Institute for Software Integrated Systems
Vanderbilt University
Nashville, Tennessee, 37235

Efficient Simulation of Component-Based Hybrid Models Represented as Hybrid Bond Graphs

Matthew Daigle, Indranil Roychoudhury,
Gautam Biswas, and Xenofon Koutsoukos

TECHNICAL REPORT

ISIS-06-712

Efficient Simulation of Component-Based Hybrid Models Represented as Hybrid Bond Graphs

Matthew Daigle, Indranil Roychoudhury, Gautam Biswas, and Xenofon Koutsoukos

Institute for Software Integrated Systems (ISIS)
Vanderbilt University, Nashville, TN 37235
`matthew.j.daigle, indranil.roychoudhury, gautam.biswas, xenofon.koutsoukos@vanderbilt.edu`

Abstract. The complexity of modern embedded systems often requires the use of simulations for systematic design, analysis, and verification tasks. The nonlinear and hybrid nature of these systems make the building of accurate and computationally efficient simulation models very challenging. In this work, we adopt the Hybrid Bond Graph (HBG) paradigm, a uniform, multi-domain physics-based modeling language with local switching functions that enable the reconfiguration of energy flow paths to model hybrid systems. The inherent causal structure in HBG models is exploited to derive efficient hybrid simulation models as reconfigurable block diagram structures. We demonstrate our approach by modeling and analyzing the behavior of an electrical power system.

1 Introduction

Modern engineering systems are complex and made up of a large number of interacting components with nonlinear hybrid behaviors. This makes the building of accurate and computationally efficient simulation models a very challenging task. Recently, researchers and practitioners have adopted component- [1] and actor-oriented [2] frameworks for systematic construction of large models of complex systems. Such frameworks consist of mathematical models for specifying individual component behavior and formal models of computation for defining component interactions. Simulation models are derived by representing the component behavior models as computational blocks, and the interaction models define the syntax and semantics of the connections between the blocks.

We have adopted the Hybrid Bond Graph (HBG) paradigm [3], an extension of the Bond Graph (BG) modeling language [4], for component-based modeling of embedded systems. It is a domain-independent topological modeling language that captures interactions among the different processes that make up the system and can be very effective in parameterized component-based modeling of hybrid systems. The challenge we face is in translating these models to computationally efficient simulation models.

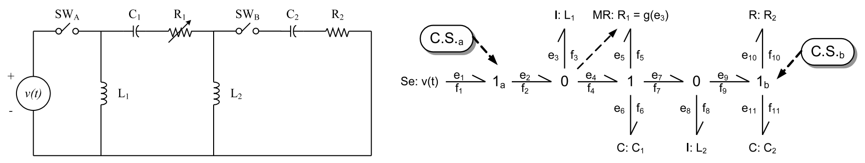
The causal structure inherent in BG models provides the basis for conversion of BGs to efficient computational models. For HBGs, mode changes imply dynamic changes in the causal structure, and this alters the computational model during execution. This paper develops a method for efficient simulation of HBG models by converting them to block diagram models, extending the procedure for BGs [4]. Run-time changes are handled by reconfiguring the data flow paths within the blocks of the model. We demonstrate the technique by creating a computational model of an electrical power system in Matlab Simulink [5].

Related Work In our approach, hybrid behavior is implemented through idealized configuration changes, resulting in verifiable physically correct models [3]. Structurally, connections remain intact but the components' internal computations change form and the interpretation of their inputs and outputs changes in order to maintain a consistent computational structure. Other work in reconfigurable hybrid systems includes SHIFT [6] and R-Charon [7]. SHIFT can simulate dynamic networks of hybrid automata by allowing dynamic creation and destruction of components and their connections. R-Charon supports reconfiguration as well, based on formal definitions of syntax and operational semantics in Charon [8]. In contrast, our approach is based on the formal physical systems modeling semantics of HBGs. Existing approaches to simulation (e.g., in Dymola [9]) of switching behavior in BGs, such as [10, 11], do not perform on-line reconfiguration, thus must know switching behavior *a priori* to pre-enumerate possible modes, or do not exploit causality. Unlike these approaches, our approach does not assume any such *a priori* knowledge. We avoid pre-enumeration of system modes because of the formal basis in energy-based physical systems modeling, allowing for a systematic procedure to derive the computational structure of a new mode with knowledge of local switching behavior.

2 Modeling Complex Embedded Systems

A *system* is a collection of connected *components* where each component is defined by an internal behavior model and an interface through which the component interacts with other components and the environment. In our work, components are physical processes with hybrid behaviors. Component interfaces are defined by (i) energy ports for energy transfer, and (ii) signal ports for non-energy related information transfer. Component connections include energy and signal links. We model components as HBG fragments that contain BG elements, modulating functions, and control specifications. We briefly describe these constructs next. Details of the modeling paradigm are presented in [12].

