

# DAE TOOLS SOFTWARE

## INTRODUCTION

---

D.D. Nikolić

Updated: 4 May 2017

DAE Tools Project, <http://www.daetools.com>







1. General Information
2. Motivation
3. Programming Paradigms
4. Architecture
5. Use Cases

## GENERAL INFORMATION

---

# What is DAE Tools?

MODELLING, SIMULATION, OPTIMISATION & PARAMETER ESTIMATION software <sup>1</sup>

- 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**   
- **MULTIPLE ARCHITECTURES** (32/64 bit x86, ARM, ...)

---

<sup>1</sup>Nikolić DD. (2016) *DAE Tools: equation-based object-oriented modelling, simulation and optimisation software*. PeerJ Computer Science 2:e54

# What is DAE Tools? (cont'd)

- DAE Tools **IS NOT**:
  - A modelling language (such as Modelica)
  - An integrated software suite of data structures and routines for scientific applications (such as PETSc, Sundials, ...)
- DAE Tools **IS**:
  - An **ARCHITECTURAL DESIGN OF INTERDEPENDENT SOFTWARE COMPONENTS** providing an API for:
    - **MODEL SPECIFICATION**
    - Activities on developed models (**SIMULATION**, **OPTIMISATION**, ...)
    - **PROCESSING OF THE RESULTS**
    - **REPORT GENERATION**
    - **CODE GENERATION** and **MODEL EXCHANGE**
- DAE Tools apply a **HYBRID APPROACH** between **MODELLING** and **GENERAL PURPOSE** programming languages, **COMBINING THE STRENGTHS OF BOTH APPROACHES** into a single one

# What can be done with DAE Tools?

- SIMULATION
  - Steady-State
  - Transient
- OPTIMISATION
  - Non-Linear Programming (NLP) problems
  - Mixed Integer Non-Linear Programming (MINLP) problems
- PARAMETER ESTIMATION
- CODE-GENERATION, MODEL-EXCHANGE, CO-SIMULATION
  - Modelica, gPROMS
  - Matlab MEX-functions, Simulink user-defined S-functions
  - 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

# MOTIVATION





# 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 in less time and no risks
- **OPTIMISATION** of an **EXISTING** product/process/...
  - Increase the performance, yield, productivity, purity, ...
  - Reduce the operating costs, energy consumption, ...
  - Debottleneck

# Why YET ANOTHER modelling software?

Current approaches to mathematical modelling:

1. Use of **MODELLING LANGUAGES** (domain-specific or multi-domain): **MODELICA**, **ASCEND**, **gPROMS**, **DYMOLA**, **APMonitor**
2. Use of **GENERAL-PURPOSE PROGRAMMING LANGUAGES**:
  - Lower level third-generation languages such as C, C++ and Fortran (**PETSc**, **SUNDIALS**)
  - Higher level fourth-generation languages such as **PYTHON** (NumPy, SciPy, Assimulo), **JULIA** etc.
  - Multi-paradigm numerical languages (**MATLAB**, **MATHEMATICA**, **MAPLE**, **SCILAB**, and **GNU OCTAVE**)

# Why YET ANOTHER modelling software? (cont'd)

The advantages of the **HYBRID** approach over the **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**
5. Suitability for **EMBEDDING** and use as a **WEB APPLICATION** or **SOFTWARE AS A SERVICE**
6. **CODE-GENERATION**, **MODEL EXCHANGE** and **CO-SIMULATION** capabilities

# Additional DAE TOOLS features

- Support for **MULTIPLE PLATFORMS / ARCHITECTURES**
- Support for the **AUTOMATIC DIFFERENTIATION** (ADOL-C)
- Support for the **SENSITIVITY ANALYSIS** through the auto-differentiation capabilities
- Support for the **PARALLEL** computation (OpenMP, GPGPU, MPI)
- Support for a large number of **DAE**, **LA** and **NLP** solvers
- Support for the generation of **MODEL REPORTS** (XML + MathML, Latex)
- **EXPORT** of the **SIMULATION RESULTS** to various file formats (Matlab, Excel, json, xml, HDF5, Pandas, VTK)

