Systematic Modeling and Symbolically Assisted Simulation of Power Systems

Ian A. Hiskens, *Senior Member, IEEE* and Peter J. Sokolowski

***Abstract*—Large disturbance behavior of power systems often involves complex interactions between continuous dynamics and discrete events. Such behavior can be captured in a systematic way by a model that consists of differential, switched algebraic and state-reset (DSAR) equations. The paper presents a practical object-oriented approach to implementing the DSAR model. Each component of a system can be modeled autonomously. Connections between components are established by simple algebraic equations. Simulation of the model using numerically robust implicit integration requires the generation of partial derivatives. The object-oriented model structure allows this differentiation to be achieved symbolically without sacrificing simulation speed.**

***Index Terms*—Dynamic modeling, hybrid systems, simulation.**

I. Introduction

POWER systems frequently exhibit interactions between continuous dynamics and discrete events. They are an important class of *hybrid systems*. Typically the continuous dynamics relate to components that obey physical laws. Event-driven discrete behavior results from logical rules that govern the system. Examples in this latter category include protection logic, supervisory control and saturation limits.

Analysis of power system dynamic behavior can be difficult. Even simulation is not always straightforward. Yet the importance of clearly understanding such behavior makes systematic modeling, analysis and simulation imperative. Historically, analysis of physical systems focused on their continuous behavior, with the discrete-event activity remaining largely external. *Ad hoc* approaches to modeling and simulation of the discrete interactions were sufficient. However, such methods are less appropriate with the recent trend toward tighter integration of continuous and discrete activity. Fresh systematic approaches to the simulation of hybrid systems are under development [1]. This paper establishes benefits of symbolic differentiation in an object-oriented simulation environment.

Hybrid systems can be modeled by a set of differential, switched algebraic and state-reset (DSAR) equations [2]. A summary of this model structure is presented in Section II. The utility of such a model for building large systems is not immediately clear. However, Section III shows that the DSAR structure is amenable to object-oriented modeling. Such modeling is perfectly suited to symbolic manipulation. The implementation of symbolic differentiation is discussed in Section IV. Conclusions are presented in Section V. Examples are used to illustrate model development and implementation.

Manuscript received September 8, 2000. This work was supported by the Australian Research Council through the grant "Analysis and Assessment of Voltage Collapse," and the EPRI/DoD Complex Interactive Networks/Systems Initiative.

I. A. Hiskens is with the Department of Electrical and Computer Engineering, University of Illinois at Urbana-Champaign, USA.

P. J. Sokolowski is with the Department of Electrical and Computer Engineering, The University of Newcastle, Australia.

Publisher Item Identifier S 0885-8950(01)03787-7.

II. Hybrid System Representation

A. Model

As indicated in Section I, hybrid systems, which include power systems, are characterized by:

- continuous and discrete states,
- continuous dynamics,
- discrete events, or triggers, and
- mappings that define the evolution of discrete states at events.

It is shown in [2] that such behavior can be captured by the DSAR model

$$\dot{\underline{x}} = \underline{f}(\underline{x}, y) \tag{1}$$

$$0 = g^{(0)}(\underline{x}, y) \tag{2}$$

$$0 = \begin{cases} g^{(i-)}(\underline{x}, y) & y_{d,i} < 0 \\ g^{(i+)}(\underline{x}, y) & y_{d,i} > 0 \end{cases} \qquad i = 1, \ldots, d \tag{3}$$

$$\underline{x}^{+} = \underline{h}_j\left(\underline{x}^{-}, y^{-}\right) \qquad y_{e,j} = 0 \qquad j \in \{1, \ldots, e\} \tag{4}$$

where

$$\underline{x} = \begin{bmatrix} x \\ z \\ \lambda \end{bmatrix}, \qquad \underline{f} = \begin{bmatrix} f \\ 0 \\ 0 \end{bmatrix}, \qquad \underline{h}_j = \begin{bmatrix} x \\ h_j \\ \lambda \end{bmatrix}$$

and

- x are the continuous dynamic states, for example generator angles, velocities and fluxes,
- z are discrete dynamic states, such as transformer tap positions and protection relay logic states,
- y are algebraic states, e.g., load bus voltage magnitudes and angles,
- λ are parameters such as generator reactances, controller gains and switching times.