BGs are domain-independent, topological, lumped-parameter models that capture the energy exchange mechanisms in physical processes [4]. The nodes of a bond graph are primitive elements that include energy storage (C and I), energy dissipation (R), energy transformation (TF and GY), and input-output elements (Se and Sf). The connecting edges called *bonds* are energy pathways between the elements. Each bond is associated with two variables: *effort* and *flow*. The product of effort and flow is power, i.e., the rate of energy transfer. Connections in the system are modeled by two idealized elements: 0- (or parallel) and 1- (or series) junctions. For a 0- (1-) junction, the efforts (flows) of all incident



(a) Circuit diagram. (b) Hybrid bond graph.

Fig. 1. Example system.

bonds are equal, and the sum of flows (efforts) is zero. Component parameters of nonlinear models are algebraic functions of other system variables and external input signals, called *modulating functions*.

HBGs extend continuous BG models by allowing discrete changes in system configuration at explicit time points by turning junctions on and off [3]. A finite state machine implements the junction *control specification* (CSPEC). Each state of the CSPEC maps to an *on* or *off* state of the junction, and the CSPEC transition guards are expressed as boolean functions of system and input variables. When a controlled junction is on, it behaves like a conventional junction. In the off state, all bonds incident on a 0- (1-) junction are deactivated by enforcing a zero effort (flow) at the junction. The system mode at any time is determined by composing states of the individual switched junctions.

To illustrate the modeling paradigm, we use an electrical circuit, shown in Fig. 1(a). The components of the circuit are a voltage source, $v(t)$, two capacitors, C_1 and C_2, two inductors, L_1 and L_2, two resistors, R_1 and R_2, and two switches, SW_A and SW_B. Resistor R_1 is modeled as a nonlinear resistance, where R_1 is a polynomial function g of the voltage drop across the inductor L_1, i.e., $R_1 = g(e_3)$. Fig. 1(b) shows the HBG model for this circuit. The switching junctions in the HBG, 1_a and 1_b, have associated CSPECs, denoted by CS_a and CS_b, respectively.

3 Computational Semantics of Hybrid Bond Graphs

Our goal is to build efficient simulation models from HBG representations. The block diagram (BD) formalism is a widely used graphical, computational scheme for describing simulation models of continuous and hybrid systems (e.g., Ptolemy [13], Modelica [14], and Simulink [5], among others). We adopt the BD modeling paradigm, and develop a methodology for transforming HBGs to BDs. We encounter two primary challenges in generating simulation models from HBG representations.

Challenge 1: Avoid pre-enumeration of model configurations. Consider a HBG model with m components and assume that each component has n_i switching junctions, where $i = 1, 2, \ldots m$. The HBG model, then, defines $2^{\sum_{i=1}^{m} n_i}$ different system modes (or model configurations). When $\sum_{i=1}^{m} n_i$ is large, it is infeasible to pre-enumerate all the model configurations before running the simulation. Therefore, mode changes, and reconfiguration of the BD model, have to be performed during run-time. Mode changes, implemented as junction switching, produce changes in the HBG model topology, which implies that the connections between blocks in the BD model may change dynamically during the simulations.

Challenge 2: Avoid algebraic loops. In component-based modeling, the underlying mathematical model is usually a set of differential-algebraic equations (DAEs) [9, 15]. The DAE models may include algebraic loops. A system of equations with algebraic loops has a *fixed point solution* if the conditions for a unique solution are satisfied [15]. However, generating the solution may become computationally expensive if the fixed point method needs many iterations to converge to a solution when algebraic loops are present. The order of equation evaluation, then, becomes very important [16].

3.1 Conversion of Continuous Bond Graphs to Block Diagrams

BG models imply a causal structure, and algorithms like the *Sequential Causal Assignment Procedure* (`SCAP`) [4] applied to well-formed BG models assign causal directions to all bonds in the model. The causal direction of a bond determines the functional relation between the associated effort and flow variables. When a BG model is in *integral causality*, which means that the constituent relations of its energy storage elements are expressed in their integral form, and the causal assignments of the bonds are unique, the derived BD model will have no algebraic loops [4]. Causality fixes the evaluation order of BG constituent equations to avoid the algebraic loops.

Fig. 2 shows the BD structure for each BG element [4]. The Sf, Se, C, and I elements have a single unique BD representation because their incident bonds have only one possible causal assignment. The R, TF and GY elements allow two causal representations each, and each one produces a different BD representation. A junction with m incident bonds can have m possible BD configurations. Mapping a junction structure to its BD is facilitated by the notion of the *determining bond*, which captures the causal structure for the junction.

