Simulation

Simulation: Transactions of the Society for Modeling and Simulation International
87(6) 467–498
© The Author(s) 2010
Reprints and permissions:
sagepub.co.uk/journalsPermissions.nav
DOI: 10.1177/0037549710364478
sim.sagepub.com

Efficient simulation of hybrid systems: A hybrid bond graph approach

Indranil Roychoudhury[1], Matthew J Daigle[2], Gautam Biswas[3] and Xenofon Koutsoukos[3]

Abstract
Accurate and efficient simulations facilitate cost-effective design and analysis of large, complex, embedded systems, whose behaviors are typically hybrid, i.e. continuous behaviors interspersed with discrete mode changes. In this paper we present an approach for deriving component-based computational models of hybrid systems using hybrid bond graphs (HBGs), a multi-domain, energy-based modeling language that provides a compact framework for modeling hybrid physical systems. Our approach exploits the causality information inherent in HBGs to derive component-based computational models of hybrid systems as reconfigurable block diagrams. Typically, only small parts of the computational structure of a hybrid system change when mode changes occur. Our key idea is to identify the bonds and elements of HBGs whose causal assignments are invariant across system modes, and use this information to derive space-efficient reconfigurable block diagram models that may be reconfigured efficiently when mode changes occur. This reconfiguration is based on the incremental reassignment of causality implemented as the Hybrid Sequential Causal Assignment Procedure, which reassigns causality for the new mode based on the causal assignment of the previous mode. The reconfigurable block diagrams are general, and they can be transformed into simulation models for generating system behavior. Our modeling and simulation methodology, implemented as the Modeling and Transformation of HBGs for Simulation (MoTHS) tool suite, includes a component-based HBG modeling paradigm and a set of model translators for translating the HBG models into executable models. In this work, we use MoTHS to build a high-fidelity MATLAB Simulink model of an electrical power distribution system.

Keywords
component-oriented modeling, dynamic causal assignment, hybrid bond graphs, modeling and simulation environments, simulation of hybrid systems

1. Introduction

Accurate and efficient simulations facilitate cost-effective design and behavior analysis of large, complex, embedded systems. For example, high-fidelity simulations help to reduce expensive and time-consuming design tasks, such as prototype building and testing. In addition, simulation models help to speed up the design, implementation, testing, and validation of control, diagnosis, and prognosis algorithms built to ensure safety and reliability in system operations. Moreover, such models provide a comprehensive framework for experimental analysis of a variety of fault scenarios that cannot be easily introduced into real systems.

However, developing simulations of large-scale embedded systems is non-trivial, since these complex systems may consist of a large number of interacting components with *hybrid* (i.e. mixed continuous and discrete) behaviors that are represented by a discrete set of system *modes*, with each defining a new set of

[1]SGT Inc., NASA Ames Research Center, Moffett Field, CA 94035, USA.
[2]University of California, Santa Cruz, NASA Ames Research Center, Moffett Field, CA 94035, USA.
[3]Institute for Software Integrated Systems, Department of Electrical Engineering and Computer Science, Vanderbilt University, Nashville, TN 37235, USA.

Corresponding author:
Indranil Roychoudhury, SGT Inc., NASA Ames Research Center, Moffett Field, CA 94035, USA
Email: indranil.roychoudhury@nasa.gov

continuous dynamics. To represent the discrete operating modes of a hybrid system, some modeling languages, such as hybrid automata,[1] require the pre-enumeration of all system modes explicitly. However, real-world hybrid systems typically involve a large number of complex interacting subsystems, where explicit enumeration of all system modes is infeasible. Building models of hybrid systems, therefore, is greatly facilitated by using component-based modeling, since component models are reusable, and modular system models are easier to update and maintain. The composition schemes for component-based models provide the advantage of building and validating models of components individually before they are composed to create the overall system model. Further, these schemes dynamically derive the overall system mode as a *composition* of individual component modes. Therefore, the modeler specifies individual (local) component modes and does not have to work through the explicit enumeration of all (global) system modes, although mode changes must be handled at the component level. As a result, the component-based models concisely represent hybrid system behaviors, but the effects of local changes in individual component modes may force other components to reconfigure their computational structures during the simulation process. This is particularly true when model transitions occur at fast rates, such as in modern electronics-based electrical power systems.[2]

In this paper, we present a methodology for automatic generation of simulation testbeds for hybrid systems modeled as hybrid bond graphs (HBGs).[3,4] The HBG modeling language allows for multi-domain, physics-based modeling of embedded systems. HBGs extend bond graphs (BGs)[5] by incorporating idealized switching functions which allow the energy flow paths to be reconfigured whenever a mode change occurs in the system. In previous work,[6] we have developed a component-based, hierarchical scheme for modeling hybrid systems, where the system is modeled as a collection of interconnected system components, each of which is modeled as an HBG fragment. In hierarchical HBGs, the components can be organized into different levels of abstraction, and connected to each other through energy and signal ports.

We exploit the causal structure inherent in HBGs to derive efficient hybrid simulation models as reconfigurable block diagram (BD) structures. Causality defines the computational directions of signals in HBG (and, hence, BD) components. During behavior generation, as the system transitions from one mode to another, the causal structure of the HBG model changes, which requires runtime reconfigurations of the BD models. To achieve efficiency during simulation, we need to minimize the computations associated with the reconfiguration process, and this is achieved by recognizing that only small parts of the model typically change causality between consecutive mode changes. Our methodology identifies bonds and HBG elements whose causal assignments, and, therefore, corresponding BD structure configurations, are invariant from one mode of system operation to another, and also across all modes. We use this information to construct space-efficient simulation models that do not include causal configurations that can never occur during simulation. Our causal reassignment procedure for HBGs, *Hybrid Sequential Causal Assignment Procedure* (`Hybrid SCAP`), builds upon the causal assignment procedure for BGs to incrementally reassign causality of the new mode of the HBG based on the causal assignment of the previous mode and known invariant causal configurations. By avoiding the causal reassignment of the entire system HBG, when only a small subset of the HBG requires causal reassignment, we significantly reduce the computational cost of model reconfiguration, and thus improve simulation efficiency. In addition, we exploit the knowledge of invariant causal configurations and avoid unnecessary calls to our causal reassignment procedure, thus further improving the simulation efficiency.

We implement our approach to building simulation models as the **Mo**deling and **T**ransformation of **HBG**s for **S**imulation (MoTHS) tool suite (available for download at http://macs.isis.vanderbilt.edu/software/MOTHS) in the Generic Modeling Environment (GME),[7] a meta-modeling framework for defining domain-specific modeling languages. MoTHS consists of a modeling language for graphically building component-based HBG models, and a set of model translators that convert these HBGs into executable models that can be simulated using block-oriented graphical solvers. Mature, well-developed, graphical modeling simulation software, such as MATLAB® Simulink®,[8] are usually preferred over non-graphical modeling approaches because of their ease of use. Currently, MoTHS incorporates a model translator that creates MATLAB Simulink models, however, model translators for different simulation applications can be easily incorporated into MoTHS by building interpreters in GME. By translating HBGs to simulation models for existing simulation environments, we can leverage the sophisticated and robust solvers of these simulation applications for these models.

To demonstrate the effectiveness of our approach, we use MoTHS to build a simulation testbed for the Advanced Diagnostics and Prognostics Testbed (ADAPT) at the NASA Ames Research Center,[9] called VIRTUAL ADAPT. ADAPT is a representative electrical power distribution system for spacecraft that presents a number of modeling challenges, since it contains a large number of switching elements (over 50 relays and circuit breakers), fast-switching power-conversion systems (inverters), and several interacting components with complex non-linear dynamics. Moreover, ADAPT is multi-domain because it contains: electromechanical components, such as fans; hydraulic components, such as pumps; and electrochemical components, such as batteries. Therefore, ADAPT serves as an ideal example to highlight the effectiveness of our simulation approach, since domain-specific simulation programs, such as circuit simulation applications, are not suitable for simulating ADAPT. We provide experimental results to demonstrate the effectiveness of VIRTUAL ADAPT in simulating faulty and nominal system behavior, and illustrate the efficiency of our approach.

This paper is organized as follows. Section 2 reviews the HBG modeling language, and Section 3 presents an overview of our modeling and simulation methodology. Section 4 describes the generation of reconfigurable BDs from HBGs. Section 5 explains our causal reassignment procedure, and Section 6 describes how we simulate the BD models. Section 7 describes the implementation of the MoTHS tool suite. Section 8 presents the case study and experimental results. Section 9 presents related research in simulation schemes for hybrid systems, and compares these approaches with ours. Section 10 concludes the paper and presents future work.

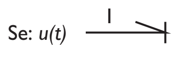
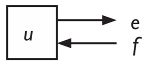
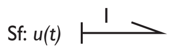
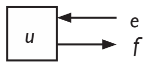
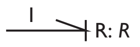
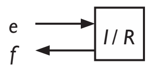
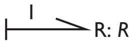
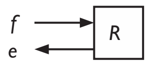
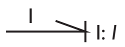
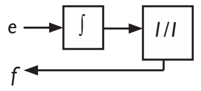
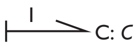
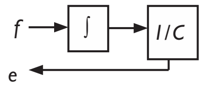
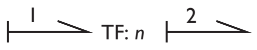
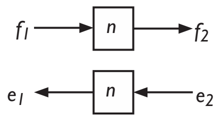
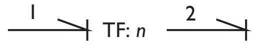
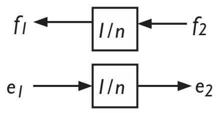
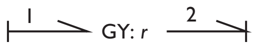
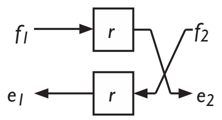
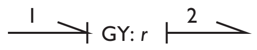
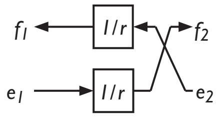
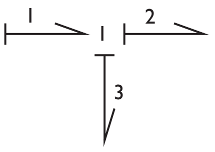
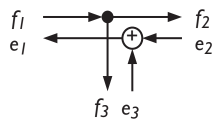
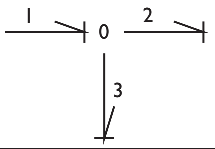
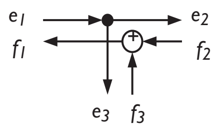
2. The hybrid bond graph modeling language

We adopt the HBG modeling language,[3,4] an extension of the BG language,[5] for component-based modeling of hybrid physical systems. An HBG is a graph whose vertices are primitive elements that include energy storage elements (capacitors, C, and inductors, I), energy dissipation elements (resistors, R), energy transformation elements (transformers, TF, and gyrators, GY), energy source elements (sources of effort, Se, and sources of flow, Sf), and junctions (0- and 1-junctions) through which all other HBG elements are connected to one another. Table 1 presents the different HBG elements and their constituent equations. The edges that connect these vertices are called *bonds* and are drawn as half arrows. Bonds represent energy pathways between the connected elements. Each bond is associated with two variables: *effort*, e, and *flow*, f, where the product of effort and flow is power, i.e. the rate of energy transfer between components. The effort and flow variables map to different physical variables in different energy domains. For example, effort and flow variables, respectively, map to voltage and current in the electrical domain, force and velocity in the mechanical domain, pressure difference and volumetric (or mass) flow rate in the hydraulic domain, and temperature difference and rate of flow of heat in the thermal domain. The 0- and 1-junctions represent idealized connections in the system. For a 0- (or 1-) junction, the efforts (or flows) of all incident bonds are equal, and the sum of flows (or efforts) is zero. For example, in the electrical domain, 0-junctions model (common voltage) parallel connections, and 1-junctions model (common current) series connections. The n-port I- (or C-) *fields* extend the 1-port energy storage elements, so that each n-port I- (or C-) field computes the value of n flows (or efforts) it determines based on the n input efforts (or flows). The parameter of each I- or C-field, therefore, is an $n \times n$ matrix, as opposed to the single scalar parameter values associated with I and C elements. Similarly, n-port R-fields extend the 1-port R elements. In Section 8, n-port elements are used to build some component models for our case study, ADAPT. For non-linear systems, component parameters can be algebraic functions of system variables and external input signals. We implement components with non-linear behaviors as *modulated* HBG elements, and denote them by an 'M' prefixed to the standard symbol for that element (e.g. a modulated resistance is denoted as MR).

HBGs, like BGs, are intended for modeling physical systems. They extend BGs by introducing idealized *switching junctions* (also called controlled junctions[3]), which allow the incorporation of mode changes. When a switching junction is on, it behaves like a conventional non-switching junction. When off, all bonds incident on a junction are deactivated by enforcing 0 flows (for a 1-junction) or 0 efforts (for a 0-junction) on these bonds (see Figures 1 and 2). A finite state machine implements the junction *control specification* (CSPEC). Each state of the CSPEC maps to either the *on* or *off* mode of the junction. Transitions between the CSPEC states, and hence, junction modes, are defined by Boolean-valued functions of system variables and/or system inputs, called *guards*. A mode transition is termed *controlled* if its occurrence is exclusively a result of control signals that are external to the physical process, or *autonomous*, if its occurrence is a result of internal system variables (e.g. a circuit breaker tripping due to an increase in current flowing through it). Since guards may include both endogenous and exogenous variables, the CSPECs of HBGs may capture either type of mode transition.