# PROGRAMMING PARADIGMS

---

# The HYBRID approach

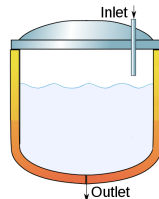
- DAE Tools approach is a **TYPE OF A HYBRID APPROACH**
- Combines strengths of **MODELLING** and **GENERAL PURPOSE** programming languages:
  1. **DEVELOPED IN C++** with the **PYTHON BINDINGS**
  2. Provides **API** (Application Programming Interface) that **RESEMBLES A SYNTAX OF MODELLING LANGUAGES** as much as possible
  3. **TAKES ADVANTAGE OF THE HIGHER LEVEL LANGUAGES** for:
    - Model specification, simulation setup, operating procedures
    - Access to the operating system
    - Access to the standard/third-party libraries

# The HYBRID approach (cont'd)

- Modelica/gPROMS grammars vs. DAE Tools API

- A simple model:

Cylindrical tank containing a liquid with an inlet and an outlet flow; the outlet flowrate depends on the liquid level in the tank



```
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 between liquid level and holdup
  HoldUp = CrossSectionalArea * Height * Density;

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

gPROMS grammar

```
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 between liquid level and holdup
  HoldUp = CrossSectionalArea * Height * Density;

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

end BufferTank;
```

Modelica grammar

# The HYBRID approach (cont'd)

```
class BufferTank(daeModel):
    def __init__(self, Name, Parent = None, Description = ""):
        daeModel.__init__(self, Name, Parent, Description)

        self.Density = daeParameter("Density", unit(), self)
        self.CrossSectionalArea = daeParameter("CrossSectionalArea", unit(), self)
        self.Alpha = daeParameter("Alpha", unit(), self)

        self.HoldUp = daeVariable("HoldUp", no_t, self)
        self.FlowIn = daeVariable("FlowIn", no_t, self)
        self.FlowOut = daeVariable("FlowOut", no_t, self)
        self.Height = daeVariable("Height", no_t, self)

    def DeclareEquations(self):
        # Mass balance
        eq = self.CreateEquation("MassBalance")
        eq.Residual = self.HoldUp.dt() - self.FlowIn() + self.FlowOut()

        # Relation between liquid level and holdup
        eq = self.CreateEquation("LiquidLevelHoldup")
        eq.Residual = self.HoldUp() - self.CrossSectionalArea() * self.Height() * self.Density()

