DAE Tools Software

Introduction

D.D. Nikolić

Updated: 16 May 2016

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

Outline

1. General Information

2. Motivation

3. Programming Paradigms

4. Architecture

General Information

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

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**
  - Levenberg–Marquardt algorithm
- **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)

Programming Paradigms

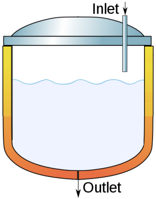
The HYBRID approach

- 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 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)

```
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
Simulation operating procedures limited to the options allowed by the langueage 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:

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

Separation of MODEL DEFINITION from its APPLICATIONS

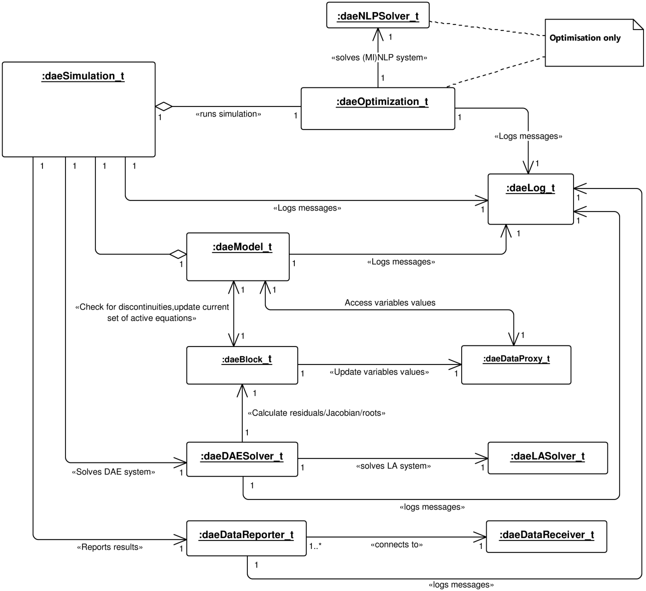
- Model structure specified in the model class
- Runtime information specified in the simulation class
- Solvers/auxiliary objects declared in the main program
- Single model definition, but one or more:
  - Different simulation scenarios
  - Different optimization scenarios

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



Package CORE

The key modelling concepts in the CORE package.

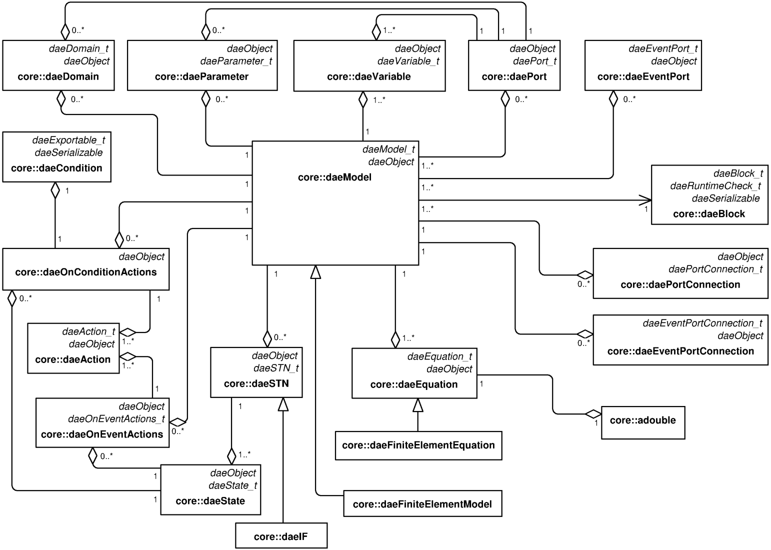
Concept	Description
daeVariableType_t	Defines a variable type that has the units, lower and upper bounds, a default value and an absolute tolerance
daeDomain_t	Defines ordinary arrays or spatial distributions such as structured and unstructured grids
daeParameter_t	Defines time invariant quantities that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

Package CORE (cont'd)

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

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

Package CORE - interface implementations



Package ACTIVITY

The key concepts in the ACTIVITY package.

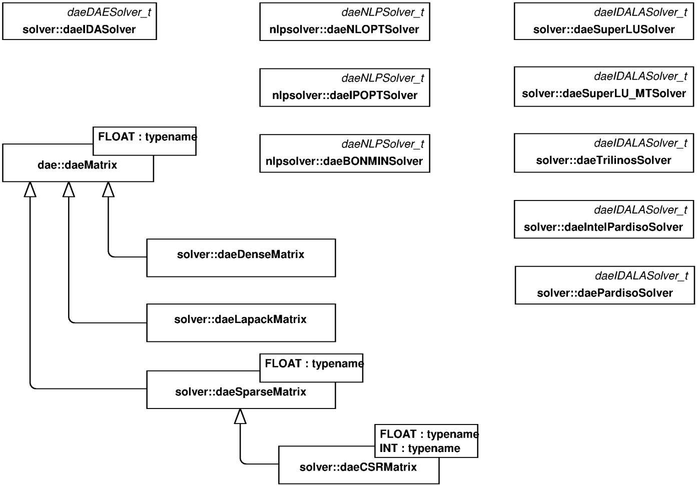
Concept	Description
daeSimulation_t	Defines a functionality used to perfom simulations
daeOptimisation_t	Defines a functionality used to perform optimisations

Package SOLVERS

The key concepts in the SOLVERS package.

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

Package SOLVERS - interface implementations

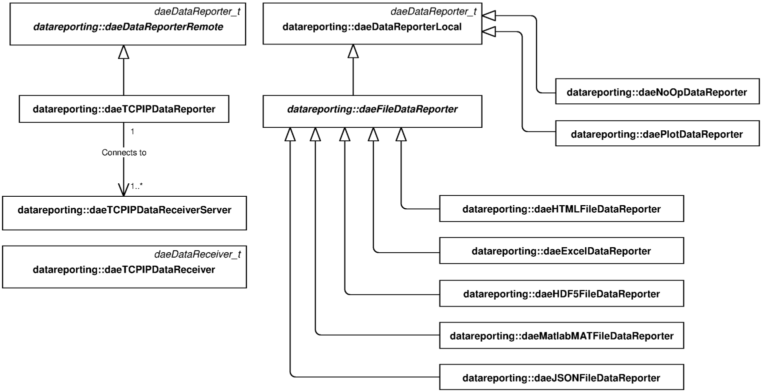


Package DATAREPORTING

The key concepts in the DATAREPORTING package.

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

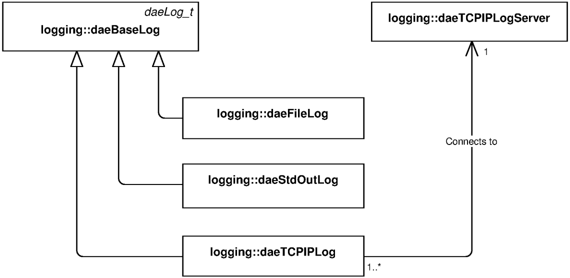
Package DATAREPORTING - interface implementations



Package LOG and its interface implementations

The key concepts in the LOG package.

Concept	Description
daeLog_t	Defines a functionality for sending messages from a simulation



Package UNITS

The key concepts in the UNITS package.

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