Definition 1 (Determining Bond) *The* determining bond *for a* 0- *(*1-*) junction is the bond that determines the effort (flow) value for that junction.*

Fig. 2 shows the BD expansions for 0- and 1- junctions with bond 1 as the determining bond. For a 0-junction (1-junction), all other bonds' effort (flow) values are equal to the determining bond's effort (flow) value, and the flow (effort) value of the bond is the algebraic sum of the flow (effort) values of the other bonds that are connected to this 0- (1-) junction.

Fig. 3(b) shows the derived BD model for our example circuit with both switches on. The first step assigns causality to all bonds using the `SCAP` algorithm (Fig. 3(a)). The second step selects the BD model (from Fig. 2) for each BG element taking into account the causality of the incident bonds. The BD

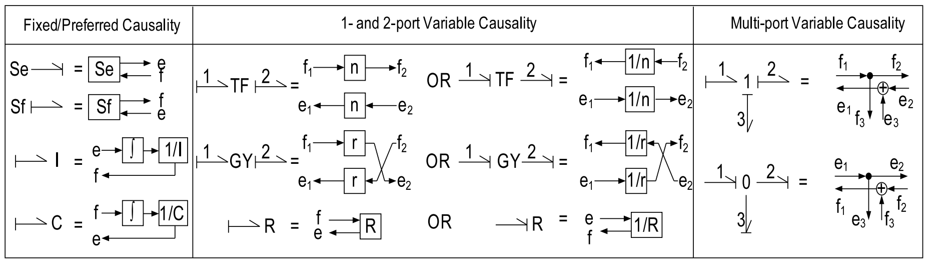
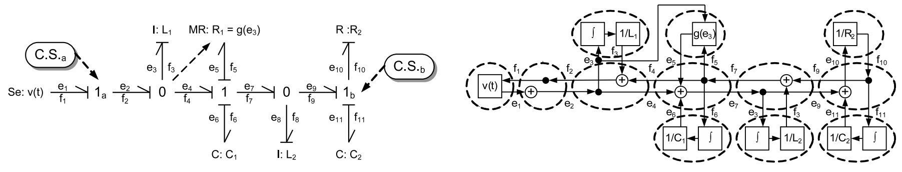

Fixed/Preferred Causality
1- and 2-port Variable Causality
Multi-port Variable Causality
Se
Sf
I
C
TF
GY
R
OR
1/I
1/C
1/n
1/r
1/R

Fig. 2. Block Diagram expansion of bond graph elements.



(a) Hybrid bond graph. (b) Block diagram.

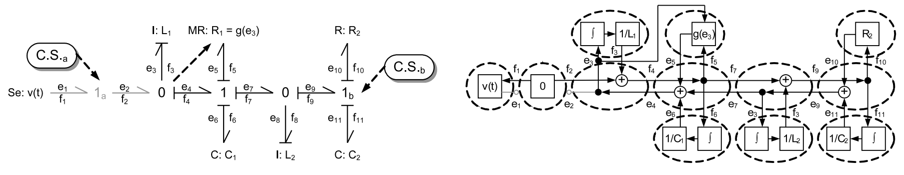
Fig. 3. Example system with both switches on.

fragments for each BG element are connected appropriately in step 3 to generate the BD model of the system.

As discussed, causal assignments define the data flow paths for variable values between the blocks. Because they fix the evaluation order of the equations, they minimize the occurrence of algebraic loops. There are three cases where the model may contain algebraic loops in the BD structure: (i) the BG model does not have a unique causal assignment, e.g., bonds 5, 7, 9, and 10 in Fig. 3(a) do not have unique assignments, and the resulting BD model has an algebraic loop; (ii) nonlinearities in the system, e.g.,the modulating function creates the algebraic loop: $e_5 \rightarrow e_4 \rightarrow e_3 \rightarrow g(e_3) \rightarrow e_5$ (see Fig. 4(b)); and (iii) the CSPEC function of hybrid models, e.g., if the state of a 1-junction is a function of its flow, the guard condition of the CSPEC produces an algebraic loop.

3.2 Conversion of Hybrid Bond Graphs to Block Diagrams

BD models derived from HBGs must incorporate mechanisms that allow reconfiguration when junctions switch state. When junctions switch on or off, the determining bond for that (and possibly other junctions) can change, which, in turn, causes changes in the implementations of blocks of the BD model. To avoid pre-enumeration of BD models for all system modes, an efficient component-based hybrid system simulation method must allow for on-line reconfiguration of the BD model. A naive solution may recompute the causal assignment on all bonds after the junction switching and derive a new BD model. But this may be wasteful because only a portion of the original BD structure may have changed. We implement an efficient BD reconfiguration scheme that recomputes the causal



(a) Hybrid bond graph. (b) Block diagram.