Example 1. To illustrate our methodology, we use a simple electrical circuit, shown in Figure 3(a), as a running example. The components of the circuit are a voltage source, $v(t)$, capacitors, C_1 and C_2, inductors,

Table 1. Causal assignments and block diagrams of linear hybrid bond graph elements

Element	Equations	Block Diagram
	$e_1 = u$	
	$f_1 = u$	
	$f_1 = e_1 / R$	
	$e_1 = R f_1$	
	$f_1 = \int e_1 / I \, dt$	
	$e_1 = \int f_1 / C \, dt$	
	$e_1 = n e_2$ $f_2 = n f_1$	
	$e_2 = e_1 / n$ $f_1 = f_2 / n$	
	$e_1 = r f_2$ $e_2 = r f_1$	
	$f_2 = e_1 / r$ $f_1 = e_2 / r$	
	$f_1 = f_2 = f_3$ $e_1 = e_2 + e_3$	
	$e_1 = e_2 = e_3$ $f_1 = f_2 + f_3$	

L_1 and L_2, resistors, R_1 and R_2, a relay, Sw_1, and a circuit breaker, Sw_2. The modulated resistance R_1 is modeled as a function, g, i.e. $R_1 = g(e_3)$, where e_3 is the voltage drop across L_1. We term the function, g, the *modulating function* of resistance R_1. Figure 3(b) shows the HBG model for this circuit. There is a one-to-one mapping between electrical circuit elements and HBG elements. The switching junctions in the HBG, 1_a and 1_e, have associated CSPECs, denoted by $CSPEC_a$ and $CSPEC_e$, respectively. The two CSPECs

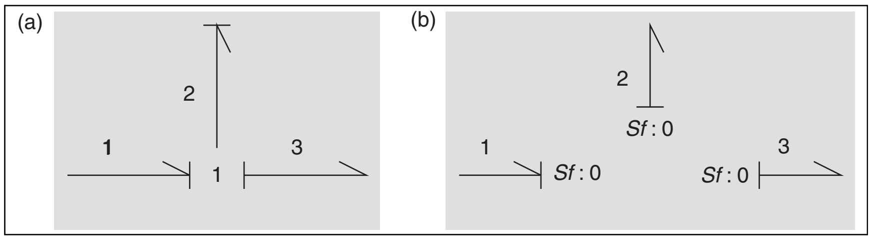


Figure 1. Semantics of a switching 1-junction: (a) on configuration; (b) off configuration.

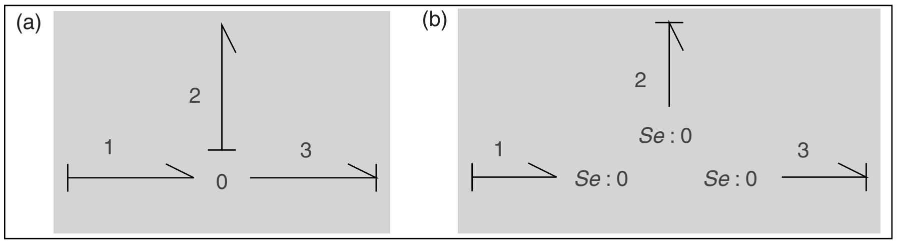


Figure 2. Semantics of a switching 0-junction: (a) on configuration; (b) off configuration.

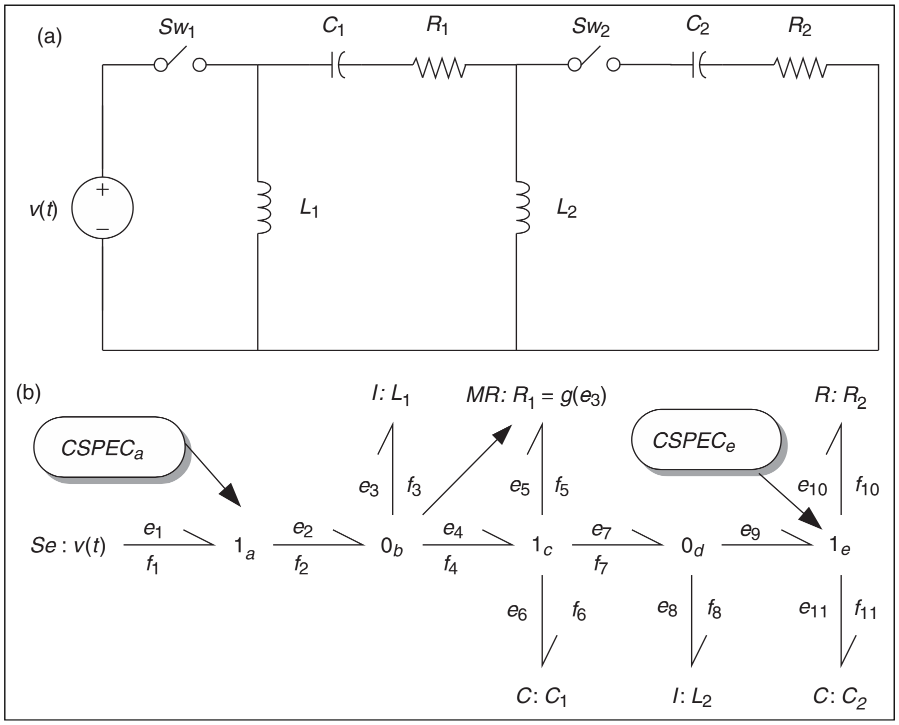


Figure 3. Example system: (a) circuit diagram; (b) hybrid bond graph.

are shown in Figure 4. In the example, the switching signal, *sw*, is an external Boolean signal that determines whether the relay is on or off. When *sw* goes from low to high, the junction goes from its off to on mode, and when the signal goes from high to low, the junction goes from its on to off mode. The circuit breaker turns off when the current through it, f_9 (where $f_9 = f_{10} = f_{11}$), exceeds the current limit (10 A), and is reset by an external command, *reset*.

3. Building simulation models from hybrid bond graphs: An overview

We develop a space- and time-efficient simulation framework for hybrid systems modeled as HBGs. Computational models of dynamic systems are typically expressed as ordinary differential equations (ODEs), differential algebraic equations (DAEs), and BD models. We choose the BD formulation as the

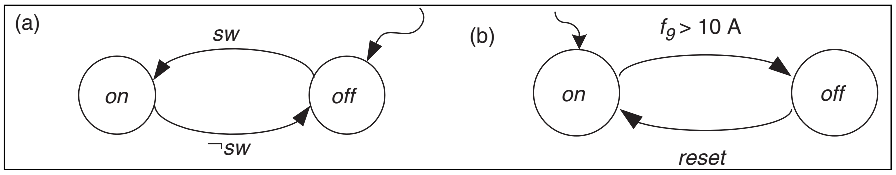


Figure 4. Example CSPECs: (a) relay CSPEC (*sw* denotes the switching signal); (b) circuit breaker CSPEC (*reset* denotes the external reset signal).

underlying computational model for HBGs for three main reasons: (i) it is a widely used visual computational scheme for describing simulation models of continuous and hybrid systems (e.g. Simulink,[8] Ptolemy[10]) (ii) the input–output formulation of each block in a BD can be determined using the causality information provided by HBGs,[11] and (iii) BD models can preserve the component structure of the model, which facilitates introduction of faults into components for simulation-based diagnosis and prognosis studies.

3.1. Causality

We use the causality information embedded in HBGs to derive BD models and facilitate their reconfiguration. *Causality* expresses the computational dependencies between the effort and flow variables in the system, and determines the independent variable at each bond, thus describing the input–output behavior of each component. In our approach, we assume that the energy storage elements (C and I) are in *integral causality*, which means that their constituent equations are expressed in integral form, e.g., for a C element, $e = (1/C)\int f\,dt$. The corresponding differential form, i.e. $f = C(de/dt)$, known as *derivative causality*, may introduce numerical problems during simulation, and also requires knowing a future value to correctly compute the derivative at the current time point. Hence, we assume that all energy storage elements can be assigned integral causality in all modes. The consequences of relaxing this integral causality assumption for energy storage elements are discussed briefly in Section 10.

Table 1 shows the possible causal assignments for linear BG (and, hence, HBG) elements.[5] The possible causal assignments remain unchanged for non-linear HBG elements as modulating functions do not change their effort–flow causality. The Sf, Se, C, and I elements each have a unique causal assignment on their incident bonds. The n-port I- and C-fields also have unique causal assignments, and being simple functional and structural extensions to the 1-port I and C elements, are not shown in Table 1. The R, TF, and GY elements allow two possible causal assignments each. We capture the causal assignment of a junction through the commonly used notion of a *determining bond*.

Definition 1 (Determining bond). The *determining bond* of a 0- (or 1-) junction is the bond that establishes the effort (or flow) value for all other bonds incident on that junction.

For example, in Table 1, bond 1 is the determining bond of the 1-junction, as it is the single bond that imposes flow on the 1-junction. The causal structure of an HBG model can be represented concisely by the determining bonds of all of the junctions, since every single bond must be connected to a junction (either directly, or through a TF or GY element), and the determining bond of a junction determines the causality of all other bonds of the junction.

A standard algorithm for assigning causality to a BG is the *Sequential Causal Assignment Procedure* (SCAP).[5] Although SCAP is most commonly used,[12] several other causal reassignment approaches exist in literature, e.g., *Modified SCAP* (MSCAP),[13] *Relaxed Causal Assignment Procedure* (RCAP),[14] and *Lagrangian Causal Assignment Procedure* (LCAP).[15] The SCAP algorithm starts at elements having a unique causal assignment, i.e. first at the energy source elements (Sf and Se) and energy storage elements (C and I), and assigns causality. Then the constraints of this causal assignment propagate to adjacent junctions, since the possible options for determining bonds become constrained. If the determining bond for a junction can be assigned *uniquely*, the junction is assigned that determining bond, and the constraints imposed by this assignment are propagated along the BG to further restrict the possible options for determining bonds at other junctions. After the causal changes have been propagated from all energy source and storage elements, if some junctions are yet to be assigned a determining bond, SCAP arbitrarily chooses an R element connected to one such junction, and assigns causality such that the bond connecting to the R element becomes the adjacent junction's determining bond. The constraints imposed by this assignment are then propagated along the BG, and the process continues until every junction has been assigned a determining bond and the overall assignment is consistent.

Example 2. Figure 5(a) shows a possible causal assignment for the configuration with both junctions on. SCAP starts at the Se, which imposes effort on

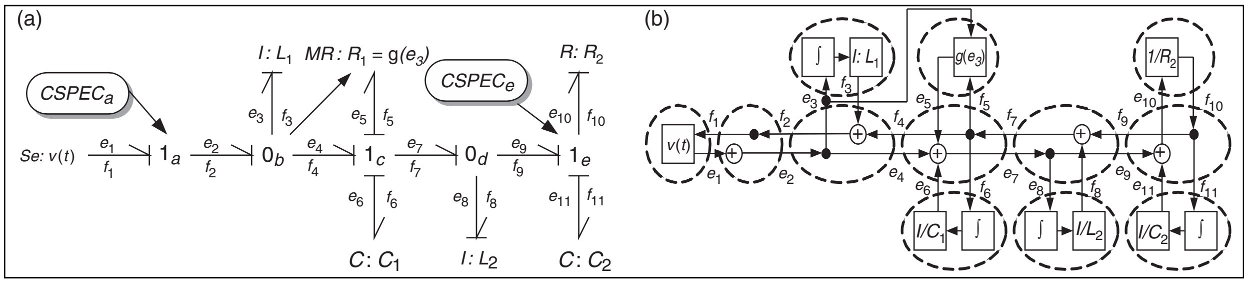


Figure 5. Example system with both switches on: (a) hybrid bond graph; (b) block diagram.

junction 1_a. Since a junction can have only one determining bond, bond 2 is the only possible determining bond for junction 1_a, and SCAP makes this assignment. This causal assignment, in turn, makes bond 2 the determining bond for junction 0_b. Since only one bond can impose effort at a 0-junction, and bond 3 imposes flow on 0_b (because inductor L_1 must be in integral causality), SCAP assigns the causality at bond 4 such that this bond imposes effort at junction 1_c. Capacitor C_1 also imposes effort at this junction through bond 6. At this point, bonds 5 and 7 do not have a causal assignment, and SCAP cannot assign junction 1_c a determining bond. So, SCAP inspects other junctions connected to energy sources or energy storage elements. Capacitor C_2 imposes effort on junction 1_e through bond 11, while the inductor L_2 imposes flow on junction 0_d through bond 8. Even after propagating all possible constraints generated by the energy source and storage elements, junctions 1_c, 0_d, and 1_e, have multiple options for their determining bonds. Therefore, SCAP arbitrarily assigns bond 10 (which is connected to resistor R_2) as the determining bond of junction 1_e. As a result, bond 9 imposes flow on junction 0_b, and SCAP assigns bond 7 to be the determining bond of junction 0_d. Bond 7 also becomes the determining bond of junction 1_c, and, since all of the assignments are consistent and complete, SCAP terminates.

3.2. Using causality to derive block diagrams

Once a BG is assigned causality, a well-defined procedure can be applied to convert the BG to a BD structure,[5] an alternative representation for the constituent equations of the elements. Based on the assigned causality, each 1- and 2-port BG element is replaced by its corresponding BD structure (see Table 1). Note that Table 1 illustrates the BD for linear HBG elements. For non-linear HBG elements, the BD for each corresponding linear HBG element includes the extra input signals for its modulating function that determines the parameter value of the element. Mapping a junction to its BD model is facilitated by its determining bond. Table 1 shows one possible BD expansion for a 1- and a 0-junction. At a 1- (or 0-) junction, all other bonds' flow (or effort) values are equal to the determining bond's flow value, and the effort (or flow) value of the determining bond is the algebraic sum of the effort (or flow) values of the other bonds connected to this 1- (or 0-) junction, taking into account the direction of the bonds, which define the signs in the summation. Therefore, a non-switching junction with m incident bonds can have m possible BD configurations. Once all necessary blocks of the BD are instantiated, the signals are connected appropriately to complete the BD model. Figure 5(b) shows the BD model for the HBG in Figure 5(a) for the mode with both junctions on. Note the extra input in the BD for the modulating function, g, of the non-linear resistance, R_1.

The procedure described above, however, is not suitable for generating BD models of HBGs, since the causal structure of HBGs can change as the system mode changes. When a junction switches on or off, its determining bond is activated or deactivated (see Figure 1), HBG components are connected and/or disconnected from the junction, and this reconfiguration changes the junction's causal structure. Moreover, this change in causality may propagate, i.e. the causal assignment at adjacent elements may be forced to change as a result of this causal reassignment.

