An XML Representation of DAE Systems Obtained from Modelica Models

Francesco Casella[a] Filippo Donida[a] Johan Åkesson[b,c]
[a]Dipartimento di Elettronica e Informazione, Politecnico di Milano
Via Ponzio 35/5, 20133 Milano, Italy
[b]Department of Automatic Control, Lund University, Lund, Sweden
[c]Modelon AB, Sweden

Abstract

This contribution outlines an XML format for representation of flat Modelica models. The purpose is to offer a standardized model exchange format which is based on the DAE formalism and which is neutral with respect to model usage. Many usages of models go beyond what can be obtained from an execution interface offering evaluation of the model equations. Several such usages arise in the area of control engineering, where Linear Fractional Transformations (LFTs), derivation of robotic controllers, optimization, and real time code generation are some examples. The choice of XML is motivated by its defacto standard status and the availability of free and efficient tools. Also, the XSLT language enables specification of transformation of the XML model representation into other formats.

Keywords: DAE representation; XML standard; modeling

1 Introduction

The Modelica language allows to build complex models of physical systems, described by differential-algebraic equations (DAE). These models can be used for different purposes: simulation, analysis, optimization, model transformation, control system synthesis, real-time applications and so forth. Each one of these activities involves a specific handling of the corresponding differential algebraic equations, by both numerical and symbolic algorithms. Moreover, specialized software tools which implement these algorithm may already exist, and only require the equations of the model to be input in a suitable way.

The goal of this paper is to define an XML-based representation of the DAEs of Modelica models, which can then be easily transformed into the input of such tools, e.g. by means of XSLT transformations. On one hand, this representation must be as close as possible to the mathematical equations, therefore without any aggregation, inheritance, and complex data structures left. On the other hand, it must be as general as possible with respect to the possible usage of the equations, which should not be limited to simulation.

This representation could then be used as a standard interface between the front-end of any Modelica compiler, and any possible back-end for simulation, optimization, analysis, etc.

In addition, the XML representation could also be very useful for treating other information concerning the model, for example using an XML schema (DTD or XSD) for representing the simulation results, or the parameter settings. In those cases, using a well-accepted standard will result in great benefits in terms of interoperability for a very wide spectrum of applications.

The paper is structured as follows: in Section 2, the abstract structure of the DAE representation is informally described, motivating the structure of the formal XML schema definition. Section 3 discusses some of the possible usages of such a representation. Section 4 briefly describes test implementations in the OpenModelica and JModelica.org compilers, while Section 5 ends the paper with concluding remarks and future perspectives.

2 Abstract representation of the DAE system

To the best of the authors' knowledge, the optimum representation for defining a DAE system should be as close as possible to the mathematical definition. Provided that a DAE system consists of a system of differential algebraic equations, it can be expressed as:

$$f(\dot{x}, x, y, u, v, t, p) = 0 \qquad (1)$$

where $\dot{x}$ is the derivative of state, x is the states, y are the outputs, u are the inputs, v are the algebraic variables, t is the time and p is the set of the parameters.

For the rest of this paper, we assume that the DAEs (1) describing the model have index 1. This restriction is necessary to give to the x variables the meaning of *states*, i.e., variables whose initial values can be arbitrarily selected. Most applications for DAE models (and all the applications discussed in this paper) require an index-1 DAE as input, so it is reasonable to discuss a representation limited to this class of equations. In case the equations of the original Modelica model have higher index, such DAEs can be obtained by symbolic index reduction, which is an available feature in most Modelica tools, so this is not a drastic limitation to the range of applicable models. In this case, however, we must assume that the index reduction procedure gives a fixed selection of states.

Even though the representation provided in equation 1 is very general, and is very appreciable for viewing the problem as one could see it written on paper, it cannot be directly used for inter-tools exchange in an efficient way.

The main idea is then to provide a standardized mathematical representation of the DAE system that relies on standard technology and is application-oriented. This justifies the adoption of the XML standard as the base framework. It can be noted that while XML is generally not suited for manual inspection, an XSLT transformation translating an XML description into, e.g., a flat Modelica representation is easily defined.