Fig. 4. Example system with junction 1_a off.

assignments incrementally, starting from junctions that switch state, and propagating determining bond changes till a new consistent assignment is derived. Corresponding changes are made only to those blocks in the BD structure that have changes in their determining bonds. This results in minimal changes to the BD structure before the simulation progresses. We first describe our method for incremental updating of determining bonds, and then present our procedure for building reconfigurable BD models from HBGs.

When a junction changes state, the choice of its determining bond may cause corresponding changes in the determining bonds of its neighbors, and this change can propagate. For example, in Fig. 3(a), if 1_b is switches off, the determining bond of its adjacent 0-junction does not change, and the rest of the BD structure is unchanged. If 1_a switches off, the determining bond at the adjacent 0-junction does change, and this change propagates step by step to adjacent junctions. In our example, to maintain integral causality, the I element's bond cannot switch causal direction, therefore, bond 4 becomes the determining bond. This change propagates to the adjacent 1-junction, and further up to 1_b, where the R element's bond switches its causal assignment and no further propagation is needed. Fig. 4(b) shows the resulting block diagram after the mode switch.

At junctions where a unique choice for a new determining bond is not known, an arbitrary choice may be made. But this choice may lead to an inconsistent assignment when the propagation reaches a junction whose determining bond is fixed by an incident source element or an energy-storage element. To prevent such inconsistent assignments, which requires a computationally expensive backtracking process, we identify active junctions that are in *forced causality* and *fixed causality* and avoid update paths that require determining bond changes for these junctions.

Definition 2 (Forced Causality) *For a given mode of system operation, an active junction is in* forced causality *if its determining bond is uniquely determined.*

Definition 3 (Fixed Causality) *An active junction is in* fixed causality *if, for* all *modes of system operation, its determining bond does not change.*

When a choice for determining bonds exists at a junction, we do not make a choice that would affect the determining bond of an adjacent junction already in

Algorithm 1 Hybrid SCAP

```
$UnassignedJunctionQueue$ = Set of switched junctions
while $UnassignedJunctionQueue$ is not empty do
  $j = UnassignedJunctionQueue$.pop()
  if choice of determining bond for $j$ is unique then
    Update determining bond of $j$
    $juncList$ =PropagateEffect($j$)
    $UnforcedQueue$.push($juncList$)
  else
    $UnforcedQueue$.push($j$)
while $UnforcedQueue$ is not empty do
  $j = UnforcedQueue$.pop()
  if Choice of determining bond for $j$ is unique then
    Update determining bond of $j$
    $juncList$ =PropagateEffect($j$)
    $UnforcedQueue$.push($juncList$)
  else
    if there exists a bond to an unvisited, unforced, unfixed junction to assign as
    determining bond then
      Choose that bond
    else
      Choose bond to a forced junction as determining bond
    Update determining bond of $j$
    $juncList$ =PropagateEffect($j$)
    $UnforcedQueue$.push($juncList$)
```

fixed or forced causality. For example, in Fig. 3(a), the first two junctions are in forced causality. The other junctions' determining bonds depend on an arbitrary choice of causality assignment to one of the R elements, so they are not in forced causality. In Fig. 4(a), all active junctions are in forced causality and there is only one consistent assignment of determining bonds for all active junctions.

We formalize this dynamic causality reassignment method as the *Hybrid Sequential Causal Assignment Procedure* (`Hybrid SCAP`) (see Algorithm 1). Fixed and forced causality information is computed for the initial mode with `Hybrid SCAP` and updated locally when mode changes occur. We assume that the new states of all junctions are available before `Hybrid SCAP` is applied. With the initial queue of switched junctions, `Hybrid SCAP` picks one junction off the queue, and makes all forced changes, and propagates the forced effects using `PropagateEffect` (Algorithm 2) up to junctions that are not forced or fixed. These junctions are added to $UnforcedQueue$. When all junctions in the initial queue are exhausted, the algorithm picks elements off the UnforcedQueue, assigns a determining bond and propagates its effects till it ends in a junction where another arbitrary choice can be made. If there exists a consistent causality assignment for this mode, the $UnforcedQueue$ eventually becomes empty. Otherwise, the current mode either does not support the integral causality assumption or its BG model is not well-formed.

