DAE Tools Modelling, Simulation and Optimisation Software

Dragan D. Nikolić
http://daetools.sourceforge.io
DAE Tools Project, Belgrade, Serbia

Introduction

Many engineering problems can be described by a system of non-linear (partial-)differential and algebraic equations. Different modelling approaches can be applied to their solution[Morton] such as: (a) sequential modular, (b) simultaneous modular, and (c) equation-based (acausal). One of the methods to solve this type of problems is by using the equation-based approach. In the equation-based approach, all equations and variables which constitute the model representing the process are generated and gathered together. Then, equations are solved simultaneously using a suitable mathematical algorithm. Equations are given in an implicit form as functions of state variables and their derivatives, degrees of freedom (the system variables that may vary independently), and parameters.

This class of problems is found in the process, chemical, petrochemical, pharmaceutical and other engineering areas, natural sciences and financial systems. They are employed for tasks such as: *simulation*, *optimisation*, *parameter estimation*, *sensitivity analysis*, *model predictive* and *optimal control*, and *supply chain optimisation*.

In general, simulation programs for this class of problems are developed using:

- General-purpose programming languages such as C, C++ and Fortran and one of available suites for scientific applications such as SUNDIALS[Sundials], Trilinos[Trilinos] and PETSc[PETSc].
- Domain-specific and modelling languages such as Modelica[Modelica,Jmodelica,OpenModelica], Ascend[Ascend], gPROMS[gPROMS], GAMS[Gams], Dymola[Dymola] and APMonitor[APMonitor].
- Higher level fourth-generation languages such as Python and modelling software such as Assimulo[Assimulo].
- Multi-paradigm numerical languages: Matlab and Simulink[Matlab], Mathematica[Mathematica], Maple[Maple] and Scilab[Scilab].
- Computer Aided Engineering (CAE) software such as Aspen Plus[Aspen], EMSO Simulator[EMSO] and DESIGN II for Windows[WinSim].

The lower-level general purpose languages are also often used for the development of the efficient, tailor-made software (i.e. large-scale finite difference and finite element solvers) targeting one of the available high-performance computing architectures such as General Purpose Graphics Processing Units (GPGPU) and FieldProgrammable Gate Arrays (FPGA). Domain Specific Languages (DSL) are special-purpose programming or specification languages dedicated to a particular problem domain and directly support the key concepts necessary to describe the underlying problems. They are created specifically to solve problems in a particular domain and usually not intended to be able to solve problems outside it (although that may be technically possible in some cases). More versatile, multi-domain modelling languages (such as Modelica or gPROMS) are capable of solving problems in different application domains. Despite their versatility, modelling languages commonly lack or have a limited access to the operating system, third-party numerical libraries and other capabilities that characterise full-featured programming languages, scripting or otherwise. In contrast, general-purpose languages are created to solve problems in a wide variety of application domains, do not support concepts from any domain, and have a direct access to the operating system, low-level functions and third-party libraries. Higher level and multi-paradigm numerical languages provide an Application Programming Interface (API) to assist in the process of model specification and execution of different tasks (i.e. simulation, optimisation and parameter estimation). CAE software offer a great degree of generality: all required tasks are performed (mostly) automatically through the graphical user interface.

The most important tasks required to solve a typical simulation or optimisation problem include: the model specification, the simulation setup, the simulation execution, the numerical solution of the system of algebraic/differential equations, and the processing of the results. Each task may require a call or a chained sequence of calls to other software libraries, the methods in those libraries must be available to be called with no significant additional pre-processing and must be able to operate on shared/common data structures. All of these require a two-way interoperability between the software and third-party libraries. Also, the model structure is often not fully defined beforehand and a runtime generation of models ("on-the-fly") using the results from other software is required. Frequently, simulations can not be limited to a straightforward, step-wise integration in time but the custom user-defined schedules are required, which can be performed only using the fully-featured programming languages. In addition, it is often desired to compare/benchmark the simulation results between different simulators. This requires the code-generation and the model-exchange capabilities to automatically generate the source code for the target language or export the model definition to a specified (often simulator-independent) model specification language. Exposing the functionality of the developed models to another simulator through a predefined standard interface such as the CAPE-OPEN (http://www.colan.org) and Functional Mock-up Interface (FMI, http://www.fmi-standard.org) is another common functionality.

In general, a typical equation-based model consists of a coupled set of *partial-differential equations* and *auxiliary* algebraic and ordinary differential equations. Models are typically multi-scale (from the molecular to the overall plant level) and must be described by mixed/multiple coupled Finite Element/Finite Volume/Finite Difference equations with additional ordinary differential and algebraic equations. Such mixed non-linear systems of equations are difficult to model using libraries and Computer Aided Engineering software for finite element analysis and computational fluid dynamics.

