DAE Tools Software

Introduction

D.D. Nikolić

Updated: 1 April 2016

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

Outline

1. General Information

2. Motivation

3. Programming Paradigms

4. Architecture

5. Developing models with DAE Tools

6. Use Cases

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, gPROMS, ...)
  - An integrated software suite of data structures and routines for scientific applications (such as PETSc, Sundials, ...)
- DAE Tools IS:
  - A HYBRID APPROACH between modelling and general-purpose programming languages
  - An ARCHITECTURAL DESIGN OF 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

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?

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)
   - 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 features

- 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:
  - Applies general-purpose programming languages such as C++ and Python
- But provides:
  - Application Programming Interface (API) that resembles a syntax of modelling languages as much as possible
- And takes advantage of the higher level languages to:
  - Access the low-level functions in the operating system
  - Access a large number of standard and third-party libraries

The HYBRID approach (cont'd)

- To illustrate the HYBRID approach, consider a comparison between:
    - Modelica grammar
    - gPROMS grammar
    - DAE Tools API
- for a very simple dynamics model:
    - A cylindrical tank containing a liquid inside with an inlet and an outlet flow where the outlet flowrate depends on the liquid level in the tank

The HYBRID approach (MODEL. LANG. vs. DAE TOOLS)

gPROMS:

```
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);
```

Modelica:

```
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;
```

DAE Tools:

```
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())
```

The HYBRID approach (MODEL. LANG. vs. DAE TOOLS)

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	Programming language dependent
Cost of designing, implementing, and maintaining a language and a compiler/lexical parser/interpreter	A compiler/lexical parser/interpreter is an integral part of the programming language (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
Increased difficulty of integrating the DSL with other components	Calling external functions/libraries is a built-in feature
Models usually cannot be created/modified in the runtime/on the fly (or at least not easily)	Models can be created in the runtime/on the fly and easily modified in the runtime
Setting up a simulation is embedded in the language and it is typically difficult to do it on the fly or to obtain the values from other software	Setting up a simulation is done programmaticaly and the initial values can be obtained from other software
Simulation operating procedures are not flexible	Operating procedures are completely flexible (within the limits of a programming language itself)

The OBJECT-ORIENTED approach

- Everything is an OBJECT: models, parameters, variables, equations, simulations, solvers, ...
- Models, simulations, optimisations:
  - Classes derived from the corresponding base classes
  - Inherit the common functionality
  - Perform the required functionality in overloaded functions
- HIERARCHICAL MODEL DECOMPOSITION:
  - Models can contain instances of other models
  - Allows creation of complex, re-usable model definitions
  - Multi-scale modelling
- All C++/Python OBJECT-ORIENTED concepts supported, but:
  - Derived classes inherit all declared DAE Tools objects
  - All declared DAE Tools objects are public

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
- 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:

$$x_1 = -x_2 - x_3 \; or \; x_2 = -x_1 - x_3 \; or \; x_3 = -x_1 - x_2$$

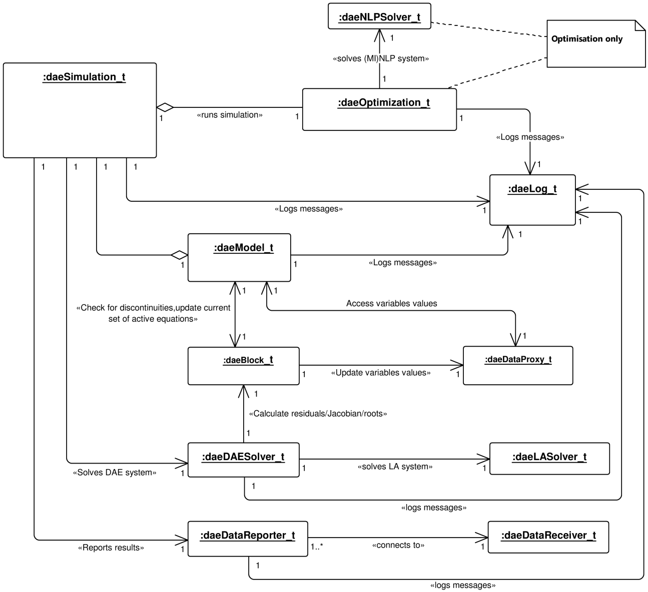
Separation of the model definition from its applications

- 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

ARCHITECTURE

The fundamental concepts/software interfaces

- Concepts/Interfaces:
  - 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.

