, Population of neurones,

DAE Plotter

- 2D plots (Matplotlib)
- Animated 2D plots
- 3D plots (Mayavi)







[d][a][e] Tools Project
Model the world freely

Solvers

Supported DAE solvers@

- Sundials IDAS
  (https://computation.llnl.gov/casc/sundials/main.html)

Supported FE libraries:

- deal.II (http://dealii.org)

Supported optimization solvers:

- IPOPT (https://projects.coin-or.org/Ipopt)

Solvers

Supported LA solvers:

- Sundials dense LU, Lapack
- Trilinos Amesos
  (http://trilinos.sandia.gov/packages/amesos)
- Trilinos AztecOO
  (http://trilinos.sandia.gov/packages/aztecoo)
- SuperLU SuperLU-MT
  (http://crd.lbl.gov/ xiaoye/SuperLU/index.html)
- Umfpack
  (http://www.cise.ufl.edu/research/sparse/umfpack)
- MUMPS (http://graal.ens-lyon.fr/MUMPS)

Programming paradigms

Approaches to process modelling

- Two approaches to process modelling:
  - Domain Specific Language (DSL)
  - General-purpose programming language (such as c, c++, Java or Python)

Domain Specific Languages

- Special-purpose programming or specification languages dedicated to a particular problem domain
- Designed to directly support the key concepts from that domain
- Specifically created to solve problems in a particular domain
- (Usually) not intended to solve problems outside that domain (although that may be technically possible in some cases)
- Commonly lack low-level functions for filesystem access, interprocess control, and other functions that characterize full-featured programming languages, scripting or

General-purpose programming languages

- Created to solve problems in a wide variety of application domains
- Do not support key concepts from any domain
- Have low-level functions for filesystem access, interprocess control etc.
- Examples: c, c++, Fortran, Python, Java etc.
- Typical scenario: solving a DAE system
  - Choose a solver (Sundials IDA, DASSL, RADAU5, DAEPACK etc)
  - Implement user functions to manually calculate residuals and derivatives for a Jacobian matrix, apply boundary conditions etc

DAE Tools approach

A sort of the hybrid approach:

- Applies general-purpose programming languages such as c++ and Python
- Offers a class-hierarchy/API that resembles a syntax of a DSL as much as possible.
- Provides low-level concepts such as parameters, variables, equations, ports, models, state transition networks, discrete events etc.
- Concepts from new application domains can be added on top of its low level concepts (for instance the simulator for biological neural networks - NineML, as it will be shown later in Use Case section)

gPROMS vs. Modelica

```
PARAMETER
  Density as Real
  CrossSectionalArea as Real
  Alpha as Real

VARIABLE
  HoldUp as Mass
  FlowIn as Flowrate
  FlowOut as Flowrate
  Height as Length

EQUATION
  # Mass balance
  $HoldUp = FlowIn - FlowOut;

  # Relation betwee liquid level and holdup
  HoldUp = CrossSectionalArea * Height * Density;

  # Relation between pressure drop and flow
  FlowOut = Alpha * sqrt(Height);
```

Model developed in gPROMS

http://www.psenterprise.com

```
model BufferTank
  /* Import libs */
  import Modelica.Math.*;

  parameter Real Density;
  parameter Real CrossSectionalArea;
  parameter Real Alpha;

  Real HoldUp(start = 0.0);
  Real FlowIn;
  Real FlowOut;
  Real Height;

equation
// Mass balance
  der(HoldUp) = FlowIn - FlowOut;

// Relation betwee liquid level and holdup
  HoldUp = CrossSectionalArea * Height * Density;

// Relation between pressure drop and flow
  FlowOut = Alpha * sqrt(Height);

end BufferTank;
```

