Sensitivity Analysis of Modelica Applications via Automatic Differentiation

Elsheikh, Atya[1] Noack, Stephan[2] Wiechert, Wolfgang[1]
[1] Siegen University, Department of Simulation, {elsheikh,wiechert}@simtec.mb.uni-siegen.de
[2] Research Center Jülich GmbH, Institute of Biotechnology 2, s.noack@fz-juelich.de

Abstract

Modeling and simulation of physical systems is, in general, a complex iterative process. Asserted models are necessarily based on simplifications, and in many cases are subject to improvement and optimization. In this context, a wide range of applications of sensitivity analysis can assist the modeling process, from parameter fitting and optimization through model validation to statistical analysis and experimental design. These common methods, among others, drew increasing attention to a research area of scientific computing, i.e. Automatic Differentiation (AD) of program code. The main objective of this work is to compute derivatives of variables in Modelica models using AD concepts to assist sensitivity analysis applications. It is shown how Open Modelica Compiler (OMC) and other tools simplify the implementation of ADModelica, a prototype of an AD-based tool for Modelica. As a proof of concept, an application in the field of biochemical networks is presented.

Keywords: Sensitivity Analysis, Automatic Differentiation, Open Modelica, Biochemical Networks

1 Introduction

AD is a methodology that refers to algorithmic techniques for semantic augmentation of numerical programs with additional code for derivative computations [6]. For many reasons, AD is a better choice over other ways for computing derivatives such as symbolic differentiation and finite difference methods. In contrast to symbolic differentiation tools, an AD tool does not generate the derivative formula explicitly, but it computes the numerical values of efficient derivative formulas expressed as a program. Nevertheless, the derivative values using AD are as accurate as the values of those generated by symbolic algebra packages up to machine precision. Furthermore, the results are not affected by any truncation errors, resulting from numerical differentiation using divided difference methods.

This work is concerned with AD of Modelica models. Modelica is essentially targeted towards modeling complex systems that can be described by differential algebraic equation (DAE) systems:

$$F(t,x,\dot{x},p) = 0, \;\; x(0) = x_0(p) \tag{1}$$

where $x \in \mathbb{R}^n$, $p \in \mathbb{R}^m$, $F : \mathbb{R}^{2\cdot n+m+1} \rightarrow \mathbb{R}^n$. Assuming that $\partial F/\partial x$ is non-singular for all $p \in \mathbb{R}^m$, and that $\partial x/\partial p$ is smooth enough, sensitivity analysis requires the sensitivities $\partial x/\partial p$ of solution variables with respect to perturbations in the parameters. These can be calculated by solving the original DAE system (1) and m sensitivity systems:

$$\begin{aligned} \frac{\partial F}{\partial \dot{x}} \cdot \frac{\partial \dot{x}}{\partial p} + \frac{\partial F}{\partial x} \cdot \frac{\partial x}{\partial p} + \frac{\partial F}{\partial p} = 0, \\ \frac{\partial x}{\partial p}(0) = \frac{\partial}{\partial p}(x_0(p)) \end{aligned} \tag{2}$$

obtained by explicit differentiation of (1) with respect to p [14]. Additionally, the sensitivities $\partial x_i/\partial x_j$ of certain variables x_i with respect to other specific variables x_j might be needed.

This paper presents first experiences with a prototype of a tool, ADModelica, that augments Modelica models with Modelica code for computing certain sensitivities, with minimal user efforts. Aiming at the full-support of Modelica language constructs, we implemented a first version, which supports most basic constructs of Modelica. The rest of the paper is structured as follows. Section 2 introduces basic terminologies and algorithmic aspects of AD. The Generalization of the introduced concepts into the Modelica framework is clarified in Sect. 3. Section 4 presents the ADModelica tool and briefly discusses some design and implementation issues. In Sect. 5, applications in the field of Biochemical Engineering using a special library is presented. Finally, conclusions are presented and future work is discussed in Sect. 6.

2 Introduction to Automatic Differentiation

Many techniques such as numerical differentiation or computer algebra methods are used to compute derivatives. However, AD has proved to be superior over other ways for obtaining derivatives in terms of computational efficiency, numerical precision and discretization parameters. ADIC [2] and ADIFOR [1] are examples of a wide range of AD tools for differentiating C and Fortran programs respectively. In this section, some basic terminologies of AD are introduced.