        # Relation between pressure drop and flow
        eq = self.CreateEquation("PressureDropFlow")
        eq.Residual = self.FlowOut() - self.Alpha() * Sqrt(self.Height())
```

## DAE Tools API



# The HYBRID approach (cont'd)

Modelling language approach	DAE Tools approach
Solutions expressed in the idiom and at the level of abstraction of the problem domain	Must be emulated in the API or in some other way
Clean and concise way of building models	Verbose and less elegant
Could be and often are simulator independent	Simulator dependent (but with code-generation)
Cost of designing, implementing, and maintaining a language and a compiler/lexical parser/interpreter, error handling and grammar ambiguities	A compiler/lexical parser/interpreter is an integral part of C++/Python with a robust error handling, universal grammar and massively tested
Cost of learning a new language vs. its limited applicability (yet another language grammar)	No learning of a new language required
Difficult to integrate with other components	Calling external libraries is a built-in feature
Models usually cannot be created/modified in the runtime (or at least not easily)	Models can be created/modified in the runtime
Setting up a simulation embedded in the language; difficult to obtain initial values from other software	Setting up a simulation done programmatically and the initial values can be obtained from other software
Simulation operating procedures limited to the options allowed by the language grammar	Operating procedures completely flexible (within the limits of a programming language itself)

# The OBJECT-ORIENTED approach

- Everything is an **OBJECT** (variables, equations, models ...)
- All objects can be **MANIPULATED** in **THE RUNTIME**
- **ALL C++/Python OBJECT-ORIENTED CONCEPTS SUPPORTED**
  - **EXCEPTION:** all declared DAE Tools objects **REMAIN PUBLIC**
- Models, simulations, optimisations:
  - Classes **DERIVED FROM** the corresponding **BASE CLASSES**
  - **INHERIT** the **COMMON FUNCTIONALITY** from the base classes
  - Perform the **FUNCTIONALITY** in **OVERLOADED FUNCTIONS**
- The **HIERARCHICAL MODEL DECOMPOSITION** possible:
  - Models can contain instances of other models
  - Complex, re-usable model definitions can be created
  - Models at different scales can be loosely coupled

# The EQUATION-ORIENTED (ACAUSAL) approach

- 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

- An example:

- The 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 MODEL DEFINITION from its APPLICATIONS

- MODEL STRUCTURE specified in the MODEL CLASS
- RUNTIME INFORMATION specified in the SIMULATION CLASS
- SOLVERS/AUXILIARY OBJECTS declared in the MAIN PROGRAM
- SINGLE MODEL DEFINITION, but ONE OR MORE:
  - Different SIMULATION SCENARIOS
  - Different OPTIMIZATION SCENARIOS

# ARCHITECTURE

---

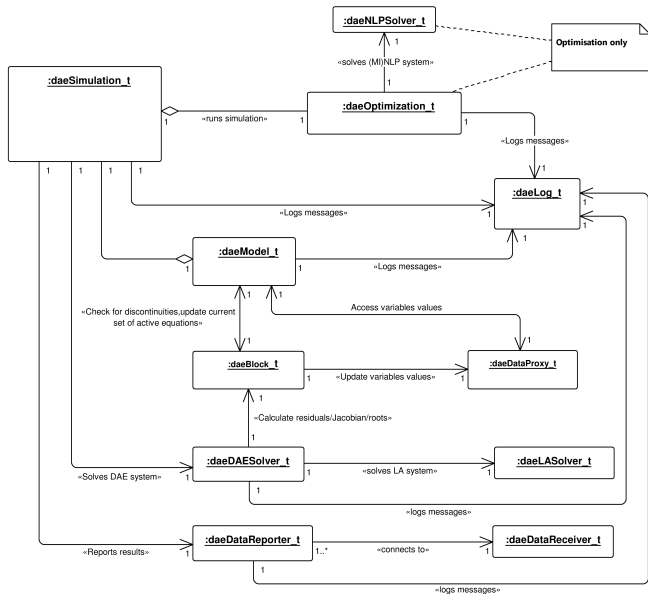
# The fundamental concepts/software interfaces

- The main concepts:

- `daeModel_t`
- `daeSimulation_t`
- `daeOptimization_t`
- `daeBlock_t`
- `daeDAESolver_t`
- `daeLASolver_t`
- `daeDataReporter_t`
- `daeBlock_t`

- In 6 packages:

- CORE
- ACTIVITY
- DATAREPORTING
- SOLVERS
- LOGGING
- UNITS



TabelaThe key modelling concepts in the **CORE** package.

Concept	Description