Example 3. Consider the scenario where the example circuit goes from the mode with both switches on (HBG shown in Figure 5(a)), to the mode where junction 1_a (corresponding to switch Sw_1) has turned off (HBG shown in Figure 6(a)). The gray color of the 1_a junction depicts that it is off. As a result of junction 1_a switching off, the determining bond of junction 0_b is deactivated, and the causality of the HBG in this new mode needs to be reassigned, as junction 0_b needs a new determining bond assignment. Since bond 3 is connected to L_1, bond 4 has to be assigned as the determining bond of junction 0_b. As a result, bond 4 also becomes the determining bond of junction 1_c, and bond 9 is assigned as the determining bond for junctions 0_d and 1_e. The reconfigured BD is shown in Figure 6(b).

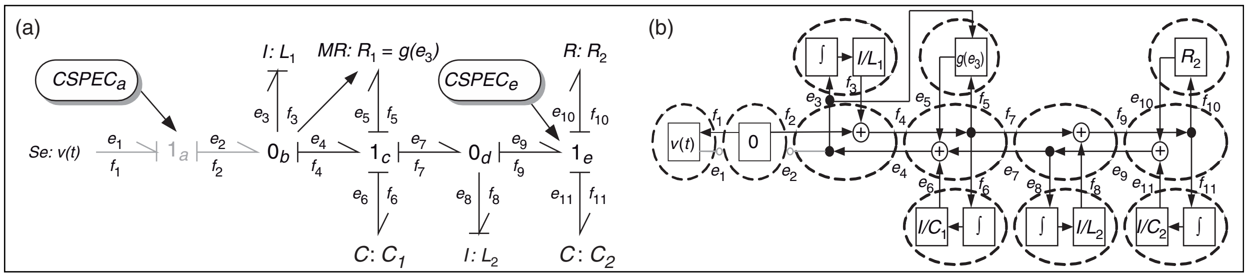


Figure 6. Example system with junction 1_a off: (a) hybrid bond graph; (b) block diagram.

The implication of the BD reconfiguration for HBGs (unlike BGs, which are designed for modeling continuous behavior) is that the system behavior is governed by different sets of equations in different modes of operation. In our approach, we use reconfigurable BDs to capture the different sets of equations. The causality assignments (and reassignments) form the basis for computing the reconfigured BD structures. To make the reconfiguration process efficient, we develop an efficient causal reassignment algorithm.

3.3. Reassignment of causality

Since mode changes necessitate the reassignment of causality, we require a causal reassignment procedure. The naive approach to causal reassignment is to run SCAP on the *entire* HBG *every* time a mode change occurs, and build the BD model for the new mode of the system, based on the causal assignment of the new mode. However, this approach is inefficient for the following reasons: (i) typically, only a small part of the HBG needs to be reassigned causality, and as a result, only a small part of the BD needs to change from what it was in the previous mode; and (ii) often, changes in causality do not propagate and the effect of a mode change produces only local changes in the computational structure of a junction. Hence, instead of building a new BD model every time a mode change occurs, we need only enumerate the *possible* BD representations of each HBG element, and include mechanisms in each element's BD to reconfigure online to the computational structure corresponding to its active causal assignment.

Example 4. If the example circuit goes from the mode shown in Figure 5(a) to that shown in Figure 7(a), where junction 1_e is switched off (instead of junction 1_a switching off, as was shown in Example 3), bond 7 must be the determining bond for junction 0_d in this new mode and the determining bond for junction 1_a remains unchanged. The BD model for this system mode is shown in Figure 7(b). Note that the only change in causality when transitioning from Figure 5(a) to Figure 7(a) is to bond 10 incident on junction 1_e. Since the causality of bond 9 that connects junction 0_d to 1_e remains unchanged, the causality change does not propagate to the rest of the HBG.

By comparing the BDs in the three different modes of the circuit shown in Figures 5–7, we observe that when a mode change occurs, the internal computational structure of HBG elements may change depending on the causal assignment to the corresponding bond. Typically, the computational structures of the BDs for some HBG elements remain invariant across all system modes. We exploit this observation to develop a general model transformation and simulation scheme for HBGs.

In our work, we require that the modeler ensures that for every physically meaningful mode, the HBG can be assigned a valid (integral) causality. As part of future work, we intend to relax this requirement, and adapt efficient methods, such as that presented by Hofbaur and Wotawa,[16] for automatically verifying that a valid causality assignment exists for each mode of system operation. The causal assignment in each mode of the system can also be checked automatically using a brute-force approach that goes through each mode and searches for a valid causality assignment. However, this approach can be computationally expensive, since the number of possible modes is exponential in the number of switching junctions.

In the following, we formalize the concepts and procedures outlined in this section. We first develop a procedure for automatic translation of HBGs into reconfigurable BD models, and then present an incremental approach for reassigning causality in HBGs when mode changes occur, followed by a discussion of the steps involved in simulating the reconfigurable BD models.

4. Hybrid bond graphs to reconfigurable block diagram models

As discussed, the BD models for each HBG element must include mechanisms for reconfiguring their

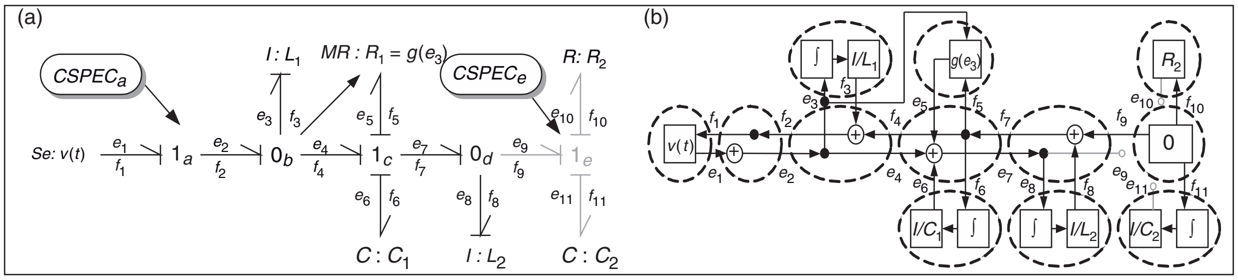


Figure 7. Example system with junction 1_e off: (a) hybrid bond graph; (b) block diagram.

computational structure to the new mode, depending on the new causal assignment. Not all possible causal assignments for individual components and junctions may apply for a particular system. By determining beforehand the set of valid causal assignments for individual elements, we can reduce the number of configurations needed for each BD element. When a single configuration is valid for a BD element in all modes of system operation, causal reassignment, and, therefore, BD structure reconfiguration, does not apply to this element. Taking this into account, savings in space and time can be achieved by facilitating the building of space-efficient BD models and the time-efficient causal reassignment of system HBG. We define this notion of invariance of causal assignment for all system modes as *fixed causality*.

Definition 2 (Fixed causality). A bond is in *fixed causality* if the causal assignment for that bond remains the same for all system modes. An HBG element is in *fixed causality* if all of its incident bonds are in *fixed causality*.

The BDs for *Se* and *Sf* elements are fixed to their only possible configuration. Under the integral causality assumption, the BDs for *C* and *I* elements are also fixed to their only possible configuration. When *R*, *TF*, and *GY* elements are in fixed causality, their BDs are fixed to one of their two possible configurations shown in Table 1. A junction element is in fixed causality if all of its bonds are in fixed causality, and since the determining bond of a junction sets the causality of all other incident bonds, all of its incident bonds must be in fixed causality if the determining bond is in fixed causality. For a non-switching junction with m incident bonds, its BD may have at most m possible configurations (see Figure 8). If the junction is in fixed causality, its BD structure is fixed to one of the m possible configurations.

Example 5. In the three modes shown in Figures 5–7, the *Se*, *C*, and *I* elements (and their incident bonds) are in fixed causality, and their corresponding BDs are invariant for all modes of operation. The other elements are not in fixed causality as their causal assignment may change from one mode to another. Therefore, their corresponding BDs may reconfigure when mode changes occur.

Even if a junction with m incident bonds is not in fixed causality, we can still simplify its BD model if the bonds that can never be its determining bond are in fixed causality, thereby *constraining* the set of determining bonds for this junction. In this case, we say the junction is in *constrained causality*, and we only need to instantiate a subset of the m possible causal configurations, thereby simplifying the junction's BD and the possible reconfiguration actions.

Definition 3 (Constrained causality). A junction with $m > 2$ bonds is in *constrained causality* if at least one and at most $m - 2$ of its incident bonds are in fixed causality, and none of these bonds are its determining bond, i.e. the junction has 2 to $m - 1$ possible causal assignments.

Example 6. In Figure 5(a), the BD model for junction 0_b does not need to include any reconfiguration mechanism to accommodate the possibility of bond 3 being its determining bond, because L_1, under the integral causality assumption, will impose flow at junction 0_b in all modes. The switching junction 1_e is also in constrained causality, because bond 11 can never be its determining bond, being connected to C_2. Hence, when on, the subset of possible determining bonds for junction 1_e is restricted to bonds 9 and 10 across all modes. Therefore, the possible causal configurations for junction 1_e are constrained to 3 from the maximum possible 4.

We can identify the situations where a given bond must be in fixed causality, and then propagate the constraints imposed by that causal assignment throughout the HBG to identify additional situations of fixed causality using a SCAP-like algorithm. There are four cases that result in a bond having fixed causality (see Figure 9)

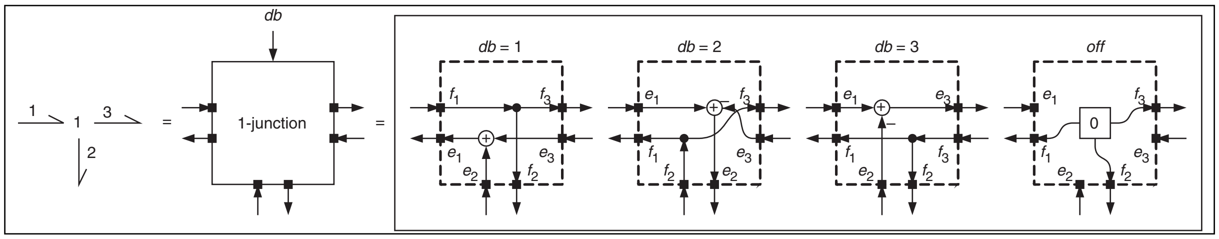


Figure 8. Block diagram expansion of a switching 1-junction (*db* denotes determining bond).

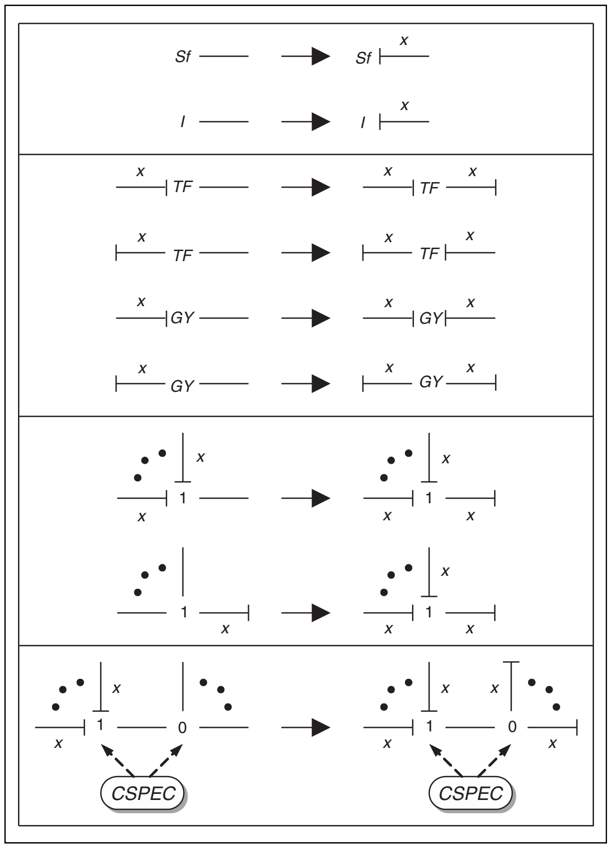


Figure 9. Propagation of fixed causality. The *x* symbol denotes a bond in fixed causality. The dual cases are not shown.

based on the possible causal assignments of HBGs discussed in Section 2:

1. Any bond incident on a energy source or storage element can only have one valid causal assignment, and, therefore, must be in fixed causality for all possible modes. The causal constraints of source elements and the integral causality assumption fix these assignments.
2. Any bond incident on a *TF* or *GY* element is in fixed causality if the other bond incident on the *TF* or *GY* element is also in fixed causality.
3. For a non-switching junction, if its determining bond is in fixed causality, then all other bonds incident on this junction must be in fixed causality. Otherwise, if all bonds except one are in fixed causality and none of these are the determining bond, then the remaining bond must be the determining bond and in fixed causality. The causal constraints of junctions fix these assignments, since a junction must have exactly one determining bond.
4. For two switching junctions of different type that are adjacent and share the same CSPEC, if the non-shared bonds for one of the junctions are in fixed causality, such that the shared bond is forced to be the determining bond when the junction is on, then the non-shared bonds for the other junction must be in fixed causality because the shared bond must be the determining bond of the other junction when on. There may be more complex cases involving shared CSPECS that affect fixed causality, but which are not physically meaningful. We disregard such cases.

Example 7. Consider the case given in the last cell of Figure 9. The non-shared bonds for the 1-junction have been determined to be in fixed causality. Therefore, when the junction is on, the shared bond will be the determining bond and will impose effort on the 0-junction. Hence, when the 0-junction is on, its remaining bonds must impose effort on the adjacent elements. When the 0-junction is off, these bonds must still impose effort, therefore, they are in fixed causality. The shared bond cannot be marked as fixed, because its causality will change depending on whether the junctions are on or off.