Algorithm 2 PropagateEffect(j)

```
$juncList = []$
for all affected adjacent junction $adjJunc$ of $j$ do
   if choice of determining bond of $adjJunc$ is unique then
      Update determining bond of $adjJunc$
      $juncList += $PropagateEffect($adjJunc$)
   else
      $juncList += adjJunc$
return $juncList$
```

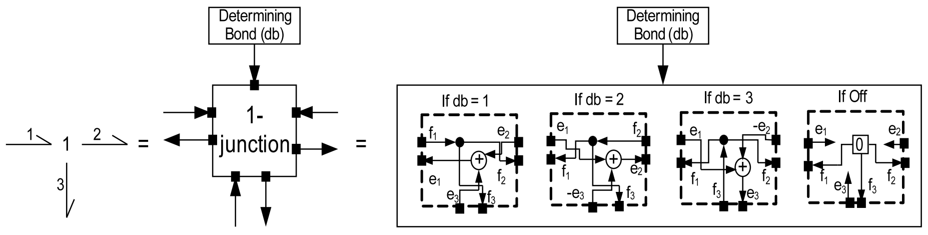


Fig. 5. Block Diagram expansion of a hybrid junction.

When the causal assignment of bonds change, the connected elements' input-output relationships change. Unlike BGs, where the causality assignment is fixed, the BD model for HBGs must consider multiple BD expansions for each element that can have variable causality. Moreover, the model needs mechanisms for reconfiguring on-line to accommodate the changing causality assignments.

Fig. 5 shows the BD representation for a hybrid 1-junction in a HBG model. Depending on its determining bond, the internal computation and the interpretation of its input and output signals change. For example, if the first bond is the determining bond, the corresponding input is flow, otherwise it is effort. The internal computation changes accordingly within the hybrid junction element block. R, TF, and GY elements are handled in a similar manner.

Using this reconfigurable BD allows us to exploit the advantages of causality in HBGs. Because `Hybrid SCAP` locally propagates effects of junction switching, the computation of the new causal structure is efficient. Since having a consistent causal assignment for each mode allows us to exploit causality, we minimize the number of algebraic loops in the computational model, and provide a fixed order of equation evaluation to solving the HBG constituent equations. Since the block diagram is reconfigurable, it also allows us to avoid pre-enumeration of all system modes since the HBG model supports automatic derivation of system modes at run-time.

4 Implementation

We now apply the HBG to BD transformation algorithms to derive a Simulink model that can be executed in the MATLAB environment. This derivation can be applied to other simulation environments in a similar fashion. BD structure reconfiguration attributed to junction switching in the HBG model is handled by switching functions that change the relationship between the input and output ports. This can be implemented in one of two ways.

Implicit Switching One implementation uses code in the form of conditional statements to model the variable input-output relation for a block element whose incident bond(s) can change causality. The switching of the data flow between blocks is implicit in the Simulink model. For R, TF, and GY elements, the implicit function is straightforward and directly linked to the causality assignment. For junctions, the input-output relations and the algebraic relations are directly a function of the determining bond and can be represented concisely[1]. Overall, the Simulink models generated by this approach are compact because all possible BD configurations are not explicitly enumerated using switches. However, representing junction blocks as implicit functions produces algebraic loops in the Simulink model, because the input-output directional structure is buried in the junction block. As a result, Simulink invokes its fixed point solver to solve for these algebraic loops during simulation. Though all of these structures produce a unique solution, there is computational overhead in invoking and applying the solver, and this may result in less efficient simulation runs.

Explicit Switching The alternative implementation uses Simulink switching elements to enumerate the possible data flow paths and computational structure for each configuration. At run time, when determining bonds change, the appropriate switches are triggered to activate the new effective block diagram structure. Each R, TF, and GY element requires a switch to model their two possible configurations, and a controlled junction with m incident bonds can have $m + 1$ possible configurations. Compared to the implicit switching method, the Simulink model created by this approach has many more atomic blocks. Since the data flow paths are made explicit for each configuration, no additional algebraic loops are created, but there is an overhead associated with zero-crossing detection.

The simulation model is created by an automatic model transformation procedure (*interpreter*) that operates on HBG models constructed in the modeling environment. The interpreter creates the simulation artifacts by first transforming the HBG into a BD, and then into a Simulink model. The implicit switching method uses Simulink S-functions implemented in C, and the explicit switching method uses multi-port Simulink switching element blocks.

Generation of the Simulink model from the BD is straightforward. For every component in the block diagram model, a corresponding Simulink subsystem with ports and internal blocks is instantiated that corresponds to the block diagram expansion of the element. These subsystems are connected exactly like the block diagram model.

[1] The determining bond's input sets the non-determining bonds' outputs, and the non-determining bonds' inputs, taking into account bond directions, sum to produce the determining bond's output.

For controlled junctions, the *control specification* is simplified to a two-state machine, with one state corresponding to on and the other to off. We evaluate the transition guards, and when enabled, the transition is taken to the new junction state. In both implementations, a global S-function tracks junction switches and calls the `Hybrid SCAP` implementation to compute the new determining bonds when this occurs.