2.1 Basic Concepts

Formally, given a program P that computes a function:

$$f : x \in \mathbb{R}^n \rightarrow y \in \mathbb{R}^m$$

with n inputs and m outputs, a new code P' is sought to compute the Jacobian $f' = \partial y / \partial x$. The following terms are commonly used in the context of AD:

- *Independent variables* are program input variables with respect to which derivatives are sought.
- *Dependent variables* are output variables whose derivatives are desired.
- A *derivative object* represents some derivative information, such as a vector of partial derivatives $(\partial z/\partial x_1, ..., \partial z/\partial x_n)^T$ of a variable z with respect to a vector $x = (x_1, x_2, ..., x_n)^T$.
- Any program variable with which a derivative object is associated is called an *active variable*.

2.2 Algorithmic Aspects of AD

The key concept behind AD is that every computation, no matter how complex it is, is executed on a computer as a sequence of a limited set of elementary operations, such as addition and multiplication, and intrinsic functions, such as sine and cosine. The derivative of each of these elementary operations can be computed by applying the chain rule to combine the local partial derivatives of each executed operator. An AD tool operates by systematic application of the chain rule on the numerical code. For example, let $a(x)$ and $b(x)$ be intermediate values that depend on an independent variable x, and let $c := f(a,b)$. Then by using the chain rule, $\nabla_x c$ the derivative of the dependent variable c with respect to x is computed as:

$$\nabla_x c := \frac{\partial f}{\partial a} \cdot \nabla_x a + \frac{\partial f}{\partial b} \cdot \nabla_x b \qquad (3)$$

The chain rule is associative. If $y := f(g(x))$, $\partial y/\partial x$ can be computed by forwardly accumulating the derivatives (i.e. $\partial f/\partial g$ and $\partial g/\partial x$) in the computational path from the independent variable(s) (eg. x) to the dependent variable(s) (eg. y). By exploiting the associativity of the chain-rule, the augmented program is generated to evaluate $f(x)$ and the partial derivatives of f simultaneously.

2.3 Why AD for Modelica?

AD is naturally implemented by Modelica compilers to provide partial derivatives of functions for solving the DAE index problem [12]. A DAE system of high index is transformed into a solvable ODE system by differentiating some equations selected by Pantelides's algorithm [13]. Here, AD is chosen for the fundamentally different task of calculating sensitivities of solution variables, motivated by the following reasons:

- DAE systems are represented in Modelica by using components and connectors; internal formulas in components and models may be implemented with loops and many branches. Therefore, it makes sense to utilize existing tools and concepts of handling DAE systems, used by modelica compilers, for generating derivative formulas.
- For a Modelica model that computes a DAE System (1), a lot of common sub-expressions in F, $\partial F/\partial x$ and $\partial F/\partial p$ arise. In many cases, these common sub-expressions need not to be reevaluated if these partial derivatives are computed using AD.
- Compiler techniques used for reducing the dimension of a generated DAE system, can be adopted by AD for reducing the number of equations needed to be differentiated , instead of blind differentiation of all equations, as the DAE system (2) suggests [4].

3 Differentiating DAE Systems

Assignments (eg. $x := f(y,z)$) are the main elementary units of procedural languages, whereas declarative equations (eg. $f(x(t),y(t),z(t)) = 0$) constitute the main building units of Modelica. While an assignment is a relation between inputs (a collection of values) and one output, an equation is a relation between several variables, that needs to be fulfilled concurrently. This conceptual difference has vital consequences on the way derivatives can be generated for DAE systems, namely, AD techniques for classical languages, such as C/FORTRAN, are not necessarily applicable for equation-based languages.

3.1 Example

Consider the DAE System

$$\begin{array}{l} \dot{A} = -v, \quad A(0) = A_0 \\ \dot{B} = v, \quad B(0) = B_0 \\ v = v_{max} \cdot \frac{A}{A+k} \cdot \frac{I_k}{B+I_k} \end{array} \tag{4}$$

describing the dynamics of a chemical reaction, in which a chemical substance with concentration $A = A(t)$ is converted to another chemical substance with concentration $B = B(t)$. $v = v(A,B,t)$ stands for reaction rate and v_{max}, k and I_k stand for enzymatic parameters. The first two ordinary differential equations represent balance equations, whereas the third equation describes the reaction rate using the well-known Michaelis-Menten Kinetics [7]. The sensitivities of $x = (A,B,v)^T$ w.r.t. parameters $p = (v_{max},k,I_k)^T$ can be computed as in (2) by adding the following equations:

$$\begin{array}{l} \dot{A_p} = -v_p, \quad A_p(0) = 0 \\ \dot{B_p} = v_p, \quad B_p(0) = 0 \\ v_p = \frac{\partial}{\partial p} f(A,B,v_{max},k,I_k) \end{array} \tag{5}$$

to (4), where

$$f(A,B,v_{max},k,I_k) = v_{max} \cdot \frac{A}{A+k} \cdot \frac{I_k}{B+I_k} \tag{6}$$

$$v_p = \nabla_p v = (\frac{\partial v}{\partial v_{max}}, \frac{\partial v}{\partial k}, \frac{\partial v}{\partial I_k})^T \tag{7}$$

and A_p, B_p are similar to v_p. Given that $J_p = I_3$ (Identity matrix of size 3), i.e.:

$$\begin{array}{ll} \nabla_p v_{max} & := (1,0,0)^T; \\ \nabla_p k & := (0,1,0)^T; \\ \nabla_p I_k & := (0,0,1)^T; \end{array} \tag{8}$$

(5) can be easily implemented in Modelica with the help of arrays. Notice that := stands for assignments.

3.2 Utilizing Common Sub-expressions

Given that the values of $A(t)$ and $B(t)$ are known for a time point t, $v(t)$ and $v_p(t)$ can be computed from the DAE systems (4) and (5). The third equation $v_p = \partial f/\partial p$ in the DAE system (5) consists of three equations of similar algebraic structure. Excessive re-evaluation of common sub-expressions arising in v and v_p can be avoided by dividing the equation $v = f$ in the DAE system (5) into a set of binary assignments using the Abstract Syntax Tree (AST) of v as shown in Fig. 1. The gradient of $v(t)$ is computed by forward accumulation of the gradients of the intermediate variables obtained by differentiating each assignment instead of direct differentiation of the algebraic formula. An implementation for the DAE systems (4) and (5) looks as follows:

$$\begin{array}{l} \dot{A} = -v \\ \quad \frac{\partial}{\partial t}\nabla_p A = -\nabla_p v \\ \dot{B} = v \\ \quad \frac{\partial}{\partial t}\nabla_p B = \nabla_p v \\ \\ u_1 := v_{max} \cdot A; \\ \quad \nabla_p u_1 := \nabla_p v_{max} \cdot A + v_{max} \cdot \nabla_p A; \\ u_2 := A + k; \\ \quad \nabla_p u_2 := \nabla_p A + \nabla_p k; \\ u_3 := u_1 \cdot u_2; \\ \quad \nabla_p u_3 := \nabla_p u_1 \cdot u_2 + u_1 \cdot \nabla_p u_2; \\ u_4 := B + I_k; \\ \quad \nabla_p u_4 := \nabla_p B + \nabla_p I_k; \\ u_5 := I_k / u_4; \\ \quad \nabla_p u_5 := (\nabla_p I_k \cdot u_4 - I_k \cdot \nabla_p u_4)/u_4^2; \\ v := u_3 \cdot u_5; \\ \quad \nabla_p v := \nabla_p u_3 \cdot u_5 + u_3 \cdot \nabla_p u_5; \end{array} \tag{9}$$

In this way, common sub-expressions are evaluated only once, and hence less arithmetic operations are needed. The assignments can be implemented in Modelica with the help of the *algorithm* construct.

3.3 Limitations

While optimizing common sub-expressions works well for AD of classical procedural languages, this may not be the case with equation-based languages. For example, in the DAE system (4), $v(0)$ can be computed by considering the available values of $A(0)$ and $B(0)$. Then, $v(0)$ is used to compute subsequent values of A and B, and hence forth. That is, at each iteration, $A(t)$ and $B(t)$ are used to compute $v(t)$. In other words, the values $v(t)$ *depends on* $A(t)$ and $B(t)$. By this way, computing $v(t)$ from $A(t)$ and $B(t)$

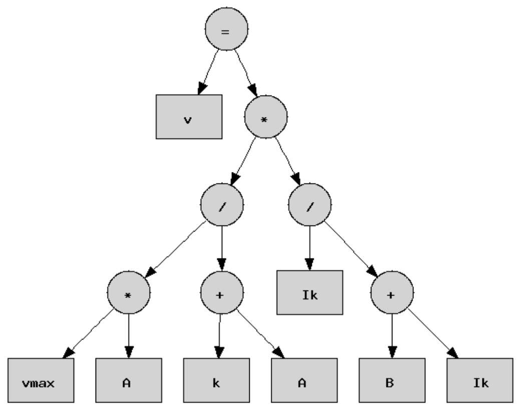


Figure 1: Abstract Syntax Tree (AST) of $v = f$

in (9) does not change the dependency of variables. However, in general, an equation can be divided into a set of binary operations if the output variable depends on the variables arising in the left hand side of all intermediate assignments.