The differential equations $\underline{f}$ are correspondingly structured so that $\dot{x} = f(\underline{x}, y)$, whilst z and λ remain constant away from events. Similarly, the reset equations $\underline{h}_j$ ensure that x and λ remain constant at reset events, but the dynamic states z are reset to new values according to $z^{+} = h_j(\underline{x}^{-}, y^{-})$. (The notation $\underline{x}^{+}$ denotes the value of $\underline{x}$ just after the reset event, whilst $\underline{x}^{-}$ and y^{-} refer to the values of $\underline{x}$ and y just prior to the event.)

Note that the model does not allow discontinuities in the dynamic states x, i.e., impulse effects are excluded. This is not a restriction forced by analysis though. The model adopts the philosophy that the dynamic states of *actual* systems cannot undergo step changes.

The model (1)–(4), which is similar to a model proposed in [3], captures all the important aspects of hybrid system behavior, namely the interaction between continuous and discrete states as they evolve over time. Between events, system behavior is governed by the differential-algebraic (DA) dynamical system

$$\dot{\underline{x}} = \underline{f}(\underline{x}, y) \tag{5}$$

$$0 = g(\underline{x}, y) = \begin{bmatrix} g^{(0)}(\underline{x}, y) \\ g^{(1)}(\underline{x}, y) \\ \vdots \\ g^{(d)}(\underline{x}, y) \end{bmatrix} \tag{6}$$

where the functions $g^{(i)}$ are chosen depending on the signs of the corresponding elements of y_d. An event is triggered by an element of y_d changing sign and/or an element of y_e passing through zero. At an event, the composition of g changes and/or elements of z are reset.

The following example illustrates the DSAR model structure.

Example 1: In order to demonstrate the ability of the DSAR structure (1)–(4) to model logic-based systems, this example considers a relatively detailed representation of the automatic voltage regulator (AVR) of a tap-changing transformer. The Petri net model [1], [4] of the AVR logic for low voltages, i.e., for increasing tap ratio, is outlined in Fig. 1. The model can be represented in the DSAR form as,

$$\begin{aligned}
\dot{x}_1 &= y_1 y_7 \\
0 &= y_2 - V_2 + V_{\text{low}} \\
0 &= y_3 - y_4 + z_1 \\
0 &= y_6 - n + n_{\max} - n_{\text{step}}/2 \\
0 &= nV_1 - V_2 \\
0 &= y_1 - 1 && y_2 < 0 \\
&\left.\begin{aligned} 0 &= y_1 \\ 0 &= y_4 - x_1 \end{aligned}\right\} && y_2 > 0 \\
0 &= y_7 - 1 && y_6 < 0 \\
0 &= y_7 && y_6 > 0 \\
0 &= y_5 - x_1 + z_1 + T_{\text{tap}} && y_3 < 0 \\
0 &= y_5 - x_1 + y_4 + T_{\text{tap}} && y_3 > 0 \\
&\left.\begin{aligned} z_1^+ &= x_1^- \\ n^+ &= N^- + n_{\text{step}} \end{aligned}\right\} && \text{when } y_5 = 0.
\end{aligned}$$

To assist in connecting AVR logic with the model, Fig. 1 indicates variables that are related to particular functions.

The dynamics of this device are driven by a number of interacting events that govern the behavior of the timer. If the tap is at the upper limit ($y_6 > 0$), or the voltage is within the deadband ($y_2 > 0$) then the timer is blocked. If the voltage is outside the deadband ($y_2 < 0$) then the timer will run. If the timer