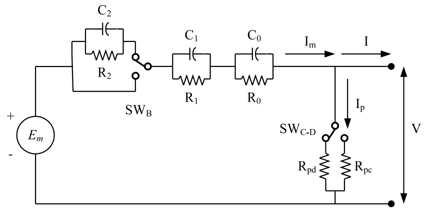
5 Case Study: Electrical Power System

We apply our modeling and simulation framework to the Advanced Diagnostics and Prognostics Testbed (ADAPT) system at NASA Ames. ADAPT is functionally representative of an electrical power system on a crew exploration vehicle, and serves as a testbed to study diagnostics and prognostics technologies. The system consists of three subsystems, *power generation*, consisting of a solar panel and battery chargers, *power storage*, consisting of three sets of lead-acid batteries, and *power distribution*, consisting of a number of AC and DC loads and AC to DC converters. Relays configure the system in different modes of operation, e.g., charge and/or discharge mode of the batteries as well as different power supply and load configurations. Therefore, the system behavior is naturally hybrid, with many possible modes. This makes it infeasible to pre-enumerate all possible modes of operation. The battery models are highly nonlinear and have different modes of operation corresponding to charging and discharging phases. In this paper, we derive simulation models from HBG models of the battery and load subsystems.

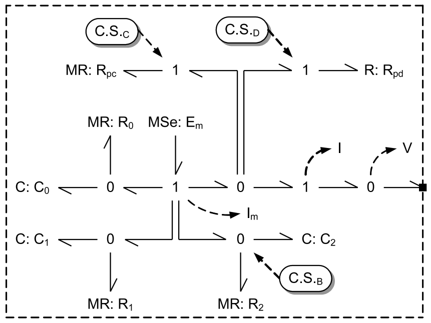
5.1 Component Models

Lead-acid Batteries An electrical equivalent circuit model (Fig. 6(a)) based on those presented in [17, 18] is developed. The current flowing through the battery is labeled as I. Part of the input current, I_m, goes into discharging/charging the battery. The rest of the current, I_p, is lost to parasitic reactions. During charge, the currents I and I_m flow opposite to the directions labeled.

The HBG model, shown in Fig. 6(b), is derived directly from the equivalent circuit diagram. Junction B is hybrid and corresponds to the switch SW_B in the circuit. It switches on or off autonomously, depending on the direction of current through the battery. When the battery is charging, i.e., current is negative, junction B is on, otherwise it is off. Junctions C and D are also hybrid and correspond to switch SW_{C-D}. When charging, C is on and D is off, otherwise D is on and C off. When charging, the parasitic resistance models the charge acceptance of the battery (based on the model in [19]), and when discharging the parasitic resistance is a large constant resistance, modeling the slow drain of charge when the battery is not being charged. Loads connect in parallel to the power port in the HBG. Because the HBG has 3 controlled junctions, it has 8 total modes.



(a) Battery equivalent circuit diagram



(b) Battery component hybrid bond graph

Fig. 6. Hybrid nonlinear battery model

Table 1. Nonlinear battery parameters

Parameter	Equation
E_m	$E_{m0} + A_m(SOC)(273 + K_m\theta)$
R_0	$R_{00} + A_0(SOC)(273 + K_0\theta)$
R_1	$-R_{10} \ln DOC$
R_2	$R_{20} \frac{\exp A_{21} SOC}{(1+exp(A_{22} I_m/I^*))} + A_{23}$
R_{pc}	$V/(-I \exp \frac{A_{p1}}{-I/I^*+A_{p2}} \exp A_{p3} SOC)$

A number of the HBG model parameters are modulated to represent its nonlinear behavior, and are derived from parameter equations given in [17]. The equations are listed in Table 1, and the constant values appear in Table 2. Many of these parameter values depend on the state of charge, which is a function of the extracted charge of the battery, Q_e, and the battery capacity, $C(I_m, \theta)$, which itself is a function of temperature, θ [17].

Battery capacity is computed as a function of the discharge current and temperature by $C(I_m, \theta) = K_c C_0^*(1+\theta/\theta_f)^\epsilon/(1+(K_c-1)(I_m/I^*)^\delta)$, where θ_f is the electrolyte freezing temperature (typically -35^oC), and I^* is a reference current. The remaining parameters are design constants. Battery extracted charge, Q_e, is a function of the current flowing through the battery, i.e., $\dot{Q}_e(t) = I_m(t)$.

Battery *state of charge* (SOC) and *depth of charge* (DOC) are computed based on Q_e and battery capacity as $SOC = 1 - Q_e/C(0, \theta)$, and $DOC = 1 - Q_e/C(I_m, \theta)$. SOC represents the state of charge of the battery based on its nominal capacity and DOC represents the state of charge of the battery based on its capacity relative to the present discharge current.