<i>daeVariableType_t</i>	Defines a variable type that has the units, lower and upper bounds, a default value and an absolute tolerance
<i>daeDomain_t</i>	Defines ordinary arrays or spatial distributions such as structured and unstructured grids
<i>daeParameter_t</i>	Defines time invariant quantities that do not change during a simulation
<i>daeVariable_t</i>	Defines time varying quantities that change during a simulation
<i>daePort_t</i>	Defines connection points between model instances for exchange of continuous quantities
<i>daeEventPort_t</i>	Defines connection points between model instances for exchange of discrete messages/events

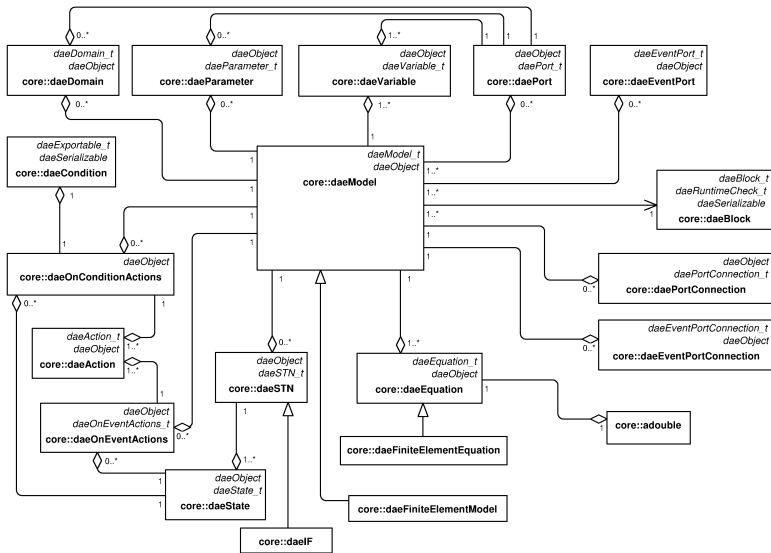
# Package CORE (cont'd)

TabelaThe key modelling concepts in the **CORE** package (cont'd).

Concept	Description
<i>daePortConnection_t</i>	Defines connections between two ports
<i>daeEventPortConnection_t</i>	Defines connections between two event ports
<i>daeEquation_t</i>	Defines model equations given in an implicit form
<i>daeSTN_t</i>	Defines state transition networks used to model discontinuous equations
<i>daeOnConditionActions_t</i>	Defines actions to be performed when a specified condition is satisfied
<i>daeOnEventActions_t</i>	Defines actions to be performed when an event is triggered on the specified event port
<i>daeState_t</i>	Defines a state in a state transition network
<i>daeModel_t</i>	Represents a model



## Package CORE - interface implementations



TabelaThe key concepts in the **ACTIVITY** package.

Concept	Description
<i>daeSimulation_t</i>	Defines a functionality used to perform simulations
<i>daeOptimisation_t</i>	Defines a functionality used to perform optimisations

TabelaThe key concepts in the **SOLVERS** package.

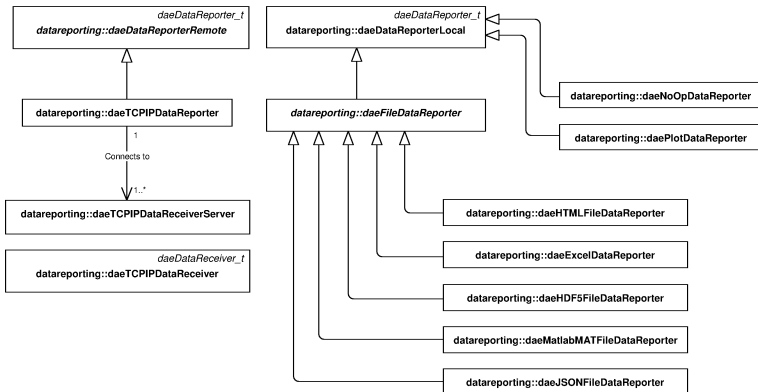
Concept	Description
<i>daeDAESolver_t</i>	Defines a functionality for the solution of DAE systems
<i>daeLASolver_t</i>	Defines a functionality for the solution of LA systems
<i>daeNLPSolver_t</i>	Defines a functionality for the solution of (MI)NLP problems
<i>daeIDALASolver_t</i>	Sundials IDAS LA solver interface
<i>daeMatrix_t</i> <typename <i>FLOAT</i> >	Defines a common matrix functionality



TabelaThe key concepts in the **DATAREPORTING** package.

Concept	Description
<i>daeDataReporter_t</i>	Defines a functionality/data structures used by a simulation to report the simulation results
<i>daeDataReceiver_t</i>	Defines a functionality/data structures for accessing the simulation results

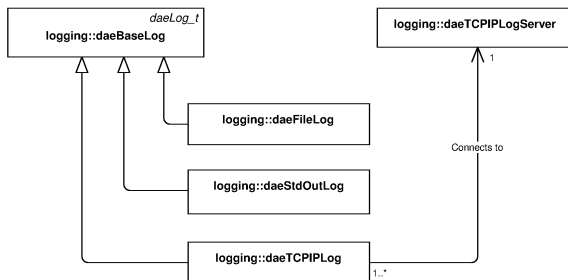
# Package DATAREPORTING - interface implementations



# Package LOG and its interface implementations

The key concepts in the **LOG** package.

Concept	Description
<i>daeLog_t</i>	Defines a functionality for sending messages from a simulation



TabelaThe key concepts in the **UNITS** package.

Concept	Description
<i>unit</i>	Defines SI base/derived units
<i>quantity</i>	Defines a numerical value in terms of a unit of measurement



# USE CASES

---

# Use Case 1 - Chemical Engineering

- CONTINUOUSLY STIRRED TANK REACTOR (Van de Vusse) [ref] [code]
- PLUG FLOW REACTOR [code]
- DISTILLATION COLUMN [ref] [code]
- BATCH CRYSTALLISER [ref] [code]
- DISCRETISED POPULATION BALANCE EQUATIONS [ref] [code1] [code2]
- NEWMAN POROUS ELECTRODE THEORY (PET) [ref] [code1] [code2]
- MULTIPHASE POROUS ELECTRODE THEORY (MPET) [ref] [code]