As an additional requirement, we must consider that the DAE systems we are dealing with are derived from the Modelica models. Even if this can be seen as a restriction, this is not, since the Modelica language specification interprets a superset of the problems that are object of this paper, providing a textual definition to describe the physical systems, concerning also the variables types and the expressions operators definitions.

Previous efforts have been registered to define standard XML-based representations of Modelica models, including [14, 15]. A standard representation of process engineering models is described in [2] and a standard for Modelica-derived simulation models is presented in [11]. In addition, a standard representation for API implementation for Modelica is given in [16], and standard representation for simulation libraries [9]. A recent initiative is the Functional Model Interface (FMI)[1], [6], which is aimed at creating a standard for a Modelica execution API.

The aim of this work is to take advantage from all this studies and try to define a simple and general representation which is not tailored for a particular usage, but rather aims at covering the largest possible problems that can be formulated starting from a Modelica model. Particular care has been exercised in order to define concepts and structures which are general enough to be usable in different contexts.

In the remainder of this section, the different parts of the proposed XML representation will be described.

2.1 Variables

The `Variables` entity corresponds to the set of scalar variables (real, integer, boolean) that are present in the equations of the DAE. In particular, taking into account the continuous-time representation (1), five types of variables are needed:

- Time-invariant, i.e., the constants and the parameters.
- Input variables, conceptually given from the outside.
- Algebraic variables corresponding to algebraic equations in the matching algorithm. This category also includes dummy derivatives obtained after index reduction. This set of variables could also be identified as the set of the time-variant variables not contained in the set of states or inputs.
- State variables.
- Derivatives of state variables.

As one could easily imagine, some constraints are present on the variables set. Firstly, a one-to-one relationship is defined between the set of the state variables and the set of derivatives. Secondly, the outputs are a subset of the state and algebraic variables, in the sense that the state and algebraic variables might also be marked as output variables, i.e., have an output attribute.

Associated with a variable is also a set of attributes, corresponding to the attributes specified by the Modelica language. These include the start attribute, min and

[1]The work on FMI is done within the ITEA2 project MODELISAR.

max, unit, nominal etc. These attributes are essential to include in a model specification format since they provide information about, for example, start values of variables, model validity regions and unit information.

Note that string parameters could also be considered, with constant binding to constant literal strings. The string binding equations will not be considered in the equation sets, because are irrelevant from a mathematical point of view.

2.2 Expressions

`Expressions` represent all the mathematical expression of the system and can be formed by aggregating identifiers and literals through:

- Unary operators. This class contains all the mathematical functions that require only one argument as input. Possible examples are the trigonometric functions (`sin`, `cos`, `tan`, `asin`, `acos`, `atan`), the hyperbolic functions (`sinh`, `cosh`, `tanh`), the exponential functions (`exp`), the logarithmic functions (`log`, `log10`), and the square root function (`sqrt`).
- Binary operators. This set contains all the algebraic operators like +, $\star$, −, /, the factor function ^, and the `atan2` function.
- Function calls referring to user-defined functions.

XML encoding of expressions is straightforward by introducing elements corresponding to an abstract grammar specification used in a compiler. This approach renders the DAE XML representation format to provide abstract syntax trees (ASTs) for expressions, which simplifies the development of XSLT transformations. Also, translating this representation into other formats for representing mathematical expressions, e.g., MathML, [5], would be trivial.

2.3 Functions

A `Function` is conceptually equivalent to an algorithm, i.e. a part of a procedural code and, in this sense has:

- Input variables (possibly with default values)
- Output variables
- Protected variables (i.e. variables visible only within the function)
- An algorithm to compute outputs from inputs, possibly using protected variables.