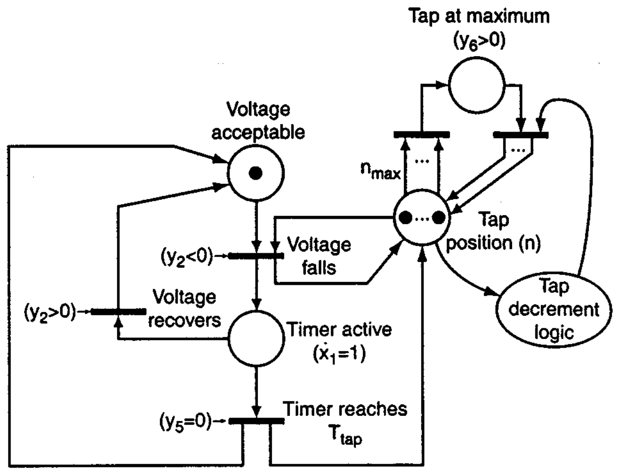

Fig. 1. Tap-changing transformer AVR logic for increasing tap.

reaches T_{tap} ($y_5 = 0$), a tap change will occur and the timer will be reset, but not necessarily blocked. If the voltage returns to within the deadband, because of smooth system dynamics or a tap change or some other system event, then the timer is blocked and reset. □

B. Numerical Integration

Numerical integration of DA systems is treated rigorously in [5]. However it is helpful to review some basic concepts before considering the implementation of the DSAR model and the role of symbolic differentiation.

Consider the DA system (5), (6) which describes behavior over the periods between events. The trapezoidal approach to numerical integration approximates the differential equations (5) by a set of algebraic difference equations coupled to the original algebraic equations (6), i.e.,

$$\underline{x}^{k+1} = \underline{x}^k + \frac{\eta}{2}\left(\underline{f}\left(\underline{x}^k, y^k\right) + \underline{f}\left(\underline{x}^{k+1}, y^{k+1}\right)\right) \tag{7}$$

$$0 = g\left(\underline{x}^{k+1}, y^{k+1}\right) \tag{8}$$

where the superscripts k, $k+1$ index the time instants t_k, t_{k+1} respectively, and $\eta = t_{k+1} - t_k$ is the integration time step. Equations (7), (8) describe the evolution of the states $\underline{x}$, y from time instant t_k to the next time instant t_{k+1}.

Notice that (7), (8) form a set of implicit nonlinear algebraic equations. Therefore to solve for $\underline{x}^{k+1}$, y^{k+1} given $\underline{x}^k$, y^k requires the use of a nonlinear equation solver. The Newton iterative technique is commonly used. Rearranging (7) allows the algebraic equations to be written

$$\begin{aligned}
F&\left(\underline{x}^{k+1}, y^{k+1}\right) \\
&= \begin{bmatrix} \frac{\eta}{2}\underline{f}\left(\underline{x}^{k+1}, y^{k+1}\right) - \underline{x}^{k+1} + \frac{\eta}{2}\underline{f}\left(\underline{x}^k, y^k\right) + \underline{x}^k \\ g\left(\underline{x}^{k+1}, y^{k+1}\right) \end{bmatrix} \\
&= 0
\end{aligned}$$

which has the form

$$F(\varkappa) = 0.$$

This equation can be solved iteratively according to

$$\varkappa_{i+1} = \varkappa_i - F_\varkappa(\varkappa_i)^{-1} F(\varkappa_i) \tag{9}$$

where $F_\varkappa$ is the Jacobian of F with respect to $\varkappa$, and has the structure

$$F_\varkappa = \begin{bmatrix} \frac{\eta}{2}\underline{f}_{\underline{x}} - I & \frac{\eta}{2}\underline{f}_y \\ g_{\underline{x}} & g_y \end{bmatrix}. \tag{10}$$

Note that i indexes the iterations of the equation solver, and is not related to the time index k. When (9) has converged, the solution $\varkappa$ provides $\underline{x}^{k+1}$ and y^{k+1}.

C. Computation of Junction Points

Switching and reset events generically do not coincide with the time instants of the numerical integration process. However for many applications it is important to find the exact time, between integration time steps, at which an event occurs. This is possible through a simple modification to the trapezoidal technique.

Referring to the compact DSAR model (1)–(4), let $y_i = 0$ trigger an event. Say $y_i < 0$ at time instant k, but $y_i > 0$ at instant $k+1$. Let η^* be the (unknown) time from instant k to the event. The variable η^* can be found by solving (7), (8) with η free to vary, but with the extra constraint $y_i = 0$. Because the extra variable is matched by an extra constraint, the Newton iterative technique can again be used to find the solution.