Note that fixed causality can never be independently assigned to an *R* element. An *R* element can be assigned fixed causality only if its adjacent junction is in fixed causality. Therefore, these four cases (illustrated in Figure 9) form a complete set of situations in which a bond can be in fixed causality. Hence, we can derive a SCAP-like procedure that recognizes these cases to determine the bonds (and, hence, HBG elements) that are in fixed causality. Since the individual cases are provably correct as shown by our analysis, the resulting algorithm, by construction, correctly assigns *fixed* labels to bonds. This algorithm, called `DetermineFixedCausality`, is shown as Algorithm 1. In this algorithm, a bond is denoted as $b = (k, l)$, where k and l denote the two HBG elements the bond b connects (note that bond directions are not implied by the order of the elements, k and l). Like `SCAP`, we start from energy source and storage elements, because their bonds must be in fixed causality (see case 1 above). The algorithm assigns causality to the bonds and labels them to be in fixed causality. The effects of this assignment are then propagated to adjacent elements. The function `PropagateFixedCausality` handles the propagation. It takes as input a junction j. If j is a non-switching junction, any incident bond in fixed causality is labeled so, and propagation continues using `PropagateFixedCausality` on all junctions connected to j directly, or through a TF or GY element, by bonds in fixed causality. If j is a switching junction, `PropagateFixedCausality` checks whether case 4 described above is satisfied for this junction before any bonds incident on it is labeled to be in fixed causality, and `PropagateFixedCausality` is called on junctions connected to j, either directly, or through a TF or GY element. When the algorithm terminates, the remaining unassigned bonds cannot be in fixed causality.

Essentially, this algorithm only differs from `SCAP` on two points: (i) it must reason about junctions turning on and off (see case 4 above); and (ii) it defers from making arbitrary causal assignments. Like `SCAP`, the worst-case computational complexity of this algorithm is linear in the size of the HBG.[5] On average, the complexity will be better than `SCAP`, because if parts of the HBG are in fixed causality, the entire HBG will not be traversed.

Algorithm 1 `DetermineFixedCausality`(HBG)

```
while all bonds incident on an energy source or storage element $k$ are not labeled as *fixed* do
    Pick a bond $b = (k, j)$, where $j$ is a junction adjacent to $k$, which is not labeled as *fixed*
    Label bond $b$ as *fixed* and assign its causality
    PropagateFixedCausality($j$)
```

Function 2 `PropagateFixedCausality`(j)

```
if $j$ is a non-switching junction then
    if $j$ can be assigned a unique determining bond then
        for all bonds $b = (j, k)$ do
            if $b$ is not labeled as *fixed* then
                Label $b$ as *fixed* and assign its causality
                if $k$ is a junction then
                    PropagateFixedCausality($k$)
                else if $k$ is a $TF$ or $GY$ element then
                    Identify junction $j'$ connected to $k$, label bond $b' = (k, j')$ as *fixed*, and assign its causality
                    PropagateFixedCausality($j'$)
else if $j$ is a switching junction then
    if $j$ is adjacent to a switching junction $j'$ of different type with the same CSPEC, and when both on, the bond $(j, j')$ must be the determining bond then
        for all $b = (j', k)$ where $k \neq j$ and $b$ is not labeled as *fixed* do
            Label $b$ as *fixed* and assign its causality with $(j, j')$ as the determining bond
            if $k$ is a junction then
                PropagateFixedCausality($k$)
            else if $k$ is a $TF$ or $GY$ element then
                Identify junction $j''$ connected to $k$, label bond $b'' = (k, j'')$ as *fixed*, and assign its causality
                PropagateFixedCausality($j''$)
```

In addition to the above four cases, HBG fragments which are not reachable from any switching junctions can be assigned fixed causality. For these bonds and elements, the causal assignment will remain unchanged from the initial causal assignment over all system modes, since the effects of junction switchings can never reach them. Effectively, these HBG elements are in fixed causality. However, `DetermineFixedCausality` does not recognize these cases. Therefore, to assign a fixed causality to the HBG elements that are unreachable from switching junctions, we run our causal assignment procedure, `Hybrid SCAP`, on the initial mode of the HBG to determine a consistent set of causal assignments for the unreachable HBG elements, and label their causal assignments as fixed. The initial causal assignment obtained by running `Hybrid SCAP` is essentially the same as what would be obtained by running `SCAP`, the main difference being that in our implementation, certain causal assignments (i.e. fixed causal assignments) are already made when it is invoked. If we are given an HBG with no switching junctions, i.e. essentially a BG, all of the elements will be assigned a fixed causality by default. `Hybrid SCAP` is explained in detail in the following section.

Based on the identification of bonds in fixed causality, we instantiate the BD model for each HBG element only with configurations for possible causal assignments. This algorithm, called `DeriveBlockDiagrams`, is shown in Algorithm 3. If all bonds incident on an element, k, are in fixed causality, then k is in fixed causality, and we use a fixed BD model for k. On the other hand, if k is a junction in constrained causality, with m incident bonds, its reconfigurable BD model will include mechanisms that can only switch between the possible causal assignments, as shown in Figure 8 for the general case. For constrained causality, only valid configurations are included. Finally, if k is neither in fixed nor constrained causality, its reconfigurable BD model includes mechanisms that can switch between all of the causal assignments. Once each BD element is instantiated, they are connected. The algorithm for deriving BDs from HBGs is presented as Algorithm 3.

5. Efficient causal reassignment in hybrid bond graphs

Once all fixed causality assignments are completed using the offline procedure described in Section 4, and the reconfigurable BD is constructed, an online procedure is needed to determine the online reconfigurations of the BDs when mode changes occur. To maximize the efficiency of this reconfiguration procedure, our causal reassignment method, `Hybrid SCAP`, takes advantage of the fact that only small parts of the HBG usually change causality from one mode to another. It begins with the causal assignment in the previous mode, and reassigns causality incrementally, starting from the junctions directly affected by the switching, and then propagating the changes only to those elements whose causal assignments are affected by the change.

When a junction changes mode and its causal assignment changes, this change may propagate to other parts of the HBG. When a junction turns on, it must be assigned a determining bond, since an off junction does not have a determining bond. Conversely, an on 1- (or 0-) junction imposes effort (or flow) through its determining bond, and when this 1- (or 0-) junction turns off, it imposes 0 flow (or effort) through its former determining bond (see Figure 1). If the bond connecting this junction, j_1, to an adjacent junction, j_2, is the determining bond of j_1, the turning off of j_1 will create a causal conflict with the causal assignment of j_2. That is, if j_1 and j_2 are of the same type, j_2 will have an extra bond with the same causal assignment as its determining bond, and if of the opposite type, j_2 must be assigned a new determining bond. The causal assignment of j_2 will have to be updated, as an on junction must have a single determining bond in any valid system mode. Whether an extra determining bond must be removed or a new one assigned, the change may propagate further if the bond that changes causality is adjacent to another junction.

These causal conflicts can be easily resolved if the junction in question has an adjacent R element. Since an R element can be in one of two causal configurations, as shown in Table 1, it can absorb the causal change, requiring no further propagation. However, if

Algorithm 3 `DeriveBlockDiagrams(`HBG`)`

```
DetermineFixedCausality($HBG$)
HybridSCAP($J$), where $J$ is the set of all junctions in the HBG that are not
in fixed causality
Fix the causal assignments of all bonds unreachable from switching junctions
for all elements $k \in HBG$ do
    Implement reconfigurable BD model for $k$ for valid causal configurations
    only
Connect the BDs for all HBG elements
```

multiple junctions switch simultaneously, then this choice should not be made immediately, because the change may prove to be inconsistent when propagating the changes due to another switching junction, requiring backtracking. That is, for a given mode of the system, certain causal assignments are *forced* into a particular type of causality. We must determine first which assignments are forced before arbitrarily resolving any causal conflicts due to a junction changing mode. A junction is assigned a forced causality if its causality is unequivocally assigned for a given mode. Consequently, all fixed junctions are necessarily forced junctions.

Definition 4 (Forced causality). For a given mode, a junction is in *forced causality* if it can be assigned only one possible determining bond in that mode.

Example 8. In Figure 5(a), junction 1_a and 0_b are in forced causality. The determining bonds of all other junctions depend on an arbitrary choice of causality assignment to the R elements, and so, they are not in forced causality. In Figures 7(a) and 7(b), all active junctions are in forced causality as there is only one consistent assignment of determining bonds for all active junctions.

Unlike SCAP, Hybrid SCAP maintains state, i.e. it remembers all causal assignments of the previous mode, including which of these were in fixed and forced causality. This state is initialized during the first call to Hybrid SCAP, in which it assigns causality to the initial mode of the system (in DeriveBlockDiagrams). By using this state, Hybrid SCAP avoids making causal assignments that could turn out to be inconsistent with the causal assignment of other junctions that have changed mode. Hybrid SCAP, given as Algorithm 4, takes as input the set of junctions, *J*, which need to be assigned causality (for its initial call) or reassigned causality (i.e. junctions that have just changed mode). This method incrementally changes the causal assignments from the previous mode to be consistent with the new mode. In order to avoid backtracking, and thus improve efficiency, we divide the process into three loops.

In the first loop, the algorithm makes causal assignments, and labels junctions to be in forced causality, based only on the bonds that are in fixed causality, and the junctions that are in forced causality and have been *assigned* a causality in the *new* mode. If multiple possible determining bond assignments are valid for a junction, the junction is added to a second set of junctions, J'. Propagations from other junctions may indirectly resolve the choice of the determining bond in the first loop. Otherwise, the decision to assign the junction's determining bond is deferred until the second loop. By deferring these decisions until all forced changes have been made, we ensure that inconsistent choices cannot be made within this loop.

In the second loop, we make decisions not only based on what is known to be true for the new mode (i.e. fixed junctions and those marked as *assigned*), but also on what was known to be true for the previous mode. At this point, we know that any forced junctions which have not been visited must be forced in the new mode into the same causality, otherwise, we would have propagated to them in the first loop. Therefore, we can consider their causality to be forced for the new mode as well. Arbitrary decisions are deferred until the third loop by placing the junctions into a third set, J''.

After the first two loops, we guarantee by construction that all forced assignments have been made. All remaining junctions are *unforced*, and have multiple valid causal assignments. To resolve remaining causal conflicts, we arbitrarily pick an unforced junction from J'', and, if it has an adjacent R element, assign the connecting bond to be the junction's determining bond, and propagate the effects of this causal assignment by adding this junction to J''. If the junction does not have an adjacent R element, we choose an incident bond connecting this junction to an unforced junction as this junction's determining bond, and propagate the effects of this causal assignment throughout the HBG. This procedure is repeated until J'' is empty.

In the first loop of Hybrid SCAP, we guarantee that no incorrect causal assignments are made, since we reassign causality based on the fixed causal assignments (which are true in all modes, including the new mode) and causal assignments based on those (which also must be true in the new mode) only. We cannot, in general, assume what was in forced causality in a previous mode is in forced into the same causality in this mode. So, that information is not used in the first loop. After the first loop, we are guaranteed that any junctions that were not reassigned causality in the first loop, but are marked as forced from the previous mode, must be forced into the same causality in the new mode, otherwise their causality would have been reassigned in the first loop. Therefore, in the second loop, we can use this information to make new assignments that are unique. Finally, in the third loop, multiple possible causal assignments exist for the remaining HBG elements (otherwise, the assignment would have been made already). Therefore, as long as we choose one of these multiple causal assignments and propagate the effects of this assignment correctly, Hybrid SCAP is correct by construction.

Example 9. Consider the scenario where junctions 1_a and 1_e simultaneously change mode, causing the

Algorithm 4 HybridSCAP(J)

```
Initialize $J'$ to $\emptyset$
while $J \neq \emptyset$ do
  Remove a junction $j \in J$
  if $j$ is not assigned then
    if $j$ is off then
      Assign the causality of $j$ as off and label $j$ as forced and assigned
      if $j$'s previous determining bond, $(j, k)$, connects $j$ to a junction $k$ then
        Add $k$ to $J$ if $k$ is not labeled as assigned
      else if $j$'s previous determining bond, $(j, k)$, connects $j$ to a $TF$ or $GY$, $k$ then
        Add junction $j^*$ connected to $k$ through bond $(j^*, k)$ to $J$ if $j^*$ is not labeled as assigned
    else if $j$ is on then
      if $j$ can be assigned a unique determining bond $db = (j, k)$, based on fixed and forced, assigned junctions then
        Assign $db$ as the determining bond of $j$ and label $j$ as forced and assigned
        if $j$'s previous determining bond, $(j, k)$, connects $j$ to a junction $k$ then
          Add $k$ to $J$ if $k$ is not labeled as assigned
        else if $j$'s previous determining bond, $(j, k)$, connects $j$ to a $TF$ or $GY$, $k$ then
          Add junction $j^*$ connected to $k$ through bond $(j^*, k)$ to $J$ if $j^*$ is not labeled as assigned
    else
      Add $j$ to $J'$ if $j$ is not labeled as assigned
while $J' \neq \emptyset$ do
  Remove a junction $j \in J'$
  if $j$ is not assigned then
    if $j$ can be assigned a unique determining bond $db = (j, k)$, based on fixed and forced junctions then
      Assign $db$ as the determining bond of $j$ and label $j$ as forced and assigned
      if $j$'s previous determining bond, $(j, k)$, connects $j$ to a junction $k$ then
        Add $k$ to $J$ if $k$ is not labeled as assigned
      else if $j$'s previous determining bond, $(j, k)$, connects $j$ to a $TF$ or $GY$, $k$ then
        Add junction $j^*$ connected to $k$ through bond $(j^*, k)$ to $J$ if $j^*$ is not labeled as assigned
    else
      Add $j'$ to $J''$ if $j'$ is not labeled as assigned
while $J'' \neq \emptyset$ do
  Remove a junction $j \in J''$
  if $j$ is not assigned then
    if $R$ element is adjacent to $j$ then
      Assign bond to adjacent $R$ element as the determining bond of $j$ and label $j$ as unforced and assigned
    else
      Arbitrarily assign the bond to an adjacent unforced junction $j'$ as the determining bond, either directly, or through a $TF$ or $GY$
      Label $j$ as unforced and assigned
      Add $j'$ to $J''$ if $j'$ is not labeled as assigned
```