The DAE representation contains only scalar variables. If any vector or array variable is present within the original object oriented model, it is flattened to their fundamental scalar elements by the compiler before producing the XML. This is also the case for the other data structures (e.g. records) and for the functions. In particular, functions returning vectors or records are split into separate scalar functions, each one corresponding to the computation of a single scalar element of the outputs. For example, `(x,y)=f(u,v)` is converted to the scalar equations:

```
x1 = f1(u1,u2,v1,v2)
x2 = f2(u1,u2,v1,v2)
y1 = f3(u1,u2,v1,v2)
y2 = f4(u1,u2,v1,v2)
```

where each function `fj()` is defined in Modelica as the original function `f()`, save that all outputs except the j-th are declared as protected variables instead of outputs. The `fj()` functions should retain a reference to the original function `f()`, allowing an efficient computation scheme, where required. As an example, a simulation applications might cache the results of calling `f(u,v)` when encountering the first call to any `fj(u,v)`, and then use this to get the results of the other scalar functions calls with the same arguments, without re-executing the algorithm.

If the compiler performs function in-lining before generating the XML representation, the corresponding function calls disappear from the model; on the other hand, sophisticated in-lining of non-trivial functions could be performed by a post-processing tool, whose input is the XML code. The subject of in-lining is thus completely transparent and orthogonal to the DAE representation discussed here.

The algorithm in a function is formulated by an imperative language equivalent to Modelica algorithms, expressed as an XML translation of the corresponding abstract syntax tree.

The abstract syntax tree representation for functions is conceptually a superset of the `Expression` entity defined in subsection 2.2. More precisely, it is necessary to add three main classes of entities. Firstly, the program flow constructs (such as the `if-then-else`, the `for-loop`, the `while-loop` and others) are necessary. Secondly, there are some Modelica-specific constructs, e.g., `sample()` and `initial()`. Finally, the basic operators provided by a majority of programming languages, i.e., the boolean relation operators including <, <=, ==, <>, > and >=, and finally type conversion primitives such as `floor` and `edge`.

Again, XML elements are conveniently introduced to represent functions and related quantities. The advantages of having such kind of abstract representation are evident, for example, it can be easily converted to any imperative programming language (C, Matlab, Mathematica, Maxima, Maple, Scilab, Python, Java, etc) by means of XSLT transformations.

2.4 Equations

In many cases, a Modelica model includes parameters depending on other parameters. For simulation, it is necessary to solve the corresponding equations numerically at initialization. The numerical values can then be used for dynamic simulation. For other purposes, it may be necessary to keep some of these relationships in the model. For instance, in optimization problems there may be a free parameter `p1` and another parameter `p2 = f(p1)`. In this case, the parameter `p2` cannot be computed before the optimization procedure starts, but rather, the relationship needs to be included amongst the constraints in the optimization. When building LFT representations, `p1` might be an uncertain parameter, while `f()` might be well known; in this case, one wants to keep the dependency of `p2` from `p1` in the dynamic model.

When dealing with simulation problems, equations for parameters are conceptually part of the initialization section. However, they may play a special role in non-simulation problems, in particular when they all have fixed = true. In the case of LFT transformations there are no initial equations, but it is still necessary to consider the relationships between uncertain parameters and all other parameters when formulating the uncertain dynamic equations.

In fact, there is a whole class of problems for which the initial equations are irrelevant. As a first example, consider the LFT representation of an uncertain dynamical system. This system only involve the dynamic equations, and the initial values of the states are not required for the transformation. Also, when dealing with the derivation of inverse kinematics, computed torque and inverse dynamics in robot models, the resulting problems are purely algebraic: there are no initial equations involved once the appropriate BLT has been performed and the irrelevant parts of the model have been discarded.

However, there are still many problems where the values of initial states are an essential part of the problem. The initial variable values are generally not known, but need to be solved from the initial equations. There are also problems where additional initial equations are required to determine the values of some parameters (set with fixed=false). An additional example is given by the so-called trimming problems, where the values of the inputs are determined by prescribing certain steady values for the outputs.