Having found the junction point, the appropriate switches in the composition of g should be made, and/or z updated, then (8) re-solved to obtain the post-event values of the algebraic variables y. The post-event values of $\underline{x}$ and y provide the initial conditions for the next section of the trajectory. It can be convenient to use the time step $\eta - \eta^*$ for the first step after the event. This aligns subsequent points with the specified time step η.

III. Implementation

Models of large systems are most effectively constructed using a hierarchical or modular approach. With such an approach, components are grouped together as subsystems, and the subsystems are combined to form the full system. This allows component and subsystem models to be developed and tested independently. It also allows flexibility in interchanging models.

The interactions inherent in hybrid systems are counter to this decomposition into subsystems and components. However the algebraic equations of the DSAR model can be exploited to achieve the desired modularity. Each component or subsystem can be modeled autonomously in the DSAR structure, with "interface" quantities, e.g., inputs and outputs, established as algebraic variables. The components are then interconnected by introducing simple algebraic equations that "link" the interface variables. This is similar to the connections concept of [6]. Note that all interconnections are noncausal [7], i.e., no rigid input–output arrangement of components is assumed.

To illustrate this linking concept, consider a case where the nth algebraic state of component j, denoted $y_{j,n}$, is required by component k. In the model of component k, the corresponding quantity would appear as an algebraic variable $y_{k,m}$. The connection is made via the simple algebraic equation $y_{j,n} - y_{k,m} = 0$. In general, all linking can be achieved by summations of the form

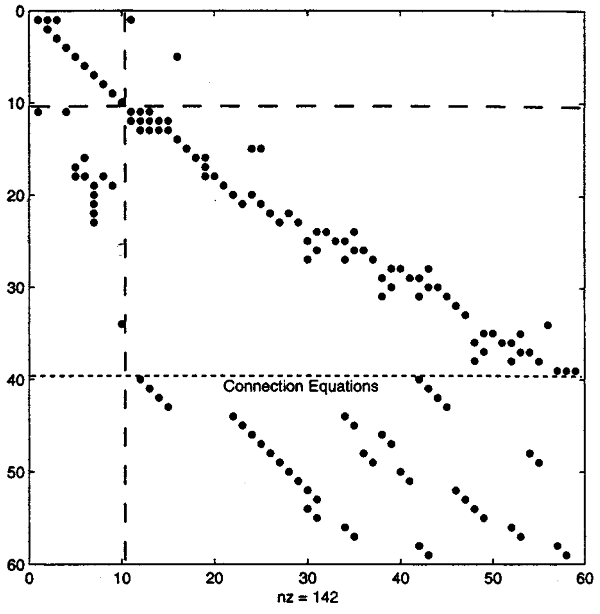

Fig. 2. Sparsity structure of Jacobian $F_\varkappa$ for the power system example.

$$\sum c_k y_{i,j} = 0 \tag{11}$$

where c_k is ± 1. Notice that all connections are external to the component models.

The linking strategy results in an interesting structure for the Jacobian $F_\varkappa$. Components contribute square blocks down the diagonal of $\underline{f}_{\underline{x}}$ and flattened rectangular blocks along the diagonal of the upper section of g_y. The lower section of g_y is an incidence matrix, with ± 1's given by the external connections (11). Fig. 2 illustrates this structure. (This particular matrix comes from Example 2, which follows.) A Jacobian structure like that of $F_\varkappa$ was identified in [8], where a similar arrangement of components and connections was used in the development of an optimal power flow.

The structure and values of the lower connection submatrix of $F_\varkappa$ are fixed for all time. This can be exploited in the factorization of $F_\varkappa$ to improve the efficiency of solving (9). The efficiency improvement is significant as (9) is solved at every time step.

The proposed modular approach to constructing hybrid systems has been implemented in Matlab. The following example illustrates the concepts.

