DAE Tools Software

Introduction

Dragan D. Nikolić

Updated: June 14, 2023

DAE Tools Project, `http://daetools.sourceforge.io`

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

Outline

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

General Information

What is DAE Tools?

MODELLING, SIMULATION, OPTIMISATION & PARAMETER ESTIMATION software [1]

- 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, ...)

[1] 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
- Sensitivity analysis
  - Local methods (derivative-based)
  - Global methods (Morris, FAST, Sobol variance-based)
- Optimisation
  - Non-Linear Programming (NLP)
  - Mixed Integer Non-Linear Programming (MINLP)
- Parameter estimation
- Code-generation, model-exchange, co-simulation
  - Modelica, gPROMS, Functional Mockup Interface (FMI)
  - Matlab MEX-functions, Simulink user-defined S-functions
  - C99, C++ MPI (embedded and distributed systems)

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:

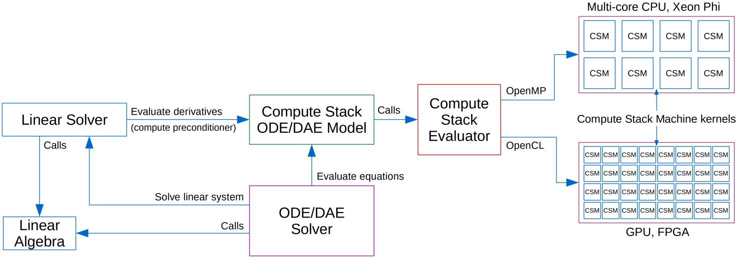
1. Support for the runtime model generation
2. Support for the runtime simulation set-up
3. Support for complex schedules (operating procedures)
4. Interoperability with the third-party software
5. Suitability for embedding and use as a web application or software as a service
6. Code-generation, model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

Main Features

Parallel Computing

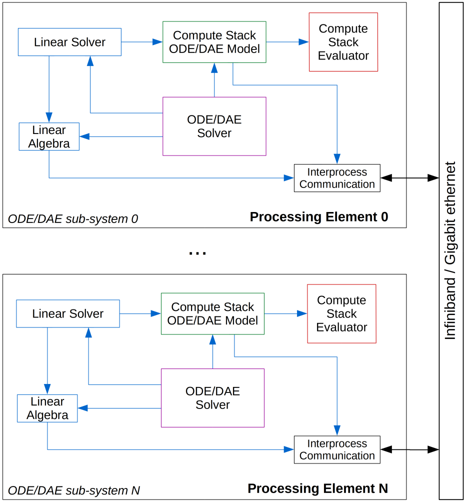
The SHARED-MEMORY parallel programming model:

- Evaluation of model equations using OpenCS framework
  - OpenMP (general purpose processors)
  - OpenCL (streaming processors/heterogeneous systems)
- Assembly of Finite Element systems (OpenMP)
- Solution of systems of linear equations (OpenMP)
- Global Sensitivity Analysis (multiprocessing.Pool)



Parallel Computing (cont'd)

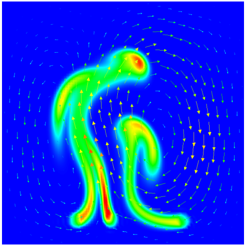
Simulation on MESSAGE-PASSING systems through OpenCS code generation.



Multiphysics Capabilities

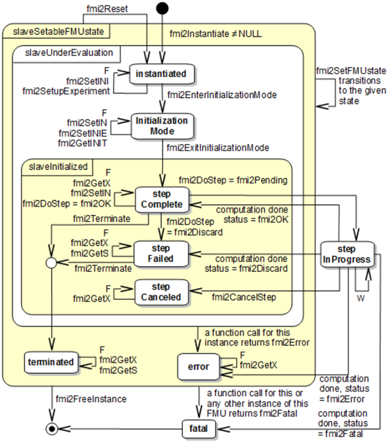
Modelling of multiple simultaneous physical phenomena

- Finite Difference (FD), Finite Volume (FV) and Finite Element (FE) methods
- Mixed coupled systems of equations (FD, FV and FE methods)
- DAE Tools variables for boundary conditions, source terms and non-linear coefficients
- Additional constraints and auxiliary equations