Battery temperature, θ, is described by a capacitance-resistance model. It is a function of the heating power generated by the battery, P_h, and ambient temperature, θ_a given by $\dot{\theta}(t) = 1/C_\theta\,(P_h + (\theta - \theta_a)/R_\theta)$. The equations describing

Table 2. Battery model constant values

Capacity parameters	$I^* = 5, K_c = 1.33e0, C_0^* = 2.71e5, \theta_f = -35, \epsilon = 6.42e-1,$ $\delta = 6.10e-1$
Main branch parameters	$E_{m0} = 2.14e1, A_m = 1.14e-2, K_m = 7.59$ $R_{00} = 2.30e-3, A_0 = 1.12e-4, K_0 = -1.37e-1,$ $C_0 = 5.00e1, R_{10} = 2.15e-2, C_1 = 5.00e3$ $R_{20} = 1/37e-6, A_{21} = 1.83e1, A_{22} = 7.83e-1,$ $A_{23} = 1.96e-2, C_2 = 2.72e3$
Parasitic branch parameters	$R_{pd} = 5.00e2, A_{p1} = -9.35e1, A_{p2} = 1.89e1, A_{p3} = 4.22e0$
Temperature parameters	$R_\theta = 1.00e-2, C_\theta = 2.44e6$

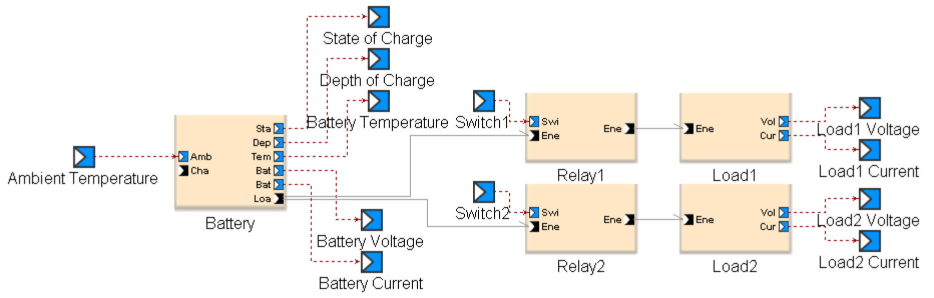


Fig. 7. Battery and DC loads configuration FACT model

extracted charge and temperature can also be represented as HBG model fragments, but are omitted from Fig. 6(b) for clarity.

Relay The relay HBG model consists of a single controlled 1-junction with two power ports. The junction models the relay's switching behavior, by turning on or off depending on some external control signal. Since the HBG has 1 controlled junction, it has 2 total modes.

Resistive Load The resistive load HBG model consists of a single power port, a 1-junction, and a R element. The R element parameter value is constant.

5.2 Simulation Results

We present the simulation results for the battery supplying power to two DC loads in parallel. Relays in the circuit enable the loads to be switched online or offline. The configuration, created in our modeling environment [12], is shown in Fig. 7. As previously described, the battery has three controlled junctions, and each relay has one, resulting in a possibility of 32 modes. Even with a few number of components, the advantage of our approach is clear. Enumerating all 32 modes is wasteful, especially since in this particular configuration, only a few of these modes will occur.

Fig. 8 shows the results of the simulation. The Simulink model was run for 7,000 seconds of simulation time with the battery discharging through different load conditions. The battery is initially fully charged. At 500 seconds, *Load*1 is

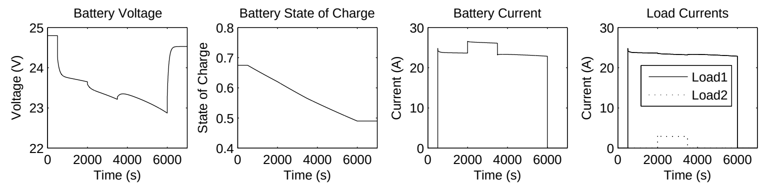


Fig. 8. Simulation results

attached, causing a sharp drop in the battery voltage and a sharp rise in the current. The battery state of charge begins to decrease as current flows through the load, causing a gradual drop in battery voltage. At $2,000$ seconds, $Load2$ is brought online. Because the loads are in parallel, the effective resistance of the load on the battery decreases, causing an increase in the current drawn, and a decrease in the voltage. At $3,500$ seconds, $Load2$ is disconnected and the current falls and the voltage rises. At $6,000$ seconds $Load1$ is disconnected, and the battery then goes back to its no load voltage output. This voltage is lower than the initial no load voltage because of the decrease of charge in the battery.