Example 2: The simple power system of Fig. 3 consists of a dynamic load supplied from an infinite bus via a tap-changing transformer. The continuous dynamics of the real power load are given by the recovery model [9],

$$\begin{aligned} \dot{x}_p &= \frac{1}{T_p}\left(P_s^0 - P_d\right) \\ P_d &= x_p + P_t V_3^2 \end{aligned}$$

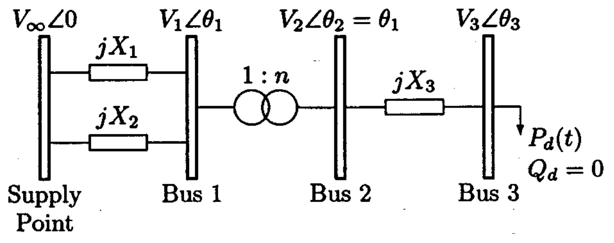

Fig. 3. Power system example.

```
model_data = {'load_dyn1' [0 5 0.4 2.0] [0.4 1 0 -0.4 0] [0.4] ;
              'tap_changer' [0 -1 1.0375 20 0.0125] ...
                 [0 0 0 0 0 0 1 0 1 0 -0.4 0 0.4 0] ...
                 [1.04 1.1] ;
              'network' [] [1 0     0 0 ... % Infinite bus
                            1 0 -0.4 0 ... % Bus 1
                            1 0  0.4 0 ... % Bus 2
                            1 0 -0.4 0]... % Bus 3
                          [1 2 0 0.65 0;    % Line data: f t R X B
                           3 4 0 0.80 0] ;
              'infinite_bus' [] [1.05 0] [1.05] ;
              'switched_line' [0] [1 0 0 0 ... % From bus
                                   1 0 0 0 -1] ... % To bus
                              [0 0.40625 0 10] ;
              'out_Vmag' [] [1 0 1] [] };
connections = {[1 2 3 -13] ...
               [1 3 3 -14] ...
               [1 4 3 -15] ...
               [1 5 3 -16] ...
               [2 7 3 -5] ...
               [2 8 3 -6] ...
               [2 9 3 -9] ...
               [2 10 3 -10] ...
               [2 11 3 -7 5 7] ...
               [2 12 3 -8 5 8] ...
               [2 13 3 -11] ...
               [2 14 3 -12] ...
               [4 1 3 -1] ...
               [4 2 3 -2] ...
               [5 1 3 -1] ...
               [5 2 3 -2] ...
               [5 5 3 -5] ...
               [5 6 3 -6] ...
               [6 1 3 -13] ...
               [6 2 3 -14] };
```

Connection vector discussed in the text.

Fig. 4. Data file for transformer-load system.

where x_p is the load state driving the actual load demand P_d. Tap-changer behavior is described in Example 1.

This system is described by the data file of Fig. 4. Each component of the system has a corresponding entry in `model_data`; the dynamic load and tap-changer are represented by `load_dyn1` and `tap_changer` respectively, whilst the models for the infinite bus, network and switched line are appropriately named. Each model is associated with three data vectors. These specify initialization values for $\underline{x}_0$, y_0 and background parameters respectively.

The actual Matlab file `load_dyn1` (shortened and compacted to save space) is provided in Fig. 5. This file calculates values for $\underline{f}$, g and $\underline{h}$, and sparsely stored elements of $\underline{f}_{\underline{x}}$, $\underline{f}_y$, $g_{\underline{x}}$, $\ldots$, $\underline{h}_y$. The parameter `flag` determines which quantity is returned at each function call.