The same model in OpenModelica

https://www.openmodelica.org

[d][a][e] Tools Project
Model the world freely

DAE Tools

```
2  class BufferTank(daeModel):
3      def __init__(self, Name, Parent = None, Description = ""):
4          daeModel.__init__(self, Name, Parent, Description)
5
6          self.Density = daeParameter("Density", unit(), self)
7          self.Area    = daeParameter("Area",    unit(), self)
8          self.Alpha   = daeParameter("Alpha",   unit(), self)
9
10         self.HoldUp  = daeVariable("HoldUp",  no_t, self)
11         self.FlowIn  = daeVariable("FlowIn",  no_t, self)
12         self.FlowOut = daeVariable("FlowOut", no_t, self)
13         self.Height  = daeVariable("Height",  no_t, self)
14
15     def DeclareEquations(self):
16         # Mass balance
17         eq = self.CreateEquation("MassBalance")
18         eq.Residual = self.HoldUp.dt() - self.FlowIn() + self.FlowOut()
19
20         # Relation between liquid level and holdup
21         eq = self.CreateEquation("LiquidLevelHoldup")
22         eq.Residual = self.HoldUp() - self.Area() * self.Height() * self.Density()
23
24         # Relation between pressure drop and flow
25         eq = self.CreateEquation("PressureDropFlow")
26         eq.Residual = self.FlowOut() - self.Alpha() * Sqrt(self.Height())
```

[d][a][e] Tools Project
Model the world freely

The same model in daetools

Key modelling concepts & grammar

DSL/Modelling languages

- Domain-specific languages allow solutions to be expressed in the idiom and at the level of abstraction of the problem domain (direct support for all modelling concepts by the language syntax)

DAE Tools

- Modelling concepts cannot be expressed directly in the programming language and have to be emulated in the API or in some other way

Verbosity

DSL/Modelling languages

- Clean, concise, ellegant and natural way of building model descriptions: the code can be self documenting

DAE Tools

- The support for modelling concepts is much more verbose and less elegant; however, DAE Tools can generate XML+MathML based model reports that can be either rendered in XHTML format using XSLT transformations (representing the code documentation) or used as an XML-based model exchange language

Maintainability & portability

DSL/Modelling languages

- Domain-specific languages could enhance quality, productivity, reliability, maintainability and portability

DAE Tools

-

Simulators & programming languages

DSL/Modelling languages

- DSLs could be and often are simulator independent making a model exchange easier

DAE Tools

- Programming language dependent; however, a large number of scientific software libraries exposes its functionality to Python via Python wrappers

Need for a compiler/parser/interpreter

DSL/Modelling languages

- Cost of designing, implementing, and maintaining a domain-specific language as well as the tools required to develop with it (IDE): a compiler/lexical parser/interpreter must be developed with all burden that comes with it (such as error handling, grammar ambiguities, hidden bugs etc)

DAE Tools

- A compiler/lexical parser/interpreter is an integral part of the programming language (c++, Python) with a robust

Need for a new language syntax

DSL/Modelling languages

- Cost of learning a new language vs. its limited applicability: users are required to master a new language (yet another language grammar)

DAE Tools

- No learning of a new language required (everything can get done in a favourite programming language)

Interoperability with the 3^{rd} party software

DSL/Modelling languages

- Increased difficulty of integrating the DSL with other components: calling external functions/libraries and interaction with other software is limited by the existence of wrappers around a simulator engine (for instance some scripting languages like Python or javascript)

DAE Tools

- Calling external functions/libraries is a natural and straightforward Interaction with other software is natural and straightforward

Runtime generation & modification

DSL/Modelling languages

- Models usually cannot be created in the runtime/on the fly (or at least not easily) and cannot be modified in the runtime

DAE Tools

- Models can be created in the runtime/on the fly and easily modified in the runtime

Simulation setup

DSL/Modelling languages

- Setting up a simulation (ie. the values of parameters values, initial conditions, initially active states) is embedded in the language and it is typically difficult to do it on the fly or to obtain the values from some other software (for example to chain several software calls where outputs of previous calls represent inputs to the subsequent ones)