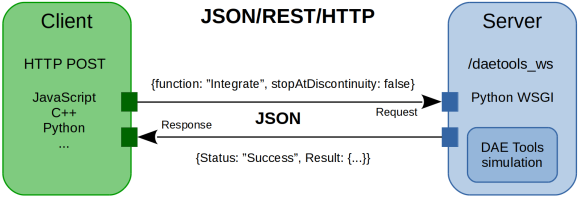
Code Generation & Co-Simulation

- **Code-generation**
  - Modelica
  - gPROMS
  - C99 (embedded systems)
  - C++ MPI (distributed systems)
- **Co-simulation**
  - Matlab MEX-functions
  - Simulink user-defined S-functions
  - Functional Mockup Interface (FMI) for Co-Simulation



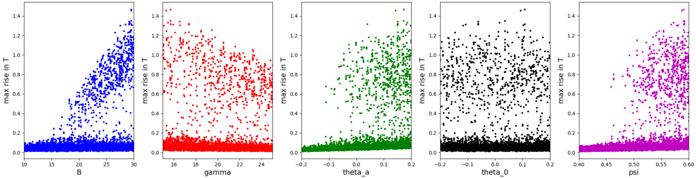
Software As a Service

- Web service with the RESTful API
  - DAE Tools simulations (daetools_ws)
  - DAE Tools FMU objects (daetools_fmi_ws)
- Language independent (JavaScript, Python, C++, ...)
- Benefits:
  - Application servers
  - Individual simulations as a web service
  - Attractive Graphical User Interface



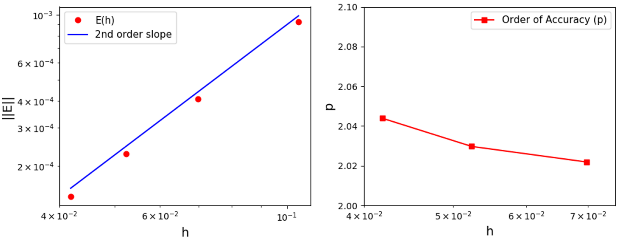
Sensitivity Analysis

- Local sensitivity analysis (derivative-based)
- Global sensitivity analysis (SALib library):
  - 1st and 2nd order sensitivities and confidence intervals
  - Total sensitivity indices and confidence intervals
  - Scatter plots
- Methods available:
  - Method of Morris (elementary effect method)
  - FAST (variance-based)
  - Sobol (variance-based)
- Simulations performed in parallel (multiprocessing.Pool)



Code Verification

- The FORMAL CODE VERIFICATION TECHNIQUES applied to test almost all aspects of the software
- The MOST RIGOROUS CODE VERIFICATION METHODS used:
  - The METHOD OF EXACT SOLUTIONS (MES)
  - The METHOD OF MANUFACTURED SOLUTIONS (MMS)
- The MOST RIGOROUS ACCEPTANCE CRITERIA used:
  - Percent error
  - Normalised global error
  - ORDER-OF-ACCURACY



Additional DAE Tools features

- Support for multiple platforms/architectures
- Support for the automatic differentiation (ADOL-C)
- 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