Therefore, three separate sets of equations need be defined:

1. Dynamic equations. This set is composed of equations specified in equation sections and binding equations for variables. These equations are matched to algebraic variables (algebraic equations) and to state derivatives (differential equations). Each equation is given in residual form <expression> = 0.

2. Binding equations for parameters with fixed=true. These equations are matched to fixed=true parameters. The equations are in the form <parameter> = <expression>, and can be solved through assignments. The latter statement follows since it is illegal to define models with cyclic dependencies between parameters in Modelica.

3. Initial equations. This set is composed of equation given in initial equation sections and binding expressions for variables with fixed=true. These equations are matched to state variables, parameters with fixed=false, and possibly to inputs, if there are any (see example below). The initial equations should be in the form expression = 0.

```
model M
  input Real u;
  output Real y;
  Real x;
equation
  der(x) = -x + u;
  y = 4*x;
initial equation
  der(x) = 0; // Implies x(0) = u(0)
  y = 4; // This equation determines
         // x(0) = 1, and therefore
         // u(0) = 1;
end M;
```

Depending on the application, these three sets can be used in different ways, as will be discussed in Section 3.

2.5 Additional information

As introduced above, the range of applications that could directly use as input an XML DAE representation or any *ad hoc* description (derived from the more general one through XSLT transformation) is extremely variegated [3]. The common aspect is that the majority of tools for these usages require an index 1 DAE, as described in this section. Other information could be available from the Modelica tool, which could be relevant for some applications. For instance, information about the BLT structure of the dynamic simulation problem (compute the derivatives and algebraic variables, given the inputs, states, parameters, and time) could also be included, as well as information about the index reduction process in case the DAEs are the result of some index reduction algorithm such as [12]. This is however beyond the scope of the present paper.

3 Application examples

3.1 Simulation

The simulation tools are generally following the same approach. Firstly, the parameters and constants within each equation are numerically evaluated, by solving all the three equation sections together to determine the initial values of everything. After that, the numerical values of parameters are fixed, and the dynamic equations are used to compute derivatives and algebraic variables at each time step in an integration algorithm. This is also the case when considering the parallel simulation problem. Functions are the "linked" as external functions if any function calls is present for the state derivative computation.

3.2 LFT Transformation

LFT is a widely used model description formalism in modern control and system identification theory, in which uncertain parameters and non-linearities are "pulled out" from the system, resulting in the feedback connection between a linear, time-invariant model and blocks representing the uncertain and/or nonlinear parts. The procedure for obtaining an LFT representation from Modelica models is fully described in [4] and is only briefly summarized here. Assuming an ODE system, the values of the parameters are given by the binding equations, which specify the value of each parameter either by a numerical value, or as a function of other parameters. At each time instant, the values of states and inputs are known; the numerical values of the parameters are not known explicitly, but they can be considered as known, given the binding equations. The goal is now to compute the state derivatives $\dot{x}$ and the algebraic variables v. To this end, the equations and the variables of the problem can be re-ordered so that the incidence matrix (equations on the rows, unknowns on the columns) is brought in Block-Lower-Triangular (BLT) form. This task is accomplished by using the well-known Tarjan algorithm [7], applied to the equations-variables bipartite graph, which is equivalent to the incidence matrix of the system. The strongly connected components of the graph correspond to the blocks on the diagonal, and a partial ordering among equations can be deduced from the graph after the algorithm has terminated. After re-ordering, the system can be formulated as

$$\Phi(x, u, \Xi, p_0) = 0, \tag{2}$$

where $\Phi(\cdot)$ is the set of re-ordered equation residuals and Ξ is the re-ordered set of the system unknowns (i.e., all the elements of vectors $\dot{x}$ and v). By defining $\Phi_j(\cdot)$ as the j-th sub-set of equations corresponding to one block of the BLT form, Ξ_j as the corresponding sub-set of unknown variables, and q as the number of blocks on the diagonal of the BLT incidence matrix, the re-ordered system equations (2) can be formulated as

$$\Phi_1(x, u, \Xi_1, p_0) = 0 \tag{3}$$
$$\Phi_2(x, u, \Xi_1, \Xi_2, p_0) = 0 \tag{4}$$
$$\ldots \tag{5}$$
$$\Phi_q(x, u, \Xi_1, \ldots, \Xi_{q-1}, \Xi_q, p_0) = 0. \tag{6}$$