Relative indexing is used within models for Jacobian elements, as each component model is autonomous. (All connection information is externally defined.) The model `load_dyn1` of Fig. 5 provides an illustration. Consider the line "`ans(12, :)` $=$ `[3 5 −y(2)]`," which is executed during the calculation of g_y. This specifies that the 12th nonzero element of the local g_y matrix is $g_{y(3,\,5)} = -y_2$. The simulation kernel uses these relative indices (3, 5), along with knowledge of the dimensions of g and y for all models, to generate the location of this element in the full g_y matrix, i.e., the absolute indices.

```
function ans = load_dyn1(x,y,t,ev,p,flag,model_no)
global empty3
%
if flag == 0                              % initialization
   ans = [4 3 []];
elseif flag == 1                          % calculate f
   ans = [(x(3)-y(1))/x(2) 0 0 0]';
elseif flag == 2                          % calculate f_x
   ans(1,:) = [1 2 -(x(3)-y(1))/(x(2)*x(2))];
   ans(2,:) = [1 3 1/x(2)];
elseif flag == 3                          % calculte f_y
   ans(1,:) = [1 1 -1/x(2)];
elseif flag == 4                          % calculate g
   ans = [x(1)+p(1)*(y(2)*y(2)+y(3)*y(3))^(x(4)/2)-y(1);...
          y(1)+(y(2)*y(4)+y(3)*y(5));
          y(3)*y(4)-y(2)*y(5)];
elseif flag == 5                          % calculate g_x
   ans(1,:) = [1 1 1];
   V = y(2)*y(2)+y(3)*y(3);
   ans(2,:) = [1 4 p(1)*log(V)/2*V^(x(4)/2)];
elseif flag == 6                          % calculate g_y
   ans(1,:) = [1 1 -1];
   fac = p(1)*x(4)*(y(2)*y(2)+y(3)*y(3))^(x(4)/2-1);
   ans(2,:) = [1 2 fac*y(2)];
   <---cut--->
   ans(11,:) = [3 4 y(3)];
   ans(12,:) = [3 5 -y(2)];
elseif flag == 7                          % calculate h
   ans = x;
elseif flag == 8                          % calculate h_x
   ans = [[1:4]' [1:4]' ones(4,1)];
elseif flag == 9                          % calculate h_y
   ans = empty3;
end
```

Line discussed in the text.

Fig. 5. Component file `load_dyn1`.

Note that the actual g_y matrix is never built explicitly, but rather is stored sparsely.

The version of the model shown in Fig. 5 describes a single (load) component. However a straightforward modification of the code, using Matlab vectorization, allows an arbitrary number of components to be described simultaneously. This significantly improves execution speed for systems with multiple instances of the same component.

As indicated earlier, the models are interconnected through interface variables. Consider the connection of components such as the load to the network. The model `network` provides a nodal representation of the network constraints, and introduces four algebraic variables at each bus, *viz.*, real and imaginary components of bus voltage V_r, V_i and injected current I_r, I_i. The model `load_dyn1` describes load behavior in terms of terminal bus algebraic variables V_r, V_i, I_r and I_i. The link between network and load variables is established via `connections`. Each vector in `connections` contains pairs of indices which set up an equation of the form (11). Referring to the data file of Fig. 4, the first vector "`[1 2 3 −13]`," for example, introduces the equation

$$0 = y_{1,\,2} - y_{3,\,13}$$

where $y_{i,\,j}$ refers to the jth algebraic variable of the ith model. This particular equation ensures that the real part of the voltage seen by the load $(y_{1,\,2})$ is equal to the appropriate network voltage $(y_{3,\,13})$.

The structure of $F_{\varkappa}$ for this example is shown in Fig. 2. It is clearly sparse. As mentioned earlier, $F_{\varkappa}$ is composed of blocks down the diagonal, together with the cross couplings $\underline{f}_y$ and $g_{\underline{x}}$, and the connection submatrix. The matrix $\underline{f}_{\underline{x}}$, which occupies the top left corner of $F_{\varkappa}$, has dimension $\mathbb{R}^{10\times 10}$. The last 20 rows correspond to the connection equations. They interconnect the diagonal blocks of g_y.

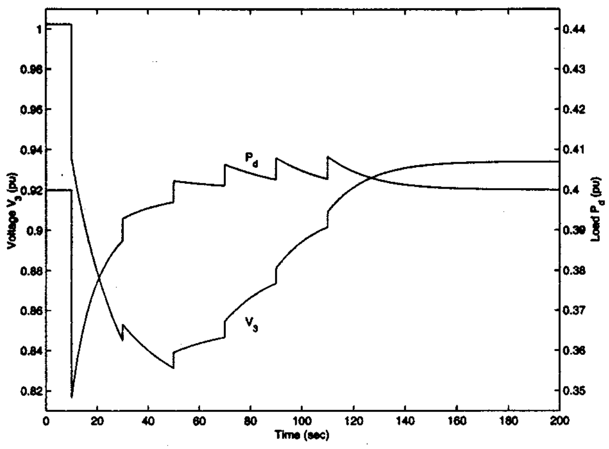

Fig. 6. System response to disturbance.