Additionally, the dimension of the rewritten DAE system increases according to the way the Modelica compiler handles local variables. If intermediate results of local variables are always stored, this exhausts extra storage and computation time. Note that, the number of local variables can be reduced by reusing local variables. For example, there is no need to introduce new local variables u_4 and u_5 if u_1 and u_2 are used instead. Moreover, excessive use of the *algorithm* section may disable some optimization methods for reducing the dimension of a DAE system and hence worsen the performance. Finally, side effects implied by the enforced order of sub-expressions evaluation result in slightly different results for state variables.

4 Automatic Differentiation of Modelica Code

ADModelica is a prototype of a source-to-source AD tool that strives to support Modelica programs. The source-to-source approach employs a combination of classical- and equation-based compiler techniques to transform a program source code into a new source code that computes the derivatives. This section gives a quick overview of the implementation of ADModelica.

4.1 Possible Approaches

There are three levels, on which AD of (implicit) DAE systems can operate:

1. **Library level**: All library units (i.e. components and connectors) are differentiated independently to generate another library that additionally computes parameter sensitivities of variables. Each component is augmented with code for derivatives.

2. **Flat Model Level**: The source code is given as (or transformed into) pure equations, represented by elementary Modelica's constructs, rather than physical formulation with components and connectors. Sensitivity Equations are added in a new Modelica model.

3. **Generated C-code level**: The generated C-code is differentiated.

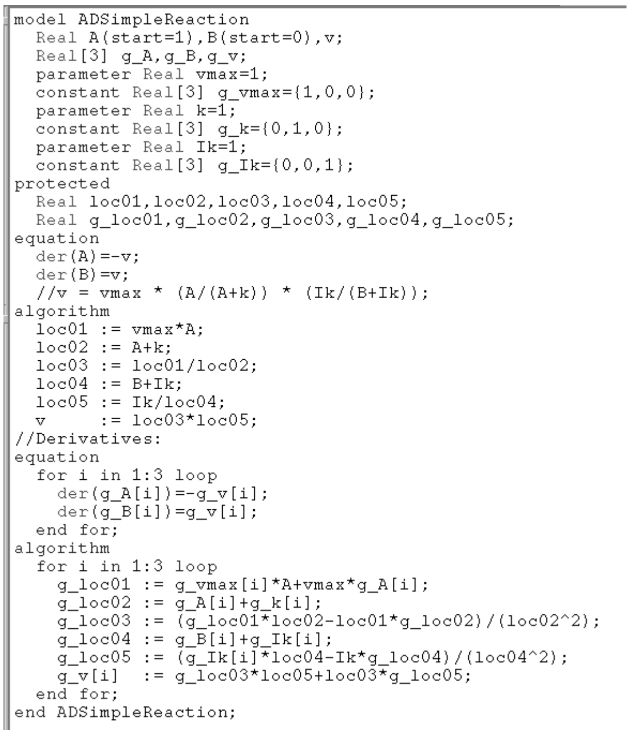
In [4], the above approaches are discussed in more details. The adopted approach is based on differentiation on the flat model level. The current supported input models, are namely those, which flattened models have pure mathematical formulation. Particularly, input models with components, connectors and arrays with equations expressed as *for*-loops are supported. However, some control constructs in Modelica, such as *if*, *while* and others, are not yet supported. As a remark, AD of such classical languages constructs is a well-know problem and has been successfully handled [6].

4.2 Overview of ADModelica

Figure 2 shows the corresponding Modelica implementation of the DAE system (9). The user specifies the independent variables. If not specified, all parameters are considered as independent variables. To every variable v of type Real an array representing the gradient of that variable *g_v* is associated. The array's size represents the number of independent variables. Each entry of the array represents the derivative of v with respect to an independent variable. To each active variable, a gradient is associated. ADModelica follows a conservative strategy that considers all variables and parameters active. In that case, non-interesting parameters have the zero gradients.