The system expressed in the form (3)-(6) can be executed within a suitable environment, which supports the symbolic manipulation of LFTs. Summing up, in the case of LFTs, the binding equations for parameters are solved by keeping the uncertain parameters as symbolic objects, and the resulting expressions are symbolically substituted in the dynamic equations; then, the relationship between the states and the inputs on one hand and the derivatives and the outputs on the other hand, is transformed into an LFT.

3.3 Derivation of robotic controllers

The design of controllers for robotic systems with N degrees of freedom usually starts with the equations of motion obtained from the Euler-Lagrange equations:

$$B(q)\ddot{q} + H(q,\dot{q})\dot{q} + E(q) = \tau \quad (7)$$

$$y_p = K(q) \quad (8)$$

$$y_v = \frac{\partial K}{\partial q}\dot{q}, \quad (9)$$

where q is the N-element vector of Lagrangian coordinates, which usually correspond to the rotation angles of the actuator motors, $\dot{q}$ is the vector of the corresponding generalized velocities, y_p is the vector of the Cartesian positions of selected points of the robot, τ is the vector of generalized applied forces corresponding to each degree of freedom (the torques applied by the actuators), $B(q)$ is the inertia matrix, $H(q,\dot{q})$ is the matrix corresponding to the centripetal, Coriolis, and viscous friction forces, $E(q)$ accounts for the effects of the gravitational field.

The classical approach to write (7) requires to compute the so-called direct kinematics, i.e. how the values of q and $\dot{q}$ translate into the position and motion of the robot's links, then to compute the Lagrange function, i.e. the difference between kinetic and potential energy, and apply the Euler-Lagrange equations. This can be done manually, or using one of the specialized tools available for this task.

With an object-oriented approach the original model is usually an index-3 DAE. This model is then brought into index-1 form

$$F(x,\dot{x},y,u) = 0 \quad (10)$$

where

$$x = \begin{bmatrix} x_p \\ x_v \end{bmatrix} = \begin{bmatrix} q \\ \dot{q} \end{bmatrix}, \quad y = \begin{bmatrix} y_p \\ y_v \end{bmatrix}, \quad u = \tau, \quad (11)$$

by means of connection tree analysis, change of state variables, and index reduction algorithms; this model is mathematically equivalent to the Lagrange model (7)-(9). Currently available Modelica tools solve the simulation problem by producing an efficient procedure to solve it for $\dot{x}$ and y given x and u. This procedure effectively brings the system into state-space form, which can then be linked to any ODE/DAE solver. In fact, there are other things that can be done with the model (10), which are very useful for the design of control system.

Robot trajectories are originally defined in Cartesian space as functions of time $y_p^0(t)$. Obtaining the corresponding reference trajectories in Lagrangian coordinates for the low-level robot joint controllers requires solving the problem to obtain the the so-called inverse kinematics:

$$q^0(t) = K^{-1}(y_p) \quad (12)$$

$$\dot{q}^0(t) = \left(\frac{\partial K}{\partial q}\right)^{-1} y_v; \quad (13)$$

the Jacobian of $K(q)$ is also needed to numerically invert (8). Furthermore, two interesting approaches to model-based robot control are based on the direct use of (7): the pre-computed torque approach and the inverse dynamics approach.

The pre-computed torque approach is a feed-forward compensation scheme, which requires to solve (7) backwards, i.e. compute the (theoretical) torque required to follow the reference trajectory:

$$\tau = B(q^0)\ddot{q}^0 + H(q^0,\dot{q}^0)\dot{q}^0 + E(q^0), \quad (14)$$

and then feed it directly to the actuators; some decentralized feedback action is also included to deal with uncertainties and disturbance.