example circuit to transition from the mode shown in Figure 6(a), where junction 1_a is off and junction 1_e is on, to that shown in Figure 7(a), where junction 1_a is on and junction 1_e is off. To reassign causality of the HBG for the mode shown in Figure 7(a), `Hybrid SCAP` is invoked with junctions 1_e and 1_a, in its input queue, J. `Hybrid SCAP` removes junction 1_e from J, assigns the causality of 1_e as off, labels 1_e as *forced* and *assigned*, and then adds junction 0_d to J as 0_d is connected to 1_e through 1_e's determining bond, when 1_e was on. Then, junction 1_a is removed from J, and assigned a forced causality, as the *Se* connected to 1_a through bond 1 forces bond 2 to be junction 1_a's determining bond. Once bond 2 is assigned as junction 1_a's determining bond, 1_a is labeled as *forced* and *assigned*, and junction 0_b that is connected to 1_a by 1_a's determining bond is added to queue J. Next, junction 0_d is removed from J, and bond 7 is assigned the determining bond of 0_d, since junction 1_e is labeled *forced* and *assigned*, and bond 8, being connected to inductor L_2, has fixed causality. So junction 0_d is labeled *forced* and *assigned*, and junction 1_c is added to queue J, since it is connected to 0_d by 0_d's determining bond. `Hybrid SCAP` then removes junction 0_b from J and assigns bond 2 as its determining bond, and labels junction 0_b as *forced* and *assigned*. Even though junction 1_a is connected to junction 0_b by 0_b's determining bond, it, being already labeled as *assigned*, is not added to queue J. Finally, junction 1_c is removed from J, and based on the causal assignment of its neighboring *forced* and *assigned* junctions, is assigned bond 7 as its determining bond, and is labeled as *forced* and *assigned*. Junction 0_d, being already marked as *assigned*, is not added to J, even though it is connected to junction 1_c by 1_c's determining bond. Since J is now empty, and so are queues J' and J'' (since the causality of all HBG elements are forced by fixed elements in the new mode), `Hybrid SCAP` terminates.

The worst-case complexity of `Hybrid SCAP` is the same as that of `SCAP`. However, on average, we do much better than `SCAP` because the causality of only a few HBG elements needs to be reassigned as the system transitions from one mode to another. This was shown earlier using the example circuit, and the experimental results presented in Section 8.3 show this to be true for our case study, as well.

6. Simulating the block diagrams

In the previous sections, we showed how to build reconfigurable BDs from HBGs, and how to reassign causality efficiently to HBGs as mode change occurs. In this section, we describe how to use the assigned causality to reconfigure the BD, and how to simulate these reconfigurable BDs efficiently. The simulation involves two recurring steps: (i) reconfiguring the BD to the correct computational structure based on the causal assignment in a new mode, and (ii) simulating the BD configuration until another mode change occurs.

Every time a mode change occurs, the generation of the correct BD configuration for the new mode involves invoking the `Hybrid SCAP` algorithm to assign causality for the new system mode, reconfiguring the BD model according to this assignment, and resuming the simulation until the next mode change occurs. However, using the knowledge of fixed causality, we can increase simulation efficiency by avoiding calls to `Hybrid SCAP`, if a switching junction that changes its mode satisfies the following properties: (i) for all system modes in which the junction is on, it is always assigned the same determining bond, and (ii) when the junction turns off, the causality change does not affect the determining bond assignments of its neighboring junctions. If these conditions are satisfied, the BD may be reconfigured based directly on the junction mode, without invoking `Hybrid SCAP`, thus avoiding its overhead. Essentially, we pre-compile what the result of running `Hybrid SCAP` would be into our reconfigurable BD structure by implementing its reconfiguration as a function of the junction mode. A switching junction will never require a call to `Hybrid SCAP` in the following situations, which can be easily identified through an offline analysis of the HBG:

1. The determining bond assigned to this switching junction in its on state is always the same, and this determining bond is connected to an R element. The R element does change causality when the junction turns off, but since the input to the BD for the R is zero, we get a zero output from the BD, irrespective of whether the R element is implemented as $e = Rf$ or $e = f/R$, and therefore, we do not need to reconfigure the R element.
2. A switching junction is connected to another switching junction, of different type, and they share the same CSPEC, such that the shared bond is the determining bond for both junctions, when both are on. Recall this scenario is the same as that described in Section 4 which results in propagation of fixed causality labels for bonds across switching junctions (see Figure 9). In this situation, when the adjacent switching junctions change mode together, only the causal assignment of the shared bond changes, and no propagation of causal changes occurs.

The BD model of any such junction is implemented so that it can switch between only two configurations: (i) the fixed configuration corresponding to the invariant determining bond assignment whenever this junction is on; and (ii) the configuration corresponding to the junction's off mode. The BD model of such a junction can switch between these two configurations depending on the CSPEC state alone.

Algorithm 5 presents our overall approach for simulating the BD models. Once the reconfigurable BD model is generated, simulation starts with the BD structure corresponding to the initial mode, and the simulation continues until a mode change occurs. If the conditions to avoid calling `Hybrid SCAP`, as explained above, are not satisfied, the simulation is paused, the `Hybrid SCAP` algorithm is invoked to reassign causality, and the BD is reconfigured accordingly before the simulation resumes. Otherwise, the BD is reconfigured based on the junction mode, and the overhead of calling `Hybrid SCAP` is avoided.

In our approach, we leverage the solvers of existing simulation applications to generate the correct solution of the underlying mathematical model of hybrid systems produced by component-based modeling. These models can be either a set of ODEs, or DAEs that may include algebraic loops. Algebraic loops exist when the value of a system variable is algebraically dependent on its own value. There are three cases that result in algebraic loops in the BD structure: (i) just as in BGs, [5] the BD always has an algebraic loop if the HBG model of the BD does not have a unique causal assignment, e.g., bonds 5, 7, 9, and 10 in Figure 5(a) do not have unique assignments; (ii) the modulating function may sometimes create causal algebraic loops depending on element's causality, e.g., in Figure 6(b), the following algebraic loop exists $e_5 \rightarrow e_4 \rightarrow e_3 \rightarrow g(e_3) \rightarrow e_5$ when the modulated R element imposes effort on its adjacent junction; and (iii) the CSPEC function may also introduce algebraic loops based on the variables involved and the HBG causality, e.g., in the example circuit shown in Figure 3(b), the on- and off-guard of the CSPEC$_e$ for junction 1_e (representing the circuit breaker, Sw_2) is a function of flow f_9, producing an algebraic loop. Generating fixed-point solutions for DAEs with algebraic loops becomes computationally expensive when the fixed-point method has to iterate to converge to a solution.

7. A tool suite for the modeling and transformation of hybrid bond graphs for simulation

We have implemented our modeling and simulation approach as the MoTHS tool suite. Recall that the MoTHS tool suite consists of a graphical modeling language for building hierarchical, component-based HBG models[6] in the GME and a set of model translators, or *interpreters*, that automatically convert these HBGs into BD-based simulation models for selected simulation tools. Initially, we have developed an interpreter that creates MATLAB Simulink models.

7.1. Constructing hybrid bond graph models

In our simulation framework, we preserve the component-based hierarchical structure inherent in present-day engineered systems by using component-based models. From the perspective of modeling, a *system* is a collection of interconnected *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. The component models represent physical processes with hybrid behaviors, and in our HBG framework, the internal structure of a component model is defined by an HBG fragment with a corresponding set of functions to define the CSPECs of the switching junctions. Details of the modeling paradigm are presented in Manders et al.[6] We review the main features in the following.

The component interfaces are defined by (i) energy ports for energy transfer, and (ii) signal ports for information transfer. Energy ports are connected using bonds, and signal ports are connected using signal links. Ports are denoted by the ■ symbol in the HBG component models (e.g. see Figures 16, 19, and 20). Energy transfer between two component models is denoted by a bond drawn from an energy port in one component to an energy port in another (see Figure 21). Signal transfer between two components is denoted by a directed link drawn from one signal port to another.

Algorithm 5 `Simulate`(BD)

```
do simulate $BD$
  until a set of junctions $J$ switch mode
    if the mode switch of each junction does not require any propagation of causality changes then
      Reconfigure BD based on junction mode
    else
      Reassign causality for the new mode by calling HybridSCAP($J$)
      Reconfigure BD based on the new causality assignment
```

CSPECs are formulated as simplified two-state finite automata, with one state mapping to the on mode and the other to the off mode of the junction. As such, the modeler needs only to specify the transition guards going from the on to off mode (the *off-guard*) and the off to on mode (the *on-guard*). These functions can be specified using effort, flow, and signal variables within the model, so we can capture both controlled and autonomous mode transitions. Note that the two-state formulation is not complete in its representation of hybrid behaviors, but is sufficient for many realistic physical systems. Extending the formulation to the general *n*-state framework will be addressed in future implementations.

Figure 11 shows how the circuit HBG shown in Figure 3(b) is drawn using the GME visual editor. The largest pane is the main editor where the HBG is drawn. Immediately below this pane is the *part browser* that contains the different HBG elements that can be dragged and dropped in the main editor to compose the HBG model in GME. In addition to the standard HBG elements, the part browser also contains some MoTHS-specific elements, such as *DecisionFunctions*. DecisionFunctions (e.g. Sw2Func) use logical expressions to convert continuous measurement signals into Boolean values, which can then be used within on- or off-guards for switching junctions. The *browser pane* on the right of the editor pane provides a tree view of the different components and systems. In this example, it shows the three components, `Circuit`, `Circuit2`, and `Circuit3`. The *attribute panel* can be found below the navigator pane, and besides the part browser. This panel provides an editor to view and modify the attributes of the HBG elements. In this example, the attribute panel shows the attributes of the switching junction 1_a, and indicates that the on-guard for the junction is the signal 'Sw1Input' and the off-guard is '!Sw1Input' (note that the attributes are expressed using the syntax of C). Hence, junction 1_a switches on when 'Sw1Input' evaluates to true, and it switches off when '!Sw1Input' evaluates to true. The bottom pane of the GME window is called the *console* and displays warning or error prompts and reports whether or not the translation of the HBG to a block diagram completed successfully. For example, in Figure 11, the console warns the user that the parameter value of resistor R2 is 0, and the function specification for modulating function R1ModFunc is empty.

7.2. Translating hybrid bond graphs to block diagrams

An interpreter, which operates on the HBG models created in GME, generates the simulation model automatically. The interpreter operates in two steps: (i) the transformation of the HBG model to an implementation-independent *intermediate* BD model (IBD), and (ii) the conversion of the IBD model to simulation artifacts implemented in a target simulation environment. Although we have initially chosen MATLAB Simulink as the target simulation environment, use of the IBD model facilitates easy development of interpreters for several different simulation tools (e.g. Ptolemy[10]), in which only the second step has to be rewritten. In Figure 11, the interpreter is invoked by clicking the icon that is enclosed within the black rectangle.

7.2.1. Generating intermediate block diagrams.

Given an HBG model, the first stage of the interpreter navigates the model hierarchy, mapping HBG elements to IBD elements. The IBD language consists of the primitives `blocks` and `ports`. Hierarchy in the modeling environment is supported by the `systems` construct, which contains `systems`, `blocks`, and `ports`. The hybrid behavior of junctions are captured through `stateMachines`, which model the CSPECs. Figure 10 shows the IBD model of the example circuit HBG shown in Figure 3(b).

Blocks describe the mapping of their inputs to their outputs through a *block specification*. For the purposes of representing an HBG as an IBD, block specifications describe what kind of BG element a block corresponds to, and gives parameter values of that element. For example, as shown in Figure 10, capacitor C_1's block specification is given as $C_1(10, 0.01)$, where 10 is the capacitance (expressed, say, in μF), and 0.01 is the initial value of the voltage across the capacitor (expressed, say, in V).

Since the IBD language represents signal flow and BGs represent energy flow, each bond is converted into effort and flow signals, and energy ports are converted into pairs of signal ports. Additional ports are also introduced to support the hierarchical, component-based structure of the model. At the IBD level, the variable passed along a signal connection is not specified to be an effort or a flow, because no causality assignments have been made. The transformation is purely structural.