A disturbance was applied to the power system to illustrate the interactions between continuous dynamics (due to the load) and discrete event dynamics (resulting from the tap-changing transformer). At $t = 10$ sec, the feeder with impedance jX_2 was tripped. The behavior of the voltage at bus 3 is shown in Fig. 6, along with the load demand P_d. The nonsmooth nature of the trajectory is clearly evident. $\square$

In general, components and subsystems of any form can be modeled, provided they are structured with interfacing algebraic variables that can be linked to other components. Noise and/or random disturbances can be added to the model by linking components that generate random signals.

IV. Symbolic Differentiation

As indicated in Section II-B, simulation of hybrid systems using numerically stable implicit integration techniques requires the generation of the partial derivative matrices $\underline{f}_{\underline{x}}$, $\underline{f}_{\underline{y}}$, $g_{\underline{x}}$, ..., $\underline{h}_{\underline{y}}$. In the implementation described in this paper, these values are calculated and stored sparsely by component files of the form shown in Fig. 5. Hand derivation of these partial derivatives can be tedious for large complicated models. Therefore the process has been automated through the use of symbolic differentiation.

The generation of a component file, like Fig. 5 for example, begins with an analytical model in the DSAR form. The analytical model must be unambiguously mapped into a character representation that can be manipulated symbolically. It is also important that this mapping does not restrict the implementation of the DSAR form. Fortunately the DSAR model structure is well suited to such translation. For example, the model

$$\begin{aligned}
\dot{x}_1 &= x_1 y_1^2 \\
\dot{x}_2 &= 0 \\
0 &= y_3 - p_1/x_1 \\
0 &= \begin{cases} y_1 + x_2 & y_5 < 0 \\ \tan(\log(x_1)/y_4) & y_5 > 0 \end{cases} \\
&\left.\begin{aligned} x_1^+ &= x_1^- \\ x_2^+ &= y_2^- + x_2^- \end{aligned}\right\} \quad \text{when } y_1 = 0
\end{aligned}$$

```
f equations
   f1 = x1*y1^2
   f2 = 0
g equations
   g1 = y3-p1/x1
ev-  y5
   g2 = y1+x2
ev+
   g2 = tan(log(x1)/y4)
h equations
ev   y1
   h1 = x1
   h2 = y2+x2
```

Fig. 7. Input model representation for component file building.

```
function ans = pica(x,y,t,ev,p,flag,model_no
global empty3
%
if flag == 0                    % initialisation
    ans = [2 2 [5 -1]];
elseif flag == 1                % calculate f
    ans = [x(1)*y(1)^2 ; 0];
elseif flag == 2                % calculate f_x
    ans(1,:) = [1 1 y(1)^2];
elseif flag == 3                % calculate f_y
    ans(1,:) = [1 1 2*x(1)*y(1)];
elseif flag == 4                % calculate g
    ans(1) = y(3)-p(1)/x(1);
    if ev(1) < 0
        ans(2) = y(1)+x(2);
    else
        ans(2) = tan(log(x(1))/y(4));
    end
elseif flag == 5                % calculate g_x
    ans(1,:) = [1 1 p(1)/x(1)^2];
    if ev(1) < 0
        ans(2,:) = [2 2 1];
    else
        ans(2,:) = [2 1 (1+tan(log(x(1))/y(4))^2)/x(1)/y(4)];
    end
elseif flag == 6                % calculate g_y
    ans(1,:) = [1 3 1];
    if ev(1) < 0
        ans(2,:) = [2 1 1];
    else
        ans(2,:) = [2 4 -(1+tan(log(x(1))/y(4))^2)*log(x(1))/y(4)^2];
    end
elseif flag == 7                % calculate h
    ans = x;
    if ev == 2
        ans(1) = x(1);
        ans(2) = y(2)+x(2);
    end
elseif flag == 8                % calculate h_x
    ans = [[1:2]; [1:2]; ones(1,2)]';
    if ev == 2
        ans(1,:) = [1 1 1];
        ans(2,:) = [2 2 1];
    end
elseif flag == 9                % calculate h_y
    ans = empty3;
    if ev == 2
        ans(1,:) = [2 2 1];
    end
end
```