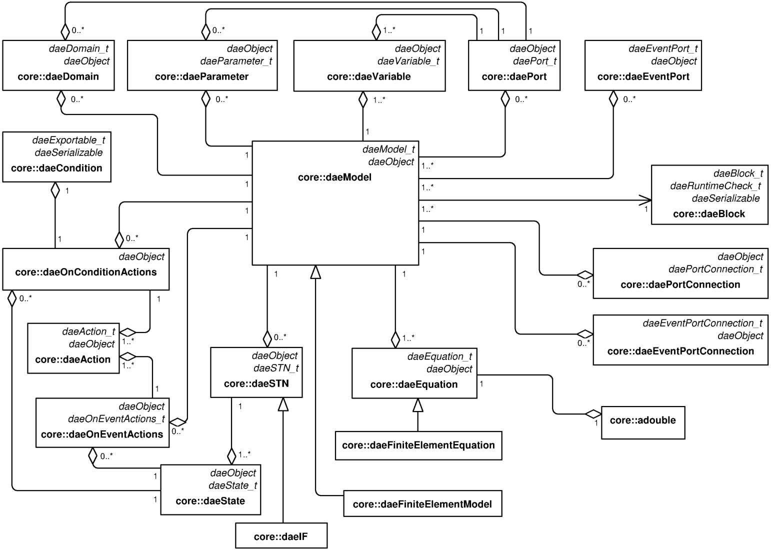
Concept	Description
daeVariableType_t	Defines a variable type that has the units, lower and upper bounds, a default value and an absolute tolerance
daeDomain_t	Defines ordinary arrays or spatial distributions such as structured and unstructured grids
daeParameter_t	Defines time invariant quantities that 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/acausal 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

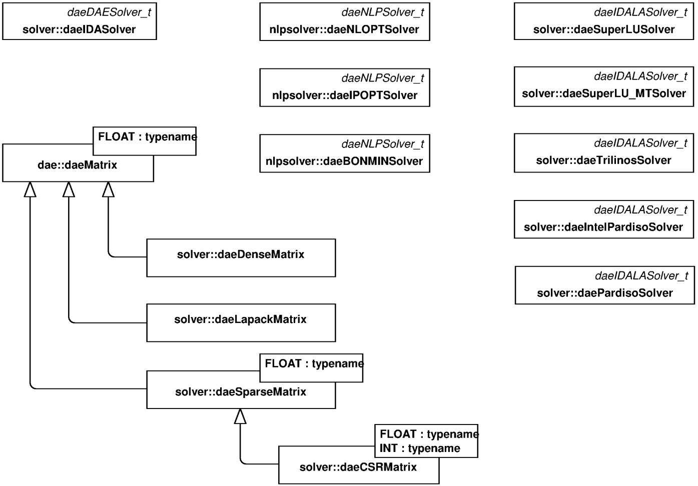
The key concepts in the ACTIVITY package.

Concept	Description
daeSimulation_t *daeOptimisation_t*	Defines ...

The key concepts in the SOLVERS package.

Concept	Description
daeDAESolver_t *daeLASolver_t* *daeNLPSolver_t* *daeIDALASolver_t*	Defines ...
daeMatrix_t<typename FLOAT>	

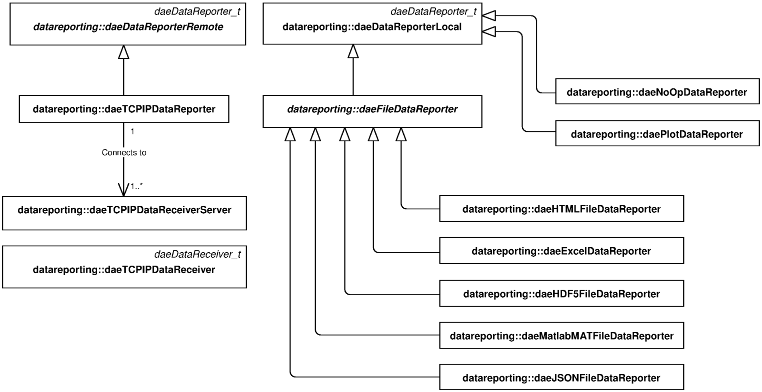
Package SOLVERS - interface implementations



The key concepts in the DATAREPORTING package.

Concept	Description
daeDataReporter_t *daeDataReceiver_t*	Defines ...

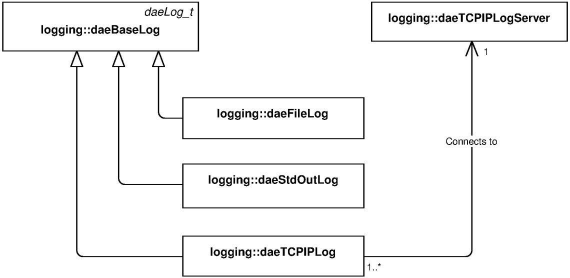
Package DATAREPORTING - interface implementations



Package LOG and its interface implementations

The key concepts in the LOG package.

Concept	Description
daeLog_t	Defines ...



Package UNITS

The key concepts in the UNITS package.

Concept	**Description**
unit	Defines ...
quantity	

Developing models with DAE Tools

Overview

Use Cases

Use Case 1 - High-Level Modelling Language

Use Case 2 - Low-Level DAE Solver

Use Case 3 - Embedded Simulator (back end)

Use Case 4 - Web Application / Web Service