7.2.2. Generating simulink models.

The second stage of the interpretation process involves generating the simulation artifacts from the IBD model. The IBD has all of the information required to generate the Simulink model in its entirety. To enable the assignment of causality, a flat HBG data structure is also constructed during the interpretation process for generating the IBD. This data structure is essentially a graph describing HBG elements and their bond connections. This graph contains the minimal information

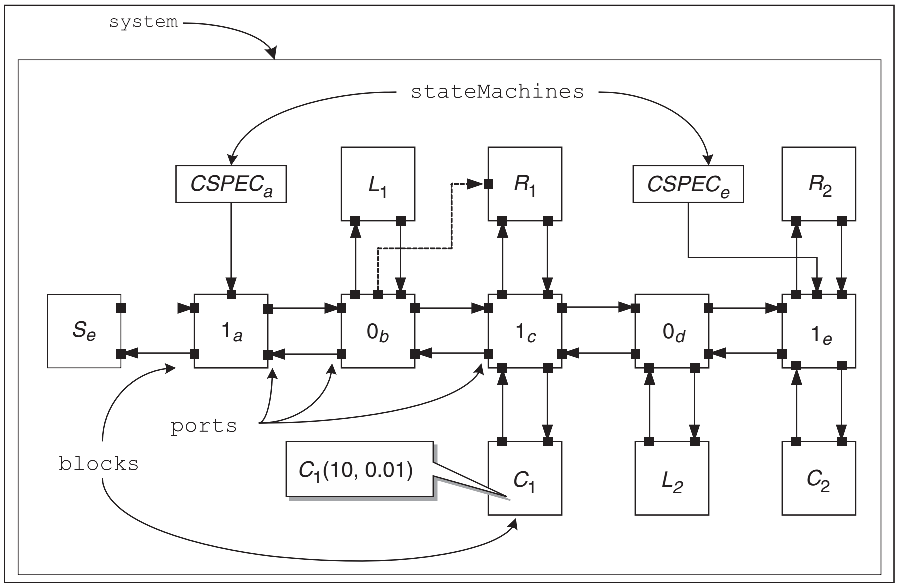
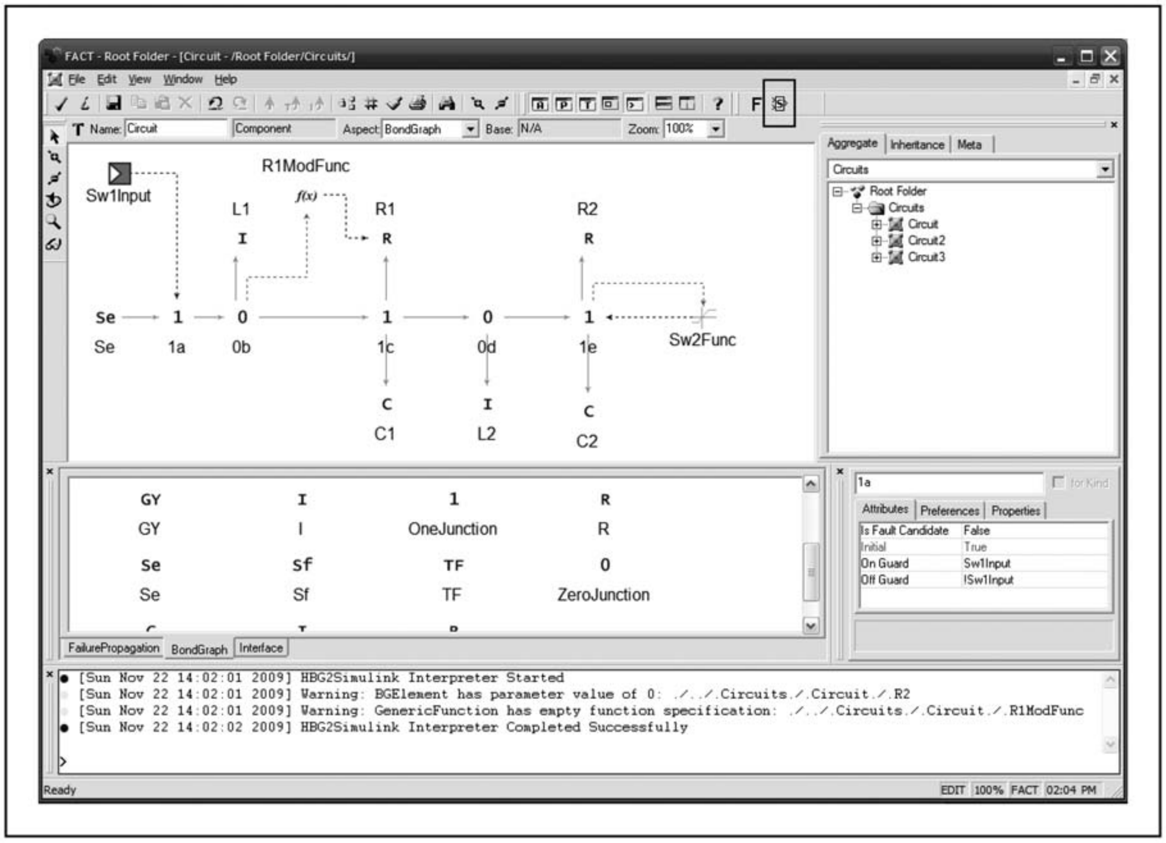


Figure 10. Intermediate block diagram model of an example circuit.



1a

Is Fault Candidate	False
Initial	True
On Guard	Sw1Input
Off Guard	!Sw1Input

```
● [Sun Nov 22 14:02:01 2009] HBG2Simulink Interpreter Started
  [Sun Nov 22 14:02:01 2009] Warning: BGElement has parameter value of 0: ./../.Circuits./.Circuit./.R2
  [Sun Nov 22 14:02:01 2009] Warning: GenericFunction has empty function specification: ./../.Circuits./.Circuit./.R1ModFunc
● [Sun Nov 22 14:02:02 2009] HBG2Simulink Interpreter Completed Successfully
>
```

Figure 11. Screenshot of the GME editor for MoTHS.

necessary to assign causality to the HBG, i.e. the element types, the junction modes, and bond connections.

First, we identify bonds in fixed causality (using an implementation of the `DetermineFixedCausality` algorithm), which in turn helps identify elements in fixed causality and junctions in constrained causality. Then, the initial causality assignment of the HBG is determined by running an implementation of `Hybrid`

SCAP. After the initial causality assignment of the HBG is determined, a MATLAB *build script* (also generated by the interpreter) is executed, which contains calls to functions in a library of MATLAB functions (M-code) for building the BD corresponding to each HBG element. The arguments to these functions include the initial causality assignment, whether or not the HBG element is in fixed or constrained causality, and the block specification of the HBG element.

When the build script is executed, a Simulink subsystem with ports and internal blocks is instantiated for every component in the BD model, as shown in Algorithm 3. Switching junctions whose causal reassignment can be implemented as a function of the junction mode are identified. For all other elements, reconfiguration is implemented using switch elements that take the corresponding determining bond as input. The Simulink implementation of this approach is depicted in Figure 12 for the junction of Figure 8. Zero-indexed multi-port switches select the correct functions of the inputs that determine the outputs depending on the determining bond. Here, $db = 0$ denotes an off junction. Because blocks cannot be connected or disconnected while a Simulink model is executing, we maintain static connections between blocks, where the *interpretation* of the signal along the connection changes. Here, $u(i)$ and $y(i)$ are the signals associated with bond i. For example, if $db = 1$, then $u(1) = f_1$, $u(2) = e_2$, $u(3) = e_3$, $y(1) = e_1$, $y(2) = f_2$, and $y(3) = f_3$. Under this interpretation, $y(1) = u(2) + u(3)$, $y(2) = u(1)$, and $y(3) = u(1)$, as depicted in the figure. Once the BD blocks are instantiated, these subsystems are connected using the signal connections in the BD model. Figure 13 shows the resultant Simulink BD of the circuit HBG shown in Figure 11. As can be seen, there is a one-to-one correspondence between the HBG components drawn in GME and the Simulink BD components.

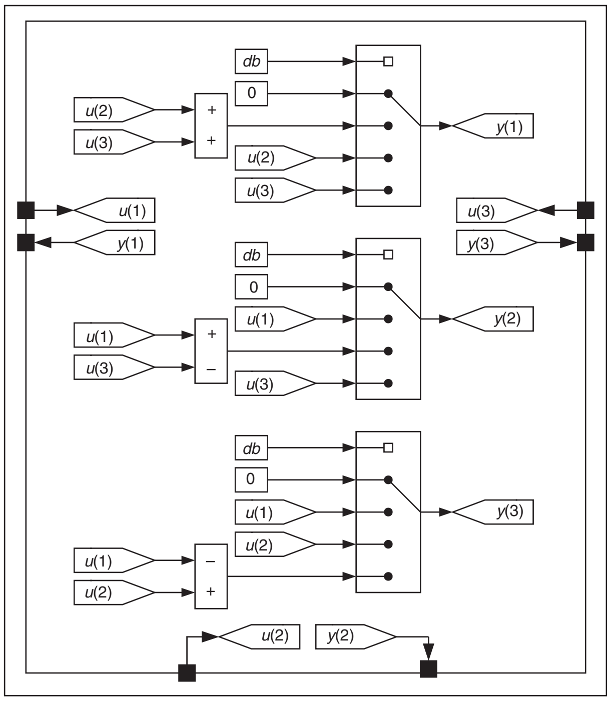


Figure 12. Simulink implementation of a switching 1-junction (*db* denotes determining bond).

7.2.3. Graphical user interface for injecting faults.

Simulation testbeds are very useful for running simulation experiments for diagnosis and prognosis applications, in which the ability to inject faults in these models is required. Our approach facilitates fault injection through a graphical user interface (GUI) that is constructed automatically during the interpretation process. As part of the modeling language, the user specifies which HBG elements can have faults. Each of the selected HBG elements is included in the GUI. Figure 14 shows the fault injection GUI for the Simulink model of Figure 13.

The GUI allows the user to select the time of fault occurrence (we consider only persistent faults), the fault profile, and the fault magnitude (where applicable) for each of the included HBG elements. Parametric faults are associated directly with HBG element parameter values (i.e. *Se*, *C*, *R*, etc.), and the available fault profiles include *abrupt*, *incipient*, and *stuck-at* faults. An abrupt fault is a fast change in a system parameter whose time constant is much smaller than that of the system dynamics, and is modeled as a constant bias term that is added to the nominal parameter value at the time of fault occurrence. An incipient fault is a slow change in system parameters, and modeled as a linear drift term (with constant slope) that is added to the actual parameter value. A stuck-at fault sets the parameter value of a modulated element to a constant value at fault occurrence. Discrete faults cause the system mode to change, so they affect the switching junctions. The fault profile includes *stuck-on* and *stuck-off* faults, introduced by adding fault terms to the CSPEC transition guards. As a special case, sensor faults can be modeled as either parametric or discrete. In our approach, we treat them as parametric faults, allowing abrupt (bias), incipient (drift), and stuck-at faults (which can also model discrete faults such as sensor failure). In addition, we allow the introduction of white Gaussian sensor noise, where the mean and variance can be specified. A change in variance can also be introduced as a sensor fault.

7.3. Executing the simulink models

To simulate the system, first, the `Hybrid SCAP` algorithm is executed on the HBG data structure to obtain an

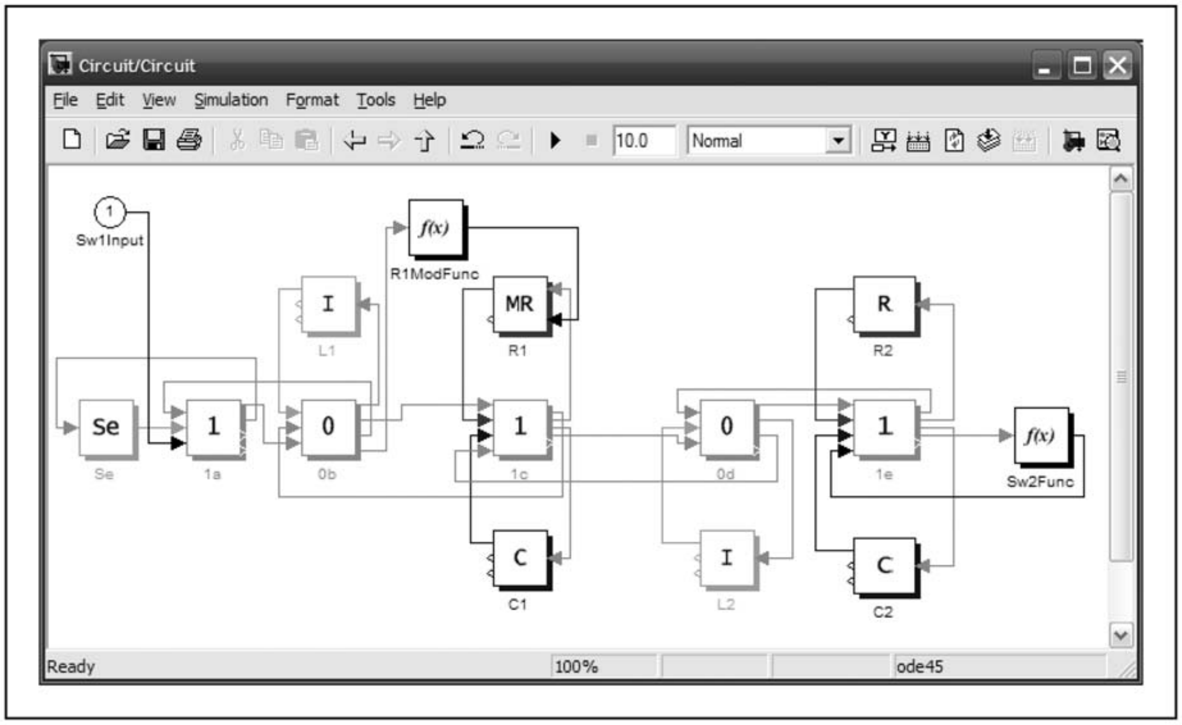
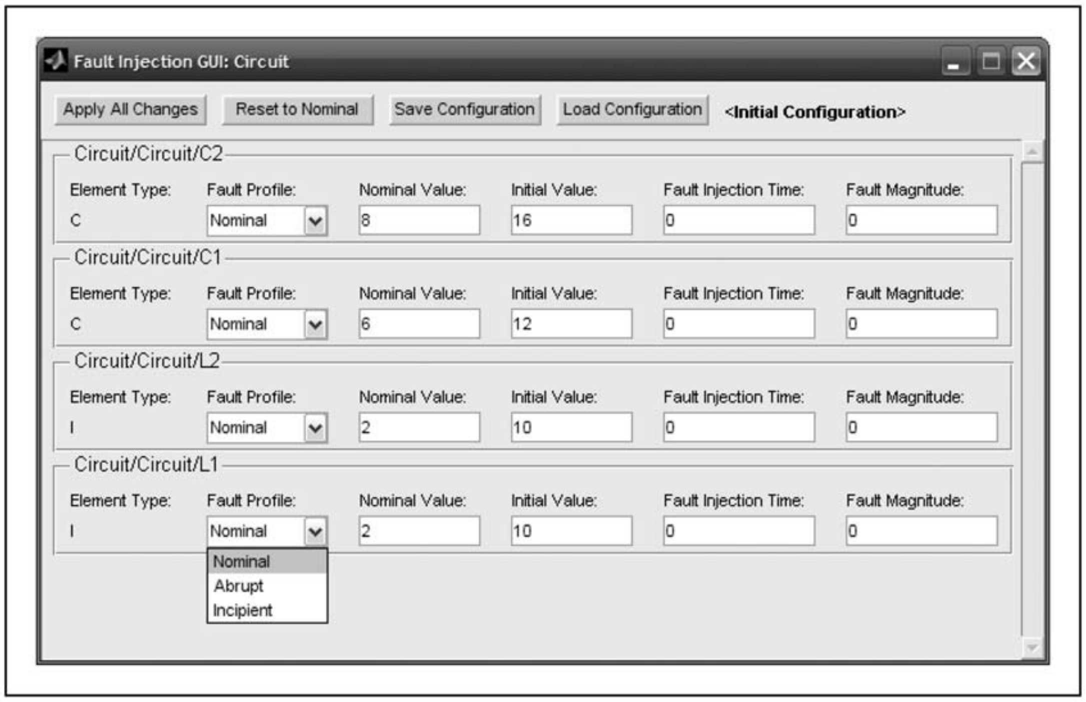


Figure 13. Screenshot of the Simulink model for the example circuit.