The inverse dynamics approach is a feedback compensation scheme, that uses the model in order to transform the non-linear control problem into a linear problem with constant coefficients. Using this method, a virtual input variable v is defined which satisfies

$$\tau = B(q^0)v + H(q,\dot{q})\dot{q} + E(q). \quad (15)$$

Since the inertia matrix B is assumed to be structurally non-singular, it is always possible to solve (15) for $v(t)$, given the generalized velocities $q(t)$ and $\dot{q}(t)$, that are sensor outputs. Using this virtual input, the robot dynamics (7) can then be formulated as a set of double integrators:

$$\ddot{q} = v \quad (16)$$

For the robotic applications, the parameter binding equations are solved numerically; their numerical values are then substituted into the dynamic equations. Depending on the specific robotic problem: direct kinematic, inverse kinematic, pre-computed torques or inverse dynamics approach, these equations (or a part of these) are then solved.

3.4 Optimization

The needs for solving optimization problems based on Modelica models usually goes beyond what is typically offered by a simulation oriented execution API, see [1]. In some cases, the initial conditions are free variables in the optimization, which implies that the initial equations must be explicitly available. The same situation holds for dependent parameters since such a parameter may be dependent on another parameter which is free in the optimization. In addition, quantities derived from the model equations such as first and second order derivatives and sparsity patterns may be required by numerical algorithms.

The availability of a standardized XML-based model exchange format is useful in an optimization context since it enables transformation of a model into various formats suitable for different algorithms. Also, having access to expression syntax trees is useful for deriving derivatives, e.g., by means of automatic differentiation. It is worth noticing, however, that in order to completely specify an optimization problem, quantities such as cost function and constraints must also be taken into account. This is not part of the specification proposed in this paper. For a discussion on representation of optimization problems derived from Modelica models and Optimica specifications, see [10].

3.5 Real time code generation

Real time code could be directly generated by in-lining the discretization method within the equations of the XML file (e.g., forward or backward Euler), thus obtaining the core of the real time code. This could also be directly obtained from the XML formulation of the DAEs through the usage of an XSLT transformation if no symbolic manipulation is required. The simulation problem is then formulated by solving the dynamic equations for the next states and for the algebraic variables and then bringing it into BLT form.

If all the equations in the BLT form are linear, or can at least be solved explicitly in symbolic form, then it is straightforward to generate simulation code with fixed execution time. Otherwise, if there are implicit nonlinear equations, iterative solvers will be needed and there might be convergence problem that require proper handling.

4 Test implementations

There are two prototype implementations available. The implementation of the XML module within the OpenModelica compiler started the past year and is now included in the latest release of the compiler. This functionality allows dumping of a flattened Modelica model after performing the index reduction (if necessary), the BLT transformation and the matching algorithm. The API method provided by the OpenModelica compiler offers the possibility to specify several inputs parameters, such as if to add or not the information for solving the system, and if to dump the equations as residuals or add MathML representation for all the equations. This XML schema is available at [8].

The JModelica.org platform currently supports generation of variable meta data, as described in Section 2.1, in XML format, see [13, 10]. It is intended that this functionality is extended to include also equations an functions as well as cost functions and constraints for optimization.

As for the actual specification of a DAE XML schema, the objective is to build on what is done in the FMI initiative concerning model meta data and to merge this with the existing schema [8].

5 Conclusions and future perspectives

In this paper, we have outlined an XML representation of DAEs. This will allow easy coupling of Modelica compiler front-ends with diverse application back-ends that require the system equations as inputs. This format is not limited to Modelica models, but could be used as a lingua franca to represent continuous-time dynamical systems originally described with other modelling languages, such as, e.g., gPROMS or VHDL-AMS. This would allow developers of application back-ends (e.g. for optimal controller generation) to support multiple modelling platform easily.

The proposed format does not support all features of Modelica. Notably, description of hybrid constructs are lacking. However, there is a number of interesting applications where this is not needed, as demonstrated by the examples outlined in this paper.