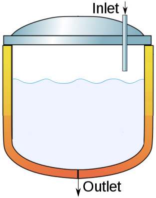
The HYBRID approach

- 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 and schedules
     - 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 betwee 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 betwee 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)

```python
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 programmaticaly and the initial values can be obtained from other software
Schedules limited to the options allowed by the langueage grammar	Schedules 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
- Models, simulations, optimisations:
  - 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:

$$x_1 = -x_2 - x_3, \quad \text{OR} \quad x_2 = -x_1 - x_3, \quad \text{OR} \quad x_3 = -x_1 - x_2$$

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 optimisation scenarios

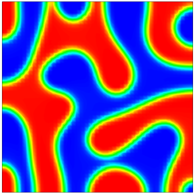
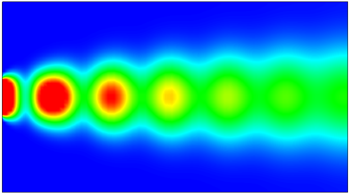
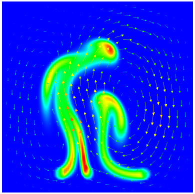
Use Cases

Use Case 1 - Chemical Engineering

- Continuously Stirred Tank Reactor (Van de Vusse)
- Plug Flow Reactor
- Distillation column
- Batch crystalliser
- Discretised Population Balance equations
- Newman Porous Electrode Theory (PET)
- Multiphase Porous Electrode Theory (MPET)
- Hydroxide Exchange Membrane Fuel Cells (HEMFCs)
- Maxwell-Stefan equations (porous membranes)
- Presssure Swing Adsorption

Use Case 2 - Finite Element Method

- Transient heat conduction/convection
- Cahn-Hilliard equation
- Flow through the porous media
- Diffusion/reaction in an irregular catalyst shape
- Stokes flow driven by the differences in buoyancy

Use Case 3 - Parameter Estimation & Optimisation

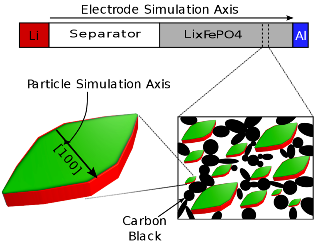
Large-scale Constrained Optimisation Problem Set (COPS)

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

Use Case 4 - Multi-scale modelling

Multi-scale model of phase-separating battery electrodes [2]

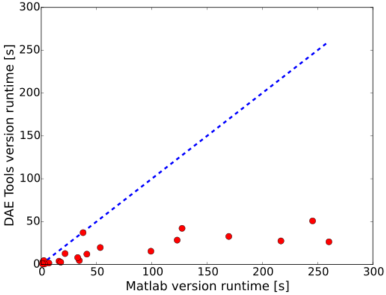
- Approach: porous electrode theory
- Lithium transport in:
  - Particles (small length scale)
  - Electrolyte (large length scale)
- Two phases are coupled via a volume-averaged approach
- Particles act as volumetric source/sink terms as they interact with the electrolyte via reactions
- The code available at Bitbucket



[2] 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)

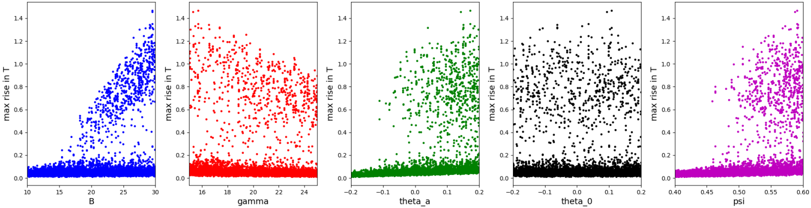
- Spatial discretisation: finite-volume method
- Large DAE system:
  - Discretised transport eqns.
  - Algebraic constraints (electrostatic eqns.)
  - Constraints on the current
- Implementations
  - MATLAB (ode15s solver)
  - DAE Tools (Sundials IDAS)
- DAE Tools up to 10x faster (average 4.22x) due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



Use Case 5 - Sensitivity Analysis

Thermal analysis of a batch reactor & exothermic reaction

- The global sensitivity analysis methods available via Python SALib library
- Three sensitivity analysis methods applied:
  - Morris (Elementary Effect/Screening method)
  - FAST and Sobol (Variance-based methods)
- Calculations can be performed in parallel (Python multiprocessing module)
- Available information:
  - 1^{st} and 2^{nd} order sensitivities and confidence intervals
  - Total sensitivity indices and confidence intervals
  - Scatter plots



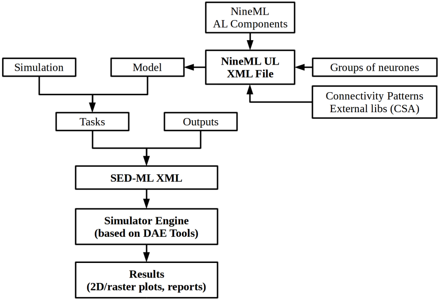
Use Case 6 - Embedded simulator (back end)

Network Interchange format for NEuroscience (NineML)

XML-based DSL for modelling of networks of spiking neurones

DAE Tools embedded into a reference implementation simulator

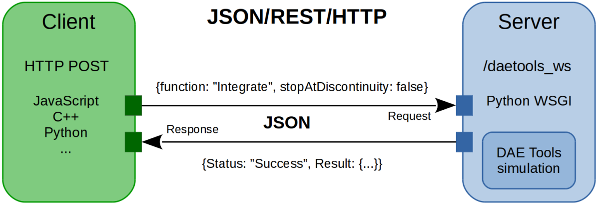
- Abstraction Layer (AL)
  - Mathematical description
  - Modelling concepts
- User Layer (UL)
  - Parameters values
  - Instantiations
- NineML concepts → DAE Tools concepts
  - Neurone models
  - Synapse models
  - Populations of neurones
  - Layers of neurones



Use Case 7 - Software as a service

DAE Tools simulations as a web service

- RESTful API (JavaScript, Python, C++, ...)
  - DAE Tools simulations (daetools_ws)
  - DAE Tools FMU objects (daetools_fmi_ws)
- Benefits:
  - Application servers or individual simulations as a service
  - Attractive Graphical User Interface



Use Case 7 - Software as a service (cont'd)

Sample HTML GUI (JavaScript + plotly.js plotting library):