<Initial Configuration>

Circuit/Circuit/C2

Element Type:	Fault Profile:	Nominal Value:	Initial Value:	Fault Injection Time:	Fault Magnitude:
C	Nominal	8	16	0	0

Circuit/Circuit/C1

Element Type:	Fault Profile:	Nominal Value:	Initial Value:	Fault Injection Time:	Fault Magnitude:
C	Nominal	6	12	0	0

Circuit/Circuit/L2

Element Type:	Fault Profile:	Nominal Value:	Initial Value:	Fault Injection Time:	Fault Magnitude:
I	Nominal	2	10	0	0

Circuit/Circuit/L1

Element Type:	Fault Profile:	Nominal Value:	Initial Value:	Fault Injection Time:	Fault Magnitude:
I	Nominal	2	10	0	0

Figure 14. Screenshot of the fault injection graphical user interface for example circuit.

initial causality assignment. The causality information for the current mode is stored in a global array whose elements are the determining bonds of every junction in the HBG. As the determining bonds change, the associated switches are triggered to activate the new block diagram structure.

For switching junctions, we evaluate the *off-guard* and the *on-guard* to determine whether the junction is turning on or off, or remaining in its current mode. If the junction is off (or on) and the on-guard (or off-guard) evaluates to true, the junction switches mode. If the junction switches off, all outgoing signals are set to 0. Otherwise, the junction configuration is determined by the causality assigned to that junction, represented by its determining bond. When `Hybrid SCAP` must be called, the simulation is paused, `Hybrid SCAP` is invoked to reassign causality, the BD is reconfigured based on the new causality assignment, and the simulation is resumed.

Recall our requirement that the overall model is causally valid in all system modes, i.e. we can assign a preferred integral causality to the HBG in every mode of the hybrid system, and it is the modeler's task to ensure this. Since we do not validate the causal assignment for each mode, during simulation, a mode can be encountered where valid causality assignment does not exist. In our implementation, we deal with this as an 'invalid causality' error, which terminates the simulation.

8. Case study: The advanced diagnostics and prognostics testbed

We use the MoTHS tool suite to model the Advanced Diagnostics and Prognostics Testbed at NASA Ames Research Center,[9] and construct automatically a simulation testbed called VIRTUAL ADAPT. ADAPT is functionally representative of a spacecraft electrical power distribution system, and includes two battery chargers, three sets of lead–acid batteries, two inverters, a number of relays and circuit breakers, and a variety of DC and AC loads. Relays configure the system into different modes of operation, therefore, the system behavior is naturally hybrid. The large number of modes makes it infeasible to pre-enumerate all possible modes of operation. Hence, ADAPT serves as an ideal example to highlight the effectiveness of our modeling and simulation methodology. Moreover, the battery, inverter, and AC and DC loads are highly non-linear, and help us demonstrate the ability of our simulation models to correctly capture such complex behaviors. In the following, we first present the HBG models of the different components of ADAPT that we use in our experiments, describe how our approach is applied to building VIRTUAL ADAPT, and present experimental results.

8.1. Component models

8.1.1. Lead–acid batteries. We develop an equivalent circuit model for the battery, shown in Figure 15, based on the model presented in Barsali and Ceraolo.[17,18] The charge-holding capacity of the battery is modeled by a large capacitance, C_0. The current

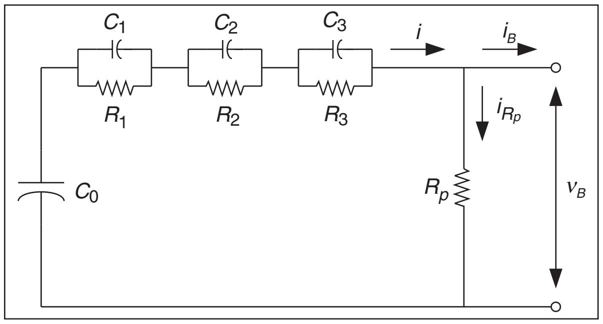


Figure 15. Equivalent circuit mode of the non-linear battery.

flowing through the battery, i_B, is comprised of i, the current that goes into discharging/charging the battery, and i_{R_p}, the current lost to parasitic reactions (modeled by R_p) due to gas emission and other sources of internal energy loss. The internal resistance parameters, R_1, R_2, and R_3 are non-linear functions of battery temperature, θ, discharge current, i_B, state of charge, SOC, and depth of charge, DOC:

$$R_1 = R_{10} + A_{11}SOC$$
$$R_2 = -R_{20}\ln(DOC)$$
$$R_3 = \frac{R_{30}\exp A_{31}(1 - SOC)}{1 + \exp A_{32}i_B}$$

where θ is a function of ambient temperature, θ_a, and power dissipated through resistances, P_B, given by

$$P_B = i_{R_p}v_{R_p} + \sum_{j=1}^{3}|i_{R_j}v_{R_j}|$$
$$\frac{d}{dt}\theta = \frac{1}{C_\theta}\left(P_B - \frac{\theta - \theta_a}{R_\theta}\right).$$

Note that the R and C elements in the above equations are based on the electrical circuit equivalent, and have no direct physical interpretation.

State of charge and depth of charge are computed parameters representing the fraction of useful charge left in the battery based on the maximum available charge, $Q_{\max}$, the actual available charge, q, and the relative battery capacity for the present battery temperature. Depth of charge is also a function of the present discharge current. The equations describing these parameters are as follows:

$$SOC = 1 - \frac{Q_{\max} - q}{K_c C_0^*(1 - \theta/\theta_f)^\epsilon}$$
$$DOC = 1 - \frac{Q_{\max} - q}{K_c C_0^*(1 - \theta/\theta_f)^\epsilon}\left(1 + (K_c - 1)\left(\frac{i_B}{I^*}\right)^\delta\right).$$

The HBG component model of the battery is shown in Figure 16. All elements in the battery HBG model are in fixed causality. Essentially, C_0 acts as a large capacitance with a voltage proportional to the amount of charge in the battery. When discharging, the *R–C* pairs subtract from this voltage and produce the non-linear curve in the battery output voltage. The battery parameters were identified using data from a set of experiments conducted on ADAPT, and their values are given in Table 2. The battery component can be connected to other components using its energy port.

8.1.2. Inverter. The inverter, modeled as a two-stage boost-buck DC–AC converter, consists of a cascade connection between a boost DC–DC converter with a full-bridge buck DC–AC converter, to achieve a transformerless DC–AC step-up conversion[2] (see Figure 17).

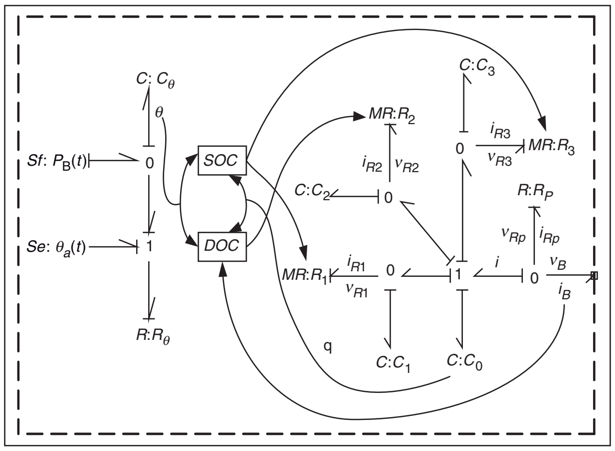


Figure 16. Battery hybrid bond graph component model.

Table 2. Identified battery parameters

Ambient Temperature	$\theta_a = 22°C$
Thermal Capacitance	$C_\theta = 615.3$ Wh/°C
Thermal Resistance	$R_\theta = 0.01$°C/W
Battery Capacitance	$C_0 = 106{,}360$ F
Parasitic Resistance	$R_p = 500\ \Omega$
Maximum Battery Charge	$Q_{\max} = 2{,}765{,}360$ C
Electrolyte Freezing Temperature	$\theta_f = -35°C$
Empirical Coefficients	$C_1 = 51.079$ F, $C_2 = 51.216$ F
	$C_3 = 567.56$ F, $R_{10} = 0.05582\ \Omega$
	$A_{11} = -0.025025$, $R_{20} = 0.001847\ \Omega$
	$R_{30} = 0.3579\ \Omega$, $A_{31} = -2.5315$
	$A_{32} = 0.22208$, $K_c = 1.33$
	$C_0^* = 270{,}720$ Ah, $\epsilon = 0.642$
	$\delta = 0.61$, $I^* = 5$ A

The boost converter first boosts the input DC voltage to a higher value (190 V, in our case), and then the buck converter generates the sinusoidal AC voltage. The fast switching in the boost and buck converters is controlled using two sliding mode controllers, one for each stage of the inverter.[2]

The equivalent circuit model of the boost-buck DC–AC inverter is given in Figure 18, where Sw_1 is a conventional power switch, and Sw_2 corresponds to a full bridge switch. The control signals for Sw_1 and Sw_2 are represented by u_1 and u_2, respectively. The differential equation model of the system, which can be found in Biel et al.,[2] is as follows:

$$\frac{\mathrm{d}}{\mathrm{d}t} i_1 = \frac{1}{L_1}(E_b - v_1(1-u_1))$$
$$\frac{\mathrm{d}}{\mathrm{d}t} v_1 = \frac{1}{C_1}(i_1(1-u_1) - i_2 u_2)$$
$$\frac{\mathrm{d}}{\mathrm{d}t} i_2 = \frac{1}{L_2}(v_1 u_2 - v_2)$$
$$\frac{\mathrm{d}}{\mathrm{d}t} v_2 = \frac{1}{C_2}\left(i_2 - \frac{v_2}{Z_{\text{load}}} - \frac{v_2}{R_{\text{on}}}\right)$$

where i_1 and i_2 represent the current through inductors L_1 and L_2, respectively, v_1 and v_2 represent the voltage across capacitors C_1 and C_2, respectively, $u_1 = \{0, 1\}$ and $u_2 = \{-1, 1\}$ represent switching signals, E_b is the input DC voltage to the inverter, Z_{load} is the load impedance, and R_{on} is the internal resistance of the inverter that accounts for no-load current draw of the inverter. The equations for determining u_1 and u_2 are obtained from Biel et al.,[2] and given below:

$$u_1 = \begin{cases} 1 & \text{if } \sigma_1\left[\frac{\alpha}{L_1} v_1 - \frac{\beta}{C_1} i_1\right] < 0 \\ 0 & \text{if } \sigma_1\left[\frac{\alpha}{L_1} v_1 - \frac{\beta}{C_1} i_1\right] > 0 \end{cases}$$

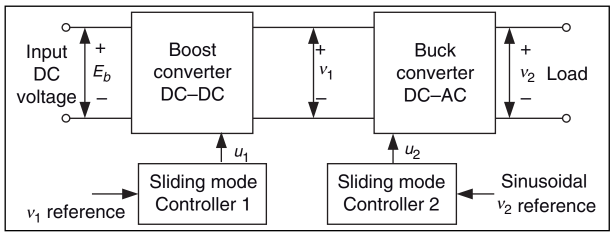


Figure 17. Block diagram of a boost-buck AC inverter.

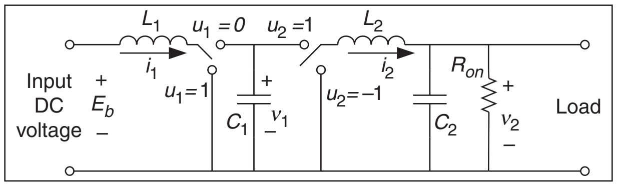


Figure 18. Circuit model of a boost-buck AC inverter.

where $\sigma_1 = \alpha i_1 + \beta v_1 - \delta \int (v_{1\text{Ref}} - v_1), \mathrm{d}t - K$, and

$$u_2 = \begin{cases} 1 & \text{if } \operatorname{sgn}(\sigma_2) < 0 \\ -1 & \text{if } \operatorname{sgn}(\sigma_2) > 0 \end{cases}$$

where

$$\sigma_2 = a_1(v_{2\text{Ref}} - v_2) + a_2\left(\frac{\mathrm{d}}{\mathrm{d}t} v_{2\text{Ref}} - \frac{\mathrm{d}}{\mathrm{d}t} v_2\right).$$

The identified parameters of the inverter model, based on experiments run on ADAPT, are shown in Table 3.