A modelling language implemented as a single monolithic software package can rarely deliver all capabilities required. For instance, the Modelica modelling language allows calls to "C" functions from external shared libraries but with some additional pre-processing. Simple schedules are supported directly by the language but they must be embedded into a model, rather than separated into an independent section or function. gPROMS also allows very simple schedules to be defined as tasks (only in simulation mode), and user-defined output channels for custom processing of the results. The runtime model generation and complex operating procedures are not supported. Invocation from other software is either not possible or requires an additional application layer. On the other hand, Python, MATLAB and the software suites such as Trilinos and PETSc have an access to an immense number of scientific software libraries, support runtime model generation, completely flexible schedules and processing of the results. However, the procedural nature and lack of object-oriented features in MATLAB and absence of fundamental modelling concepts in all three types of environments make development of complex models or model hierarchies difficult.

Software description

DAE Tools is a cross-platform equation-based object-oriented modelling, simulation and optimisation software. It is not a modelling language nor a collection of numerical libraries but rather a higher level structure – an architectural design of interdependent software components providing an API for:

- Model development/specification.
- Activities on developed models, such as simulation, optimisation, sensitivity analysis and parameter estimation.
- Processing of the results, such as plotting and exporting to various file formats.
- Report generation.
- Code generation, co-simulation and model exchange.

The approach implemented in DAE Tools software offers some of the key advantages of the modelling languages coupled with the power and flexibility of the general-purpose languages. It is a type of hybrid approach–it is implemented using the general-purpose programming languages such as C++ and Python, but provides the Application Programming Interface (API) that resembles a syntax of modelling languages as much as possible and takes advantage of the higher level general purpose languages to offer an access to the operating system, low-level functions and large number of numerical libraries to solve various numerical problems. The combination of the features of modelling and general purpose programming languages in the Hybrid approach provides the following capabilities:

- Runtime model generation
- Runtime simulation set-up
- Complex schedules
- Interoperability with the third-party software
- Suitability for embedding and use as a web application or software as a service
- Code-generation, model exchange and co-simulation

Problems that can be solved are initial value problems of implicit form described by a system of linear, non-linear, and partial-differential equations (only index-1 DAE systems, at the moment). Systems modelled can be with lumped or distributed parameters, steady-state or dynamic, and continuous with some elements of event-driven systems such as discontinuous equations, state transition networks and discrete events. Automatic differentiation is supported through the operator overloading technique using the modified ADOL-C library [@ADOL-C].

Multiple activities can be performed on models developed in DAE Tools such as:

- Simulation (steady-state or dynamic, with simple or complex schedules).
- Sensitivity analysis (local or global methods).
- Optimisation (NLP and MINLP problems).
- Parameter estimation.
- Generation of model reports (in XML + MathML/LaTex format).
- Code generation for other modelling or general-purpose programming languages.
- Simulation in other simulators using standard co-simulation interfaces.
- Export of the simulation results to various file formats.

Main capabilities

DAE Tools is free software released under the GNU General Public Licence. Models can be developed in Python or C++. All DAE Tools libraries are written in standard ANSI/ISO C++. The code is therefore portable across different platforms, and currently runs on all major operating systems such as GNU/Linux, MacOS and Windows.

A large number of numerical solvers is supported. Currently, Sundials IDAS variable-order, variable coefficient BDF solver is used to solve DAE systems and calculate sensitivities. IPOPT[IPOPT], BONMIN[BONMIN], and NLopt[NLopt] solvers are employed to solve (mixed integer) non-linear programming problems, and a range of direct/iterative and sequential/multi-threaded sparse matrix linear solvers is interfaced such as SuperLU/SuperLU_MT[SuperLU], PARDISO[PARDISO], Intel PARDISO, and Trilinos Amesos/AztecOO[Amesos].

The hybrid approach allows an easy interaction with other software packages/libraries. First, other numerical libraries can be accessed directly from the code, and since the Python's design allows an easy development of extension modules from different languages, a vast number of numerical libraries is readily available. Second, DAE Tools are developed with a built-in support for NumPy (http://numpy.scipy.org) numerical package; therefore, DAE Tools objects can be used as native NumPy data types and numerical functions from other extension modules can directly operate on them. This way, a large pool of advanced and massively tested numerical algorithms is made directly available to DAE Tools.

Parallel computation is supported using the shared-memory parallel programming model at the moment. The following parts of the code support parallelisation: evaluation of equations, derivatives (Jacobian matrix or preconditioner) and sensitivity residuals using the OpenMP API or the OpenCL framework, assembly of Finite Element systems using the OpenMP API and solution of systems of linear equations.