Fig. 8. Symbolically generated component file.

is fully described by the representation of Fig. 7. All elements of the model are clearly and uniquely identified: $\underline{f}$, $g^{(0)}$, $\{g^{(i-)}, g^{(i+)}, y_{d,i}, i = 1, \ldots, d\}$, $\{\underline{h}_j, y_{e,j}, j = 1, \ldots, e\}$.

A Matlab function has been developed for translating the input model representation, of the form shown in Fig. 7, into a component file that can interact with the simulation kernel. For example the component file produced from Fig. 7 is shown in Fig. 8 (compacted to save space). Building the $\underline{f}$, g and $\underline{h}$ equations involves relatively straightforward character string manipulation. Generating the partial derivatives is more challenging. Firstly, equations and variable strings are converted into symbols. Symbolic differentiation produces partial derivatives that must be simplified and converted back into strings. If the final expression is zero, the derivative is discarded, as the matrices are stored sparsely. Final processing adds parentheses for variable indexing, e.g., `y1` becomes `y(1)`.

Component files are generated "off-line." Therefore symbolic manipulation has no influence on simulation time.

V. Conclusion

Many systems exhibit interactions between continuous dynamics and discrete events. The paper illustrates that the dynamics of such hybrid systems can be captured by a model which has a differential-algebraic-discrete (DSAR) structure.

Models of large systems are most effectively constructed using a modular or object-oriented approach. However the integrations inherent in hybrid systems make that difficult to achieve. The paper shows that the desired modularity can be achieved in a practical way with the DSAR model. Components and/or subsystems are modeled autonomously, with connections established via simple algebraic equations. The object-oriented model structure allows partial derivatives to be generated symbolically without sacrificing simulation speed.

References

[1] M. Otter, P. J. Mosterman, and H. Elmqvist, "Modeling Petri nets as local constraint equations for hybrid systems using Modelica," in *Proceedings of the Summer Computer Simulation Conference*, Reno, NV, July 1998.

[2] I. A. Hiskens and M. A. Pai, "Trajectory sensitivity analysis of hybrid systems," *IEEE Trans. on Circuits and Systems I*, vol. 47, no. 2, pp. 204–220, Feb. 2000.

[3] J. H. Taylor, "Rigorous handling of state events in Matlab," in *Proceedings 4th IEEE Conference on Control Applications*, Albany, NY, Sept. 1995, pp. 156–161.

[4] T. Murata, "Petri nets: Properties, analysis and applications," *Proceedings of the IEEE*, vol. 77, no. 4, pp. 541–580, Apr. 1989.

[5] K. E. Brenan, S. L. Campbell, and L. Petzold, "Numerical solution of initial-value problems," *Differential-Algebraic Equations*, 1995.

[6] H. Elmqvist, "A structured model language for large continuous systems," Ph.D. dissertation, Department of Automatic Control, Lund Institute of Technology, Lund, Sweden, 1978.

[7] H. Elmqvist, S. E. Mattsson, and M. Otter, "Modelica—A language for physical system modeling, visualization and interaction," in *Proceedings of the IEEE Symposium on Computer-Aided Control System Design*, HI, Aug. 1999.

[8] R. Bacher, "Symbolically assisted numeric computations for power system software development," in *Proceedings of the 13th Power Systems Computation Conference*, Trondheim, Norway, June 1999, pp. 5–16.

[9] D. J. Hill, "Nonlinear dynamic load models with recovery for voltage stability studies," *IEEE Transactions on PowerSystems*, vol. 8, no. 1, pp. 166–176, Feb. 1993.