- HYDROXIDE EXCHANGE MEMBRANE FUEL CELLS (HEMFCs) [ref]
- MAXWELL-STEFAN EQUATIONS (porous membranes) [ref] [code]
- PRESSURE SWING ADSORPTION [ref1] [ref2]

# Use Case 2 - Finite Element Method

- TRANSIENT HEAT CONDUCTION/CONVECTION [code]
- CAHN-HILLIARD EQUATION [code]
- FLOW THROUGH THE POROUS MEDIA [code]
- DIFFUSION/REACTION IN AN IRREGULAR CATALYST SHAPE [code]

# Use Case 3 - Parameter Estimation & Optimal Control

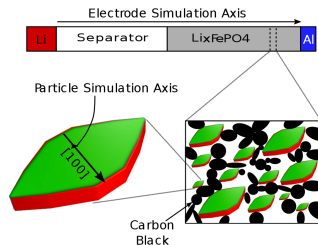
## LARGE-SCALE CONSTRAINED OPTIMIZATION PROBLEM SET (COPS)

- Determination of the reaction coefficients in the thermal isomerization of  $\alpha$ -pinene (COPS 5) [code]
- Determination of stage specific growth and mortality rates for species at each stage as a function of time (COPS 6) [code]
- Determination of the reaction coefficients for the catalytic cracking of gas oil and other byproducts (COPS 12) [code]
- Determination of the reaction coefficients for the conversion of methanol into various hydrocarbons (COPS 13) [code]
- Catalyst mixing in a tubular plug flow reactor (COPS 14) [code]

# Use Case 4 - Multi-scale modelling

## MULTI-SCALE MODEL OF PHASE-SEPARATING BATTERY ELECTRODES <sup>2</sup>

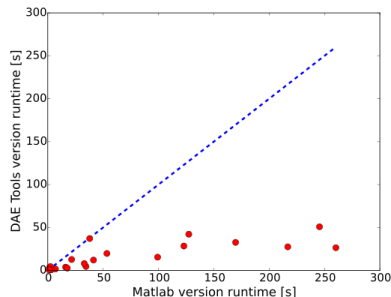
- Approach: **POROUS ELECTRODE THEORY**
- Lithium transport in:
  - Particles (small length scale)
  - Electrolyte (large length scale)
- Two phases are coupled via a volume-averaged approach
- Particles act as volumetric source/sink terms as they interact with the electrolyte via reactions



<sup>2</sup>Li et al. (2014) *Current-induced transition from particle-by-particle to concurrent intercalation in phase-separating battery electrodes*. Nature Materials 13(12):1149–1156. doi:10.1038/nmat4084.

# Use Case 4 - Multi-scale modelling (cont'd)

- Spatial discretisation: finite-volume method
- Large DAE system:
  - Discretised transport eqns.
  - Algebraic constraints (electrostatic eqns.)
  - Constraints on the current
- Implementations
  - MATLAB (ode15s solver)
  - DAE Tools (Sundials IDAS)
- DAE Tools up to 10x faster (average 4.22x) due to:
  - Built-in support for auto-differentiation
  - Rapid derivative evaluation
  - Accurate derivatives



# Use Case 5 - Embedded simulator (back end)

## NETWORK INTERCHANGE FORMAT FOR NEUROSCIENCE (NINEML)

XML-based DSL for modelling of networks of spiking neurones ↗

DAE Tools embedded into a **REFERENCE IMPLEMENTATION SIMULATOR**

### ○ ABSTRACTION LAYER (AL)

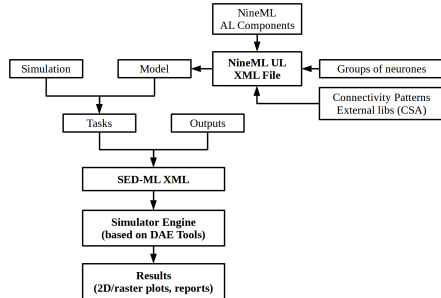
- Mathematical description
- Modelling concepts

### ○ USER LAYER (UL)

- Parameters values
- Instantiations

### ○ NineML concepts → DAE Tools concepts

- Neurone models
- Synapse models
- Populations of neurones
- Layers of neurones



# Use Case 6 - Web application / Web service

## NETWORK INTERCHANGE FORMAT FOR NEUROSCIENCE (NINEML)

DAE Tools serves as **AL COMPONENTS VALIDATOR/REPORT GENERATOR**

- Three flavours:
  - **DESKTOP APPLICATION** (Python + Qt GUI)
  - **WEB APPLICATION** (jQuery GUI)
  - **WEB SERVICE** with REST API (Apache server + Python WSGI)
- Inputs:
  - Abstraction Layer component to test
  - Parameters and inlet ports values, initial conditions
  - One or more tests (optional)
- Outputs:
  - Model report (pdf, html)
  - Test(s) results (variable plots)