Multiphysics capabilities Multiple simultaneous physical phenomena can be modelled using the finite difference, finite volume and finite element methods. DAE Tools utilise deal.II library to generate a set of differential equations for given inputs such as the mesh, the Finite Element space, the weak form of the problem and the boundary conditions. Several non-linear FE systems can be generated in the same model, they can be mixed with the other equations in the model, DAE Tools variables can be used to set boundary conditions, evaluate source terms and non-linear coefficients, impose constraints and add any number of auxiliary equations.

DAE Tools support local (derivative-based) and global sensitivity analysis (using SALib library). The global sensitivity analysis allows computation of the 1st and 2nd order sensitivities and confidence intervals, total sensitivity indices and confidence intervals and scatter plots. Three methods are available: method of Morris (elementary effect method) and variance-based FAST and Sobol methods. Simulations for a large number of samples are performed in parallel using the Python multiprocessing.Pool library.

DAE Tools also provide code generators and co-simulation/model exchange standards/interfaces for other simulators. This way, the developed models can be simulated in other simulators either by generating the source code, exporting a model specification file or through some of the standard co-simulation interfaces. To date, the source code generators for Modelica, gPROMS and OpenCS languages/frameworks have been developed. In addition, DAE Tools functionality can be exposed to MATLAB, Scilab and GNU Octave via MEX-functions, to Simulink via user-defined S-functions and to the simulators that support Functional Mock-up Interface (FMI) co-simulation capabilities.

DAE Tools can be run as a software as a service. Web services with the RESTful API are available for DAE Tools simulations and for FMI exported objects. The RESTful API is language independent and simulations can be executed using languages such as JavaScript, Python, C++ and many others. DAE Tools allow development of application servers or exposing individual simulations as a web service with an attractive Graphical User Interface.

The quality of the software is a very important aspect and the formal code verification techniques (such as the Method of Exact Solutions and the Method of Manufactured Solutions) and the most rigorous acceptance criteria (such as the order-of-accuracy) are applied to test almost all aspects of the software.

DAE Tools architecture and implementation details are presented in Nikolić[daetools].

References

[ADOL-C]: Walther A, Griewank A. 2012. Getting started with ADOL-C. Boca Raton: Chapman-Hall CRC Computational Science, 181–202. DOI 10.1201/b11644-8.

[Amesos]: Sala M, Stanley K, Heroux M. 2006. Amesos: a set of general interfaces to sparse direct solver libraries. In: Proceedings of PARA'06 Conference, Umea, Sweden. Available at https://www.researchgate.net/publication/220840090_Amesos_A_Set_of_General_Interfaces_to_Sparse_Direct_Solver_Libraries.

[APMonitor]: Hedengren JD, Shishavan RA, Powell KM, Edgar TF. 2014. Nonlinear modeling, estimation and predictive control in APmonitor. Computers & Chemical Engineering 70:133–148 (Manfred Morari Special Issue). DOI 10.1016/j.compchemeng.2014.04.013.

[ASCEND]: Piela PC, Epperly TG, Westerberg KM, Westerberg AW. 1991. ASCEND: an object-oriented computer environment for modeling and analysis: the modeling language. Computers & Chemical Engineering 15(1):53–72. DOI 10.1016/0098-1354(91)87006-U.

[Assimulo]: Andersson C, Fuhrer C, Akesson J. 2015. Assimulo: a unified framework for ODE solvers. Mathematics and Computers in Simulation 116:26–43. DOI 10.1016/j.matcom.2015.04.007.

[Aspen]: Aspen Technology, Inc., 2022. Aspen Plus. Available at https://www.aspentech.com.

[BONMIN]: Bonami P, Biegler LT, Conn AR, Cornue´jols G, Grossmann IE, Laird CD, Lee J, Lodi A, Margot F, Sawaya N, Wachter A. 2008. An algorithmic framework for convex mixed integer nonlinear programs. Discrete Optimization 5(2):186–204. DOI 10.1016/j.disopt.2006.10.011.

[daetools]: D. D. Nikolić, DAE Tools: Equation-based object-oriented modelling, simulation and optimisation software, PeerJ Computer Science, 2 (2016). https://doi.org/10.7717/peerj-cs.54.

[Dymola]: Elmqvist H. 1978. A Structured Model Language for Large Continuous Systems. Ph.D. thesis. Lund: Department of Automatic Control, Lund University.

[EMSO]: the ALSOC Project, 2022. EMSO simulator. Available at
http://www.enq.ufrgs.br/trac/alsoc/wiki/EMSO

[GAMS]: Brook A, Kendrick D, Meeraus A. 1988. GAMS, a User's Guide. SIGNUM Newsletter 23(3–4):10–11. DOI 10.1145/58859.58863.