4.3 Design and Implementation

Implementing an AD tool from scratch, supporting a wide set of Modelica grammar, would be an ex-



```
model ADSimpleReaction
  Real A(start=1),B(start=0),v;
  Real[3] g_A,g_B,g_v;
  parameter Real vmax=1;
  constant Real[3] g_vmax={1,0,0};
  parameter Real k=1;
  constant Real[3] g_k={0,1,0};
  parameter Real Ik=1;
  constant Real[3] g_Ik={0,0,1};
protected
  Real loc01,loc02,loc03,loc04,loc05;
  Real g_loc01,g_loc02,g_loc03,g_loc04,g_loc05;
equation
  der(A)=-v;
  der(B)=v;
  //v = vmax * (A/(A+k)) * (Ik/(B+Ik));
algorithm
  loc01 := vmax*A;
  loc02 := A+k;
  loc03 := loc01/loc02;
  loc04 := B+Ik;
  loc05 := Ik/loc04;
  v     := loc03*loc05;
//Derivatives:
equation
  for i in 1:3 loop
    der(g_A[i])=-g_v[i];
    der(g_B[i])=g_v[i];
  end for;
algorithm
  for i in 1:3 loop
    g_loc01 := g_vmax[i]*A+vmax*g_A[i];
    g_loc02 := g_A[i]+g_k[i];
    g_loc03 := (g_loc01*loc02-loc01*g_loc02)/(loc02^2);
    g_loc04 := g_B[i]+g_Ik[i];
    g_loc05 := (g_Ik[i]*loc04-Ik*g_loc04)/(loc04^2);
    g_v[i]  := g_loc03*loc05+loc03*g_loc05;
  end for;
end ADSimpleReaction;
```

Figure 2: Implementation of the DAE system (4) and its Sensitivity Equations (5)

pensive and error-prone process. Therefore, existing tools and software are utilized by ADModelica, particularly OMC [5]. OMC allows communication with other tools through the CORBA interface. Figure 3 shows the main steps performed to generate a Modelica model that computes additional required derivatives. These steps are summarized as follows:

- **Flattening**: A high-level model is transformed to a model with pure mathematical equations, using the Open Modelica Compiler (OMC). ADModelica makes use of the CORBA interface, offered by OMC.

- **Transforming to intermediate format**: The ModelicaXML parser [15] parses an input model to an easy-to-handle format, in which the AST representation of the equations are implicitly inherited. The ASTs are extracted into intermediate format in Java classes.

- **Analyzing**: The dimension of the generated DAE system is reduced by removing alias equations (s.a. $x=y$ and $x+y=0$) [9]. The computational path between variables is computed [3, 8].

- **Differentiating**: The ASTs of the derivatives are computed. A conservative strategy is to differentiate all equations. However, it is enough to differentiate all equations laying in all Strongly Connected Components (SCCs) of the computational path from the independent variable(s) to the dependent variable(s).

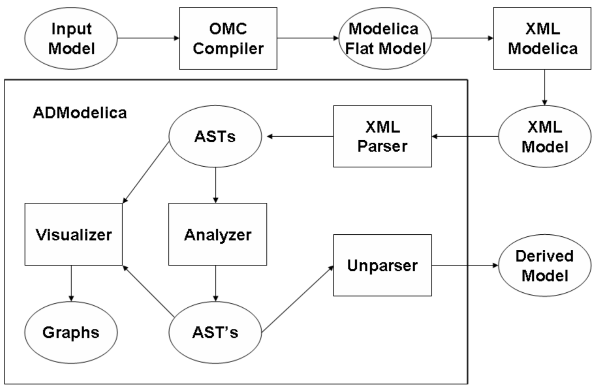


Figure 3: The Architecture of ADModelica

- **Unparsing**: The differentiated model is generated with additional code for derivatives.

- **Visualizing ASTs**: Producing graphs of the ASTs was proven to be useful during the course of development, for finding potential semantical mistakes.