The XML description format outlined in this paper might be extended in several respects. First of all, support for discontinuous expressions (e.g. *if-expressions*) could be added, possibly by including an explicit representation of the root functions that many tools need in order to handle discontinuities properly. In order to support hybrid models, it would also be necessary to introduce the concepts of discrete variables, discrete equations (those within *when* statements), time events and state events. Another extension might be to support variables declared as vectors and array equations, without reducing all equations to their scalar form; this might be useful for sophisticated symbolic processing at the vector level. Support of index-1 models with dynamic sets of states (such as those resulting from the dummy derivative algorithm [12] in some cases) might be added, as well as support for the description of higher index models. Finally, it would be interesting to investigate how this kind of formalism could be employed to describe sub-models that could then be aggregated at a higher level, by introducing some kind of connector concept; this might allow some form of separate compilation strategy, at least for a certain class of problems that do not lead to higher index DAEs when connecting the submodels.

References

[1] Johan Åkesson. Optimica—an extension of modelica supporting dynamic optimization. In *In 6th International Modelica Conference 2008*. Modelica Association, March 2008.

[2] Christian H. Bischof, H. Martin Bücker, Wolfgang Marquardt, Monika Petera, and Jutta Wyes. Transforming equation-based models in process engineering. In H. M. Bücker, G. Corliss, P. Hovland, U. Naumann, and B. Norris, editors, *Automatic Differentiation: Applications, Theory, and Implementations*, Lecture Notes in Computational Science and Engineering, pages 189–198. Springer, 2005.

[3] F. Casella, F. Donida, and M. Lovera. Beyond simulation: Computer aided control system design using equation-based object oriented modelling for the next decade. In *2nd International Workshop on Equation-Based Object-Oriented Languages and Tools*, July, 8 2008.

[4] F. Casella, F. Donida, and M. Lovera. Automatic generation of lfts from object-oriented non-linear models with uncertain parameters. In *6th Vienna International Conference on Mathematical Modeling*, February, 11-13 2009.

[5] D. Suliman D. Draheim, W. Neun. Searching and classifing equations on the web, zib report 04-22. Technical report, Konrad-Zuse-Zentrum für Informationstechnik, Berlin, 2004.

[6] DLR, Dynasim, ITI and QTronic. The functional model interface. Draft.

[7] I. S. Duff and J. K. Reid. An implementation of Tarjan's algorithm for the block triangularization of a matrix. *ACM Transactions on Mathematical Software*, 4(2):137–147, 1978.

[8] Filippo Donida. DAE XSD schema, 2009. `http://home.dei.polimi.it/donida/Projects/AutoEdit/Images/DAE.xsd`.

[9] P. A. Fishwick. Using xml for simulation modeling. In *Winter simulation conference*, December, 8-11 2002.

[10] J. Åkesson, T. Bergdahl, M. Gäfvert, and H. Tummescheit. The JModelica.org Open Source Platform. In *7th International Modelica Conference 2009*. Modelica Association, 2009.

[11] J. Larsson. A framework for simultion-independent simulation models. *Simulation*, 82(9):563–379, 2006.

[12] S. E. Mattsson and G. Söderlind. Index reduction in differential-algebraic equations using dummy derivatives. *SIAM Journal on Scientific Computing*, 14(3):677–692, 1993.

[13] Modelon AB. JModelica Home Page, 2009. `http://www.jmodelica.org`.

[14] A. Pop and P. Fritzson. Modelicaxml: A modelica xml representation with applications. In *3rd Modelica conference*, November, 3-4 2003.

[15] U. Reisenbichler, H. Kapeller, A. Haumer, C. Kral, F. Pirker, and G. Pascoli. If we only had used xml... In *5th Modelica conference*, September, 4-5 2006.

[16] M. Tiller. Implementation of a generic data retrieval api for modelica. In *4th Modelica conference*, March, 7-8 2005.