For the configuration tested, the explicit switching implementation executed about 20% faster than the implicit switching implementation. We have run a number of other configurations with Simulink, and the run-time differences between the implicit and explicit switching is about the same as reported above. Both implementations exploit causality, and even though the implicit switching implementation introduced unnecessary algebraic loops, the use of causality in these functions allowed the algebraic loop solver to converge quickly, which is why the efficiency difference was not very substantial.

6 Conclusions

We presented a modeling and simulation framework for systems design that is closely tied to the semantics of physical system principles. Subsystem components can be composed to generate the overall system model, just as designers assemble subsystems to form the complete system. Moreover, HBGs being domain independent, this framework will be useful for engineers in a variety of domains. When the switching behavior of the model is not known in advance (as is true in general), then the only way to avoid pre-enumeration of all the system modes is to do on-line reconfiguration. The main contribution of this approach is the on-line reconfiguration of the BD model which fully exploits causality in HBGs by incorporating dynamic causality reassignment. We also demonstrated the usefulness of the approach in modeling a complex system of practical value. As part of future work, we intend to apply this approach to more complex configurations of ADAPT, such as one involving DC-AC converters and AC loads, as well as other large real-world systems.

Acknowledgments This work was supported in part by grants NSF CNS-0347440 and NSF CNS-0615214, and a NASA USRA grant from Ames Research Center. We gratefully acknowledge the help from Scott Poll and Ann Patterson-Hine in building the ADAPT system models.

References

1. Liu, J., Lee, E.A.: A component-based approach to modeling and simulating mixed-signal and hybrid systems. ACM Trans. Model. Comput. Simul. **12**(4) (2002) 343–368
2. Lee, E.A., Neuendorffer, S., Wirthlin, M.J.: Actor-oriented design of embedded hardware and software systems. Journal of Circuits, Systems, and Computers **12**(3) (2003) 231–260
3. Mosterman, P.J., Biswas, G.: A theory of discontinuities in physical system models. J Franklin Institute **335B**(3) (1998) 401–439
4. Karnopp, D.C., Margolis, D.L., Rosenberg, R.C.: Systems Dynamics: Modeling and Simulation of Mechatronic Systems. Third edn. John Wiley & Sons, Inc., New York (2000)
5. MATLAB/Simulink: (http://www.mathworks.com/products/simulink/)
6. Deshpande, A., Gollu, A., Semenzato, L.: The SHIFT programming language for dynamic networks of hybrid automata. IEEE Transactions on Automatic Control **43**(4) (1998) 584–587
7. Kratz, F., Sokolsky, O., Pappas, G.J., Lee, I.: R-Charon, a Modeling Language for Reconfigurable Hybrid Systems. In: Hybrid Systems: Computation and Control. Volume 3927 of LNCS. Springer (2006) 392–406
8. Alur, R., Grosu, R., Hur, Y., Kumar, V., Lee, I.: Modular specification of hybrid systems in Charon. In: Hybrid Systems: Computation and Control. Volume 1790 of LNCS. Springer-Verlag (2000) 6–19
9. Dymola: (http://www.dynasim.com/dymola.html)
10. Cellier, F.E., Otter, M., Elmqvist, H.: Bond graph modeling of variable structure systems. In: Proc. ICBGM. (1995)
11. Edstrom, K., Stromberg, J.E., Soderman, U., Top, J.L.: Modelling and simulation of a switched power converter. In: Proc. ICBGM. (1997)
12. Manders, E.J., Biswas, G., Mahadevan, N., Karsai, G.: Component-oriented modeling of hybrid dynamic systems using the Generic Modeling Environment. In: Proc of the 4th Workshop on Model-Based Development of Computer Based Systems, Potsdam, Germany, IEEE CS Press (2006)
13. Buck, J., Ha, S., Lee, E.A., Messerschmitt, D.G.: Ptolemy: a framework for simulating and prototyping heterogeneous systems. Readings in hardware/software co-design (2002) 527–543
14. Modelica: (http://www.modelica.org/)
15. Griepentrog, E., Mārz, R.: Differential-algebraic equations and their numerical treatment. Teubner (1986)
16. Pinto, A., Carloni, L.P., Passerone, R., Sangiovanni-Vincentelli, A.: Interchange Format for Hybrid Systems: Abstract Semantics. In: Hybrid Systems: Computation and Control. Volume 3927 of LNCS. Springer (2006) 491–506
17. Ceraolo, M.: New dynamical models of lead-acid batteries. IEEE Transactions on Power Systems **15**(4) (2000) 1184–1190

18. Barsali, S., Ceraolo, M.: Dynamical models of lead-acid batteries: Implementation issues. IEEE Transactions on Energy Conversion **17**(1) (2002) 16–23
19. Guasch, D., Silvestre, S.: Dynamic battery model for photovoltaic applications. Progress in Photovoltaics: Research and Applications **11**(3) (2003) 193–206