DAE Tools

- Setting up a simulation is done programmatically and the initial values can be obtained from some other software in a

Operating procedures

DSL/Modelling languages

- Simulation operating procedures are not flexible; manipulation of model parameters, variables, equations, simulation results etc is limited to only those operations provided by the language

DAE Tools

- Operating procedures are completely flexible (within the limits of a programming language itself) and a manipulation of model parameters, variables, equations, simulation results etc can be done in any way which a user cosiders suitable for his/her problem

Outputs

DSL/Modelling languages

- Only the type of results provided by the language/simulator is available; custom processing is usually not possible or if a simulator does provide a way to build extensions it is limited to the functionality made available to them

DAE Tools

- The results processing can be done in any way which a user considers suitable(again within the limits of a programming language itself)

Architecture

Available modules

- pyDAE:
  - pyCore (key modelling concepts)
  - pyActivity (simulation, optimization)
  - pyDataReporting (results handling)
  - pyIDAS (DAE solver)
  - pyUnits (*unit* and *quantity* concepts)

- FE Solvers:
  - pyDealII

[d][a][e] Tools Project
Model the world freely

Available modules (cont'd)

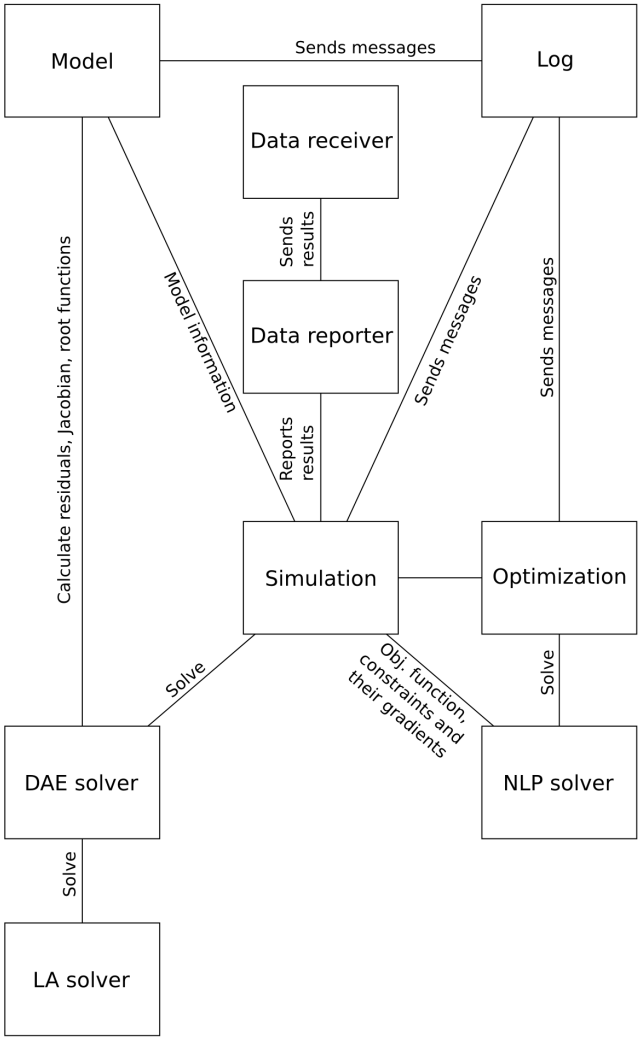
- LA Solvers:
  - pySuperLU
  - pySuperLU_MT
  - pyTrilinos (Amesos, AztecOO)
  - pyIntelPardiso

- NLP/MINLP Solvers:
  - pyIPOPT
  - pyBONMIN
  - pyNLOPT

Available components

- Model
- Simulation
- Optimization
- DAE solver
- LA solver
- NLP solver
- Log
- Data reporter
- Data receiver

Available components (cont'd)



[d][a][e] Tools Project
Model the world freely

Use Cases