5 Application

Modeling the dynamics of metabolic reaction networks has a wide spectrum of applications. Special attention has been paid to modeling biochemical systems with Modelica [11]. In general, the parameters expressing the characteristics of enzymatic reactions (eg. reaction rate, enzyme activation/inhibition constants, etc.) are one of the largest source of uncertainty in modeling metabolic networks, and are not necessarily known. Their values might be estimated by fitting them to measured data, resulted from stimulus-response experiments [16]. Estimating the correct values of parameters can reveal hidden information about the system. However, even in that case, the asserted model alone does not explain the underlying behavior.

Understanding the functions of enzymatic reactions within a metabolic network can be achieved by measuring changes to directed perturbations of certain parameters (eg. quantity of a certain enzyme). While

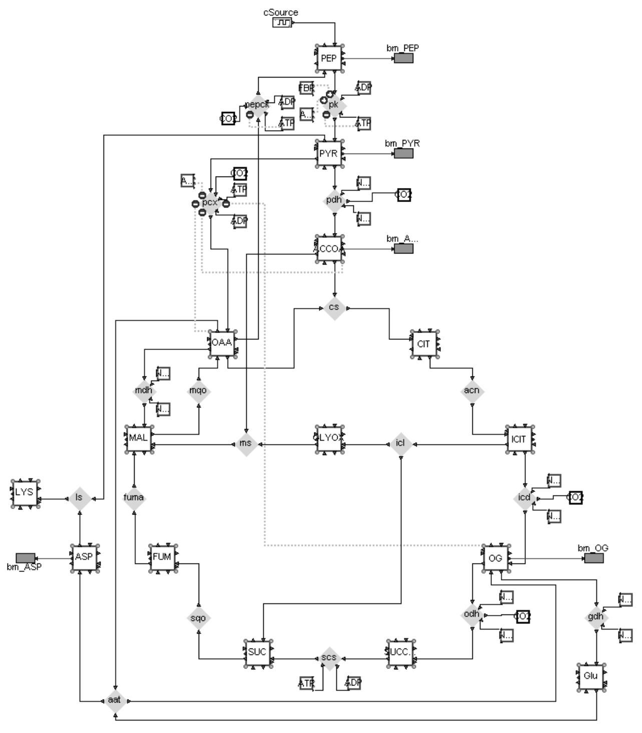


Figure 4: A dynamic Metabolic Network

this can be experimentally difficult or impractical, it is easier to quantify the effect of these changes using a validated model [17]. This can be achieved by computing the sensitivities of reaction rates and concentration to parameters $\partial r/\partial p$ and $\partial c/\partial p$, and the sensitivities of reaction rates to concentration of metabolites $\partial r/\partial c$. Using these sensitivities, the well known quantities of Metabolic Control Theory, i.e. the concentration and flux control coefficients C^M and C^F, can be calculated [10, 7].

In Figure 4, a dynamic metabolic network model including reactions of the tricarbon acid cycle is shown. The network has been implemented using a specialized library for biochemical networks, making use of many object-oriented features of the Modelica language. Various classes (e.g. Enzyme, Metabolites, Reactions) are the main common objects. Objects are connected via interfaces for potential variables (e.g. concentration c) and flux variables (e.g. reaction rate r). The dimension of the corresponding DAE system of the flattened model is 690. The number of non-trivial equations is 182. It takes few milli-seconds to simulate the network using Dymola (Dynasim AB, Sweden). The model was differentiated w.r.t. 64 independent variables, 49 of which are parameters corresponding to enzymatic characteristics and 15 concentrations variables. The dimension of the generated DAE is 12,270. It takes about 35 seconds to get the network and corresponding sensitivities simulated.

Investigations on the dynamics of metabolic network models mostly follow a system perturbation starting from a stationary state. In this example, the network is stimulated by a pulse of the input metabolite PEP. Results show that responses of following metabolite pools are very fast (e.g. PYR) or delayed (e.g. AC-COA). Especially in the case of the output metabolite LYS the concentration change is rather low in the given time frame. The results are used to identify some model parameters, which show a higher sensitivity in the instationary case directly after system perturbation, as well as others, which generally do not have any significant influence on the corresponding flux.