[gPROMS]: Barton PI, Pantelides CC. 1994. Modeling of combined discrete/continuous processes. AIChE Journal 40(6):966–979. DOI 10.1002/aic.690400608.

[IPOPT]: Wachter A, Biegler LT. 2006. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. Mathematical Programming 106(1):25–57. DOI 10.1007/s10107-004-0559-y.

[JModelica]: Akesson J, Arzen K-E, Gafvert M, Bergdahl T, Tummescheit H. 2010. Modeling and optimization with Optimica and JModelica.org–languages and tools for solving large-scale dynamic optimization problems. Computers & Chemical Engineering 34(11):1737–1749. DOI 10.1016/j.compchemeng.2009.11.011.

[Maple]: Waterloo Maple, Inc. 2015. Maple. Waterloo: Waterloo Maple, Inc. Available at http://www.maplesoft.com/products/maple/.

[Mathematica]: Wolfram Research, Inc. 2015. Mathematica. Version 10.1. Champaign: Wolfram ResearchInc. Available at https://www.wolfram.com/mathematica.

[Matlab]: MathWorks, Inc. 2015. MATLAB. Matlab 8.5 (R2015a). Natick: MathWorks. Available at https://mathworks.com/products/matlab.

[Modelica]: Fritzson P, Engelson V. 1998. Modelica—a unified object-oriented language for system modeling and simulation. In: Jul E, ed. ECOOP'98—Object-Oriented Programming, Volume 1445 of Lecture Notes in Computer Science. Berlin Heidelberg: Springer, 67–90.

[Morton]: Morton W. 2003. Equation-oriented simulation and optimization. In: Proceedings of the National Academy of Sciences of India, Section A: Physical Sciences. New Delhi: Springer India 3:317–357. Available at http://www.dli.gov.in/data_copy/upload/INSA/INSA_1/2000c4d6_317.pdf.

[NLopt]: Johnson SG. 2015. The NLopt nonlinear-optimization package. Version 2.4.2. Available at http://ab-initio.mit.edu/wiki/index.php/NLopt.

[OpenModelica]: Fritzson P, Aronsson P, Lundvall H, Nystrom K, Pop A, Saldamli L, Broman D. 2005. The openmodelica modeling, simulation, and development environment. In: 46th Conference on Simulation and Modelling of the Scandinavian Simulation Society (SIMS2005), Trondheim, Norway, October 13–14, 2005. Oulu: SIMS–Scandinavian Simulation Society.

[Pardiso]: Schenk O, Wa¨chter A, Hagemann M. 2007. Matching-based preprocessing algorithms to the solution of saddle-point problems in large-scale nonconvex interior-point optimization. Computational Optimization and Applications 36(2):321–341. DOI 10.1007/s10589-006-9003-y.

[PETSc]: Balay S, Abhyankar S, Adams MF, Brown J, Brune P, Buschelman K, Dalcin L, Eijkhout V, Gropp WD, Kaushik D, Knepley MG, McInnes LC, Rupp K, Smith BF, Zampini S, Zhang H. 2015. PETSc users manual. Techical Report ANL-95/11–Revision 3.6, Argonne National Laboratory. Available at http://www.mcs.anl.gov/petsc/petsc-current/docs/manual.pdf.

[Scilab]: Scilab Enterprises. 2015. Scilab: free and open source software. Version 5.5.2. Versailles: Scilab Enterprises. Available at http://www.scilab.org.

[Sundials]: Hindmarsh AC, Brown PN, Grant KE, Lee SL, Serban R, Shumaker DE, Woodward CS. 2005. SUNDIALS: suite of nonlinear and differential/algebraic equation solvers. ACM Transactions on Mathematical Software 31(3):363–396. DOI 10.1145/1089014.1089020.

[SuperLU]: Li XS. 2005. An overview of SuperLU: algorithms, implementation, and user interface. ACM Transactions on Mathematical Software 31(3):302–325 DOI 10.1145/1089014.1089017.

[Trilinos]: Heroux, M. A., Bartlett, R. A., Howle, V. E., Hoekstra, R. J., Hu, J. J., Kolda, T. G., Lehoucq, R. B., Long, K. R., Pawlowski, R. P., Phipps, E. T., Salinger, A. G., Thornquist, H. K., Tuminaro, R. S., Willenbring, J. M., Williams, A., & Stanley, K. S. (2005). An overview of the Trilinos project. ACM Trans. Math. Softw., 31(3), 397–423. https://doi.org/10.1145/1089014.1089021

[WinSim]: WinSim Inc., 2022. DESIGN II for Windows. Available at https://www.winsim.com.