6 Summary and Future Work

This work shows that AD is a natural choice for computing sensitivities of solution variables for Modelica models. ADModelica is a prototype of a source-to-source AD tool for the Modelica language. It follows the flat model approach, as it is easy to implement because it does not consist of high-level language constructs. ADModelica utilizes OMC by using CORBA communication. Potential improvements of ADModelica can be achieved by making more use of OMC. OMC access a lot of facilities that can be utilized by ADModelica. Examples involve, but are not limited to:

- Symbolic manipulation of algebraic equations
- An intermediate format for computational graphs for DAE systems
- Utilizing the Dependency Flow Graph (DFG) of variables for decomposing a large resorted DAE system in Block Lower Triangular (BLT) format into smaller DAE systems

These facilities are used for optimizing common subexpressions, reducing the number of equations needed to be differentiated and computing sensitivities of variables w.r.t. other (non input) variables. Although, these are partially implemented by ADModelica, it is certainly better to rely on the reliable well-maintained and continuously growing OMC.

acknowledgment. This research is funded by German Ministry of Education and Research (BMBF) within the Sysmap project.

References

[1] C. H. Bischof, P. Khademi, A. Mauer, and A. Carle. *Adifor 2.0: Automatic Differentiation of Fortran 77 Programs*. *IEEE Computational Science & Engineering*, 3(3):18–32, Fall 1996.

[2] C. H. Bischof, L. Roh, and A. Mauer. *ADIC — An Extensible Automatic Differentiation Tool for ANSI-C*. *Software: Practice and Experience*, 27(12):1427–1456, 1997.

[3] F. E. Cellier. *Continuous System Modeling*. Springer Verlag, 1991.

[4] A. Elsheikh and W. Wiechert. Automatic Sensitivity Analysis of DAE-Systems Generated from Equation-Based Modeling Languages. *submitted to 5th International Conference on Automatic Differentiation*, Bonn, Germany, 11-15 August 2008.

[5] P. Fritzson, P. Aronsson, P. Bunus, V. Engelson, L. Saldami, H. Johansson, and A. Karström. The Open Source Modelica Project. In *Proceedings of The 2nd International Modelica Conference*, pages 297–306, Munich, Germany, March 2002.

[6] A. Griewank. *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*. Number 19 in Frontiers in Appl. Math. SIAM, Philadelphia, PA, 2000.

[7] R. Heinrich and S. Schuster, editors. *The Regulation of Cellular Systems*. Springer, 1996.

[8] A. Leitold and K. M. Hangos. Structural Solvability Analysis of Dynamic Process Models. *Computers & Chemical Engineering*, 25:1633–1646, 2001.

[9] C. Maffezzoni, R. Girelli, and P. Lluka. Generating Efficient Computational Procedures from Declarative Models. *Simul. Pr. Theory*, 4(5):303–317, 1996.

[10] K. Mauch, S. Arnold, and M. Reuss. Dynamic Sensitivity Analysis for Metabolic Systems. *Chem. Eng. Sci.*, 52:2589–2598, 1997.

[11] E. L. Nilsson and P. Fritzson. A Metabolic Specialization of a General Purpose Modelica Library for Biological and Biochemical Systems. In *Modelica2005, the 4th International Modelica Conference*, pages 85–93, Hamburg, Germany, March 2005.

[12] H. Olsson, H. Tummescheit, and H. Elmqvist. Using Automatic Differentiation for Partial Derivatives of Functions in Modelica. In *Modelica2005, the 4th International Modelica Conference*, pages 105–112, Hamburg, Germany, March 2005.

[13] C. C. Pantelides. The Consistent Initialization of Differential-Algebraic Systems. *SIAM Journal on Scientific and Statistical Computing*, 9(2):213–231, Mar. 1988.

[14] L. Petzold, S. T. Li, Y. Cao, and R. Serban. Sensitivity Analysis of Differential-Algebraic Equations and Partial Differential Equations. *Computers & Chemical Engineering*, 30:1553–1559, 2006.

[15] A. Pop and P. Fritzson. Modelicaxml: A Modelica XML Representation with Applications. In *Proceedings of The 3rd International Modelica Conference*, pages 419–429, Linköping, Sweden, November 2003.

[16] S. Wahl, M. Haunschild, M. Oldiges, and W. Wiechert. Unravelling the Regulatory Structure of Biochemical Networks Using Stimulus Response Experiments and Large-scale Model Selection. In *IEEE Proceedings Systems Biology*, volume 153, pages 275–285. IEEE Computer Society, 2006.

[17] W. Wiechert. Validation of Metabolic Models: Concepts, Tools, and Problems. In B. Kholodenko and H. Westerhoff, editors, *Metabolic Engineering in the Post Genomic Era*, chapter 11. Horizon Bioscience, 2004.