The HBG component model of the inverter, shown in Figure 19, is derived from its circuit model. Switch Sw_1 is represented by switching junctions 0_b and 1_c, which operate synchronously. Switch Sw_2 is represented by the switching junctions 1_e, 1_g, 0_f, and 0_h, with 1_e and 1_g having the same *CSPEC* as junctions 0_f and 0_h, respectively. Junctions 1_e and 0_f are switched on when $u_2 = -1$, and switched off when $u_2 \neq -1$. However, junctions 1_g and 0_h are switched on when $u_2 = 1$, and switched off when $u_2 \neq 1$. Effectively, junctions 1_e and 0_f are on when junctions 1_g and 0_h are off, and vice versa. The commutation, or change in direction, of the voltage polarity is brought about by the opposite directions of bonds incident on junctions 1_e and 0_f, and 1_g and 0_h, respectively. The sliding mode controllers generate signals that switch the inverter junctions at kilohertz frequency. In the inverter component model shown in Figure 19, all but the switched junctions, and all bonds except those shared by adjacent switched junctions are in fixed causality, and marked appropriately.

Table 3. Identified inverter parameters

Inductances	$L_1 = 0.0022$ H, $L_2 = 0.075$ H
Capacitances	$C_1 = 0.0069$ F, $C_2 = 6 \times 10^{-6}$ F
Resistances	$R_{\text{on}} = 489.49\ \Omega$
Sliding mode	$\alpha = 0.8$, $\beta = 4.3649$,
Controller 1 parameters	$\delta = 111.375$, $K = 829.3347$
Sliding mode	$a_1 = 15.915$,
Controller 2 parameters	$a_2 = 0.0048$
Reference Voltages	$v_{1\text{Ref}} = 190$ V,
	$v_{2\text{Ref}} = 120\sqrt{2}\sin(120\pi)$ V

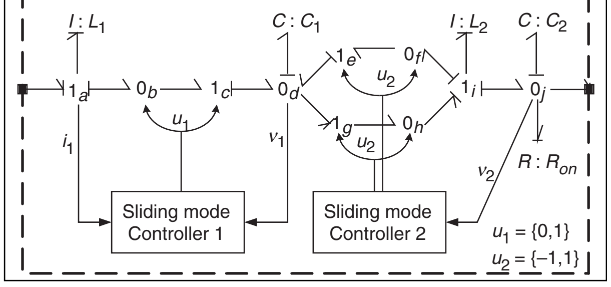


Figure 19. Inverter hybrid bond graph component model.

8.1.3. Relay.

The relay HBG model consists of a single switching 1-junction with two energy ports. The junction models the relay's switching behavior, by turning on or off depending on an external control signal. The relay CSPEC model essentially is the same as that of the relay Sw_1, described in Example 1.

8.1.4. Light bulb.

The light bulb component model consists of a single energy port, a 1-junction, and an R element whose resistance value is set to 234.7 Ω.

8.1.5. AC fan.

The AC fan is modeled as a single phase, fixed capacitor induction motor.[19] We represent the system using the standard $d-q$ model, described by Krause:[20]

$$V_{qs} = R_s i_{qs} + \frac{\mathrm{d}}{\mathrm{d}t}\phi_{qs}$$
$$V_{ds} = R_S i_{ds} + \frac{\mathrm{d}}{\mathrm{d}t}\phi_{ds}$$
$$V_{qr} = R_r i_{qr} + \frac{\mathrm{d}}{\mathrm{d}t}\phi_{qr} - \frac{p}{N}\omega_m \phi_{dr}$$
$$V_{dr} = R_R i_{dr} + \frac{\mathrm{d}}{\mathrm{d}t}\phi_{dr} + pN\omega_m \phi_{qr}$$
$$T = p\left(N\phi_{qr} i_{dr} - \frac{1}{N}\phi_{dr} i_{qr}\right)$$
$$\frac{\mathrm{d}}{\mathrm{d}t}\omega_m = \frac{1}{2J}(T - B\omega_m)$$

where V denotes voltage, ϕ denotes flux, i denotes current, p is the number of pole pairs in the motor, N is the ratio of the number of auxiliary winding's effective turns and the number of main winding's effective turns, T is the torque, J is inertia of the motor shaft, B is friction in the motor shaft, and ω_m is the angular velocity of fan. Subscripts q and d represent the q and d axes, respectively, and r and s denotes rotor and stator, respectively. We assume that the blades of the fan are directly attached to the shaft of the motor, and hence the inertia and friction of the blades are lumped with those of the motor shaft. All other parameters have their standard interpretations, i.e. $\phi_{qs} = L_{ss} i_{qs} + L_{ms} i_{qr}$, $\phi_{ds} = L_{SS} i_{ds} + L_{mS} i_{dr}$, $\phi_{qr} = L_{rr} i_{qr} + L_{ms} i_{qs}$, and $\phi_{dr} = L_{RR} i_{dr} + L_{mS} i_{ds}$.[20]

Note that the AC fan has two 2-port I-fields with parameters,

$$I_1 = \begin{bmatrix} L_{ss} & L_{ms} \\ L_{ms} & L'_{rr} \end{bmatrix}, \quad \text{and} \quad I_2 = \begin{bmatrix} L_{SS} & L_{ms} \\ L_{ms} & L'_{RR} \end{bmatrix}.$$

The identified model parameters are presented in Table 4. The HBG model of the fan, shown in Figure 20, extends the induction motor model described by Karnopp,[21] by adding resistance B to model the friction of the motor shaft. Figure 20 also shows how the single-phase output voltage from the inverter and its quadrature are added as the two inputs to the motor model presented by Karnopp.[21] We convert the voltage and current from the $a-b$ to the $d-q$ reference frame, using two *TF*s (see Figure 20).

8.2. Virtual ADAPT

Using our modeling and simulation methodology, we have developed VIRTUAL ADAPT, a high-fidelity simulation testbed for the ADAPT system. First, we developed a component library that includes the HBG component models of the different ADAPT components, many of which are described above. These component models were constructed using our modeling language in GME. Then, we estimated the parameters of each component model and validated these models using the testbed data. To generate the HBG model for the complete ADAPT system, the different components were instantiated from the component library, and connected appropriately.

Once all of the component models were connected, the Simulink model for the HBG was constructed automatically through the interpretation process of MoTHS applied to this composed model. In the complete VIRTUAL ADAPT model, all non-switching junctions are in fixed causality, and all switching junctions satisfy the conditions for avoiding calls to `Hybrid SCAP`. Hence, all reconfigurations in VIRTUAL ADAPT are local, thereby minimizing computations due to mode changes and increasing simulation efficiency.

VIRTUAL ADAPT offers a number of advantages to the end user. As a simulation testbed, VIRTUAL ADAPT provides a light-weight, portable alternative to the actual ADAPT system, and uses identical software interfaces as the actual testbed. Therefore, applications, such as a diagnostic reasoner, can be transparently developed and tested using VIRTUAL ADAPT, and then run on the actual system. This is especially useful for running offline tests, when the actual system is unavailable. Since our implementation allows the injection of faults in the simulation testbeds, faults which are dangerous or infeasible to inject in the real testbed can be introduced in VIRTUAL ADAPT, thereby increasing the functionality of the actual testbed.

Table 4. Identified AC fan parameters

Inductances	$L_{ss}=0.275$ H, $L_{SS}=0.274$ H, $L'_{rr}=0.272$ H, $L'_{RR}=0.271$ H, $L_{ms}=0.1772$ H, $L_{mS}=0.2467$ H
Inertia	$J=6.5\times10^{-4}$ kg m^2
Capacitances	$C_{\text{startRun}}=21.1$ μF
Resistances	$R_s=163.02\,\Omega$, $R_S=168.14\,\Omega$, $R'_r=145.12\,\Omega$, $R'_R=145.12\,\Omega$, $R_{\text{startRun}}=26\,\Omega$,
Friction	$B=4.734\times10^{-4}$ kg m^2 s^{-1}
Other parameters	$N=1.18$, $p=2$

8.3. Experimental results

In the following, we present the results of two experiments. The first experiment is designed to demonstrate how we can model and simulate both nominal and faulty system behavior using MoTHS, and the second is to illustrate the improvements in simulation efficiency gained by our proposed approach to causal reassignment. All experiments were performed on an 2.4 GHz Intel® Pentium Core™2 Duo CPU desktop, having 2 GB of RAM. The model was simulated using a

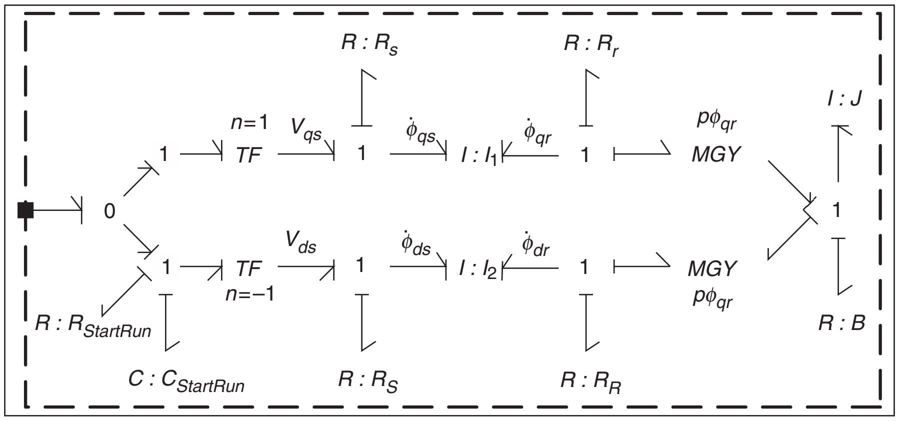


Figure 20. AC fan hybrid bond graph.

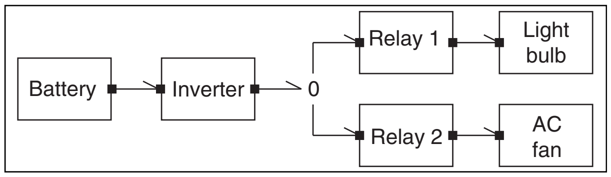


Figure 21. ADAPT subsystem for case study.

fixed-step simulation with a sample period of 7.5 μs, as required to model the fast switching of the inverter.

8.3.1. Experimental configuration. Figure 21 shows the system configuration we used in our experiments to demonstrate our simulation approach. The configuration consists of a subset of VIRTUAL ADAPT , including a battery connected to a light bulb and an AC fan through the inverter and two relays. The battery discharges through the inverter, and relays in the circuit enable the loads to be switched online or offline. As described previously, each of the two relays has one switching junction, and the inverter has three pairs of switching junctions, theoretically resulting in 2^8 possible modes. Even with such a small number of components, the large number of system modes make the advantage of our approach clear.

8.3.2. Experiment 1: Simulating nominal and faulty system behavior. In Experiment 1, we simulate VIRTUAL ADAPT in the chosen experimental configuration. Two simulation runs are performed: one nominal, and the other with a fault injected in the light bulb. The goal of this experiment is to demonstrate the ability of our modeling and simulation methodology to correctly simulate both nominal and faulty system behavior.

Figure 22 shows the results of the nominal and faulty simulation runs. We plot the voltages and currents at the output of the battery and the inverter, as well as the rotational speed of the AC fan. The Simulink model was executed for 20 seconds of simulation time. First the light bulb is connected to the inverter from $t=2$ to 5 s. An abrupt fault that is a 30% decrease in the light bulb resistance is injected at 3 s. As we can see, the sliding mode controllers are robust to load changes, and generates true 120 V RMS voltage for both the load configurations, despite the occurrence of the fault. However, the light bulb fault affects the inverter current, and therefore, the battery current and voltage. Figure 23 shows the battery current, inverter current and inverter voltage immediately before and after the fault is injected.

The fan is switched on between $t=7$ and 15 s, and results in a phase difference of 0.1346 rad between the inverter current and voltage. When the fan is switched on, its speed of rotation increases until it reaches a steady state of about 78.5 rad s^{-1}. On turning off, the velocity reduces to zero.

8.3.3 Experiment 2: Gauging the efficiency of our simulation approach. Table 5 presents the results of an experiment to illustrate the efficiency gains obtained by (i) the implementation of the causal reconfiguration using incremental `Hybrid SCAP` instead of `SCAP`, and (ii) the simplification of the reconfigurable BD model to avoid the need for causal reassignment when propagation of causal changes does not occur. For this experiment, we assume that the fan is the only load and is turned on for the entire duration of the experiment. Consider the inverter HBG model, shown in Figure 19. All non-switched junctions in this model are in fixed causality, because the adjacent energy storage elements specify a unique determining bond for these junctions. All switched junctions are not only in constrained causality, but their determining bonds are invariant whenever they are on, and their turning off does not affect the causality of any adjacent HBG elements. For example, consider 0_b and 1_c. Since they always change modes simultaneously (because they share the same CSPEC), when on, 0_b always imposes flow on its adjacent junction 1_c that is also on. When they are off, the causality assignment of other active junctions are not affected. The case is similar for pairs 1_e and 0_f, and 1_g and 0_h. Therefore, the switching of the inverter junctions does not necessitate calling `Hybrid SCAP`.

Although in the given experimental configuration, mode changes of the switching junctions do not require external calls to the `Hybrid SCAP` algorithm, for comparison purposes, we also invoke `SCAP` and `Hybrid SCAP` to reassign causality *every* time an inverter mode change occurs, in the first and second runs of this experiment, respectively. In the third run, we simulate the HBG without requiring any external calls to `Hybrid SCAP`.

The second column in Table 5 reports the real time taken to simulate 1 s of simulation time for the different simulation runs, each with a different approach for causal reassignment. As can be seen from the second column of Table 5, the first run is faster than the second run, thereby showcasing the general gains in efficiency `Hybrid SCAP` provides over `SCAP`. Also, the third run is 225.07 times faster than the first run, and 127.43 times faster than the second run, thus establishing that avoiding calls to the causal reassignment procedure results in efficiency gains. Much of the extra time in the first two runs can be attributed to the calls made to the external `SCAP` and `Hybrid SCAP` algorithms. The third column in Table 5 reports the average time taken by the different approaches to reassign causality after a mode change. This column clearly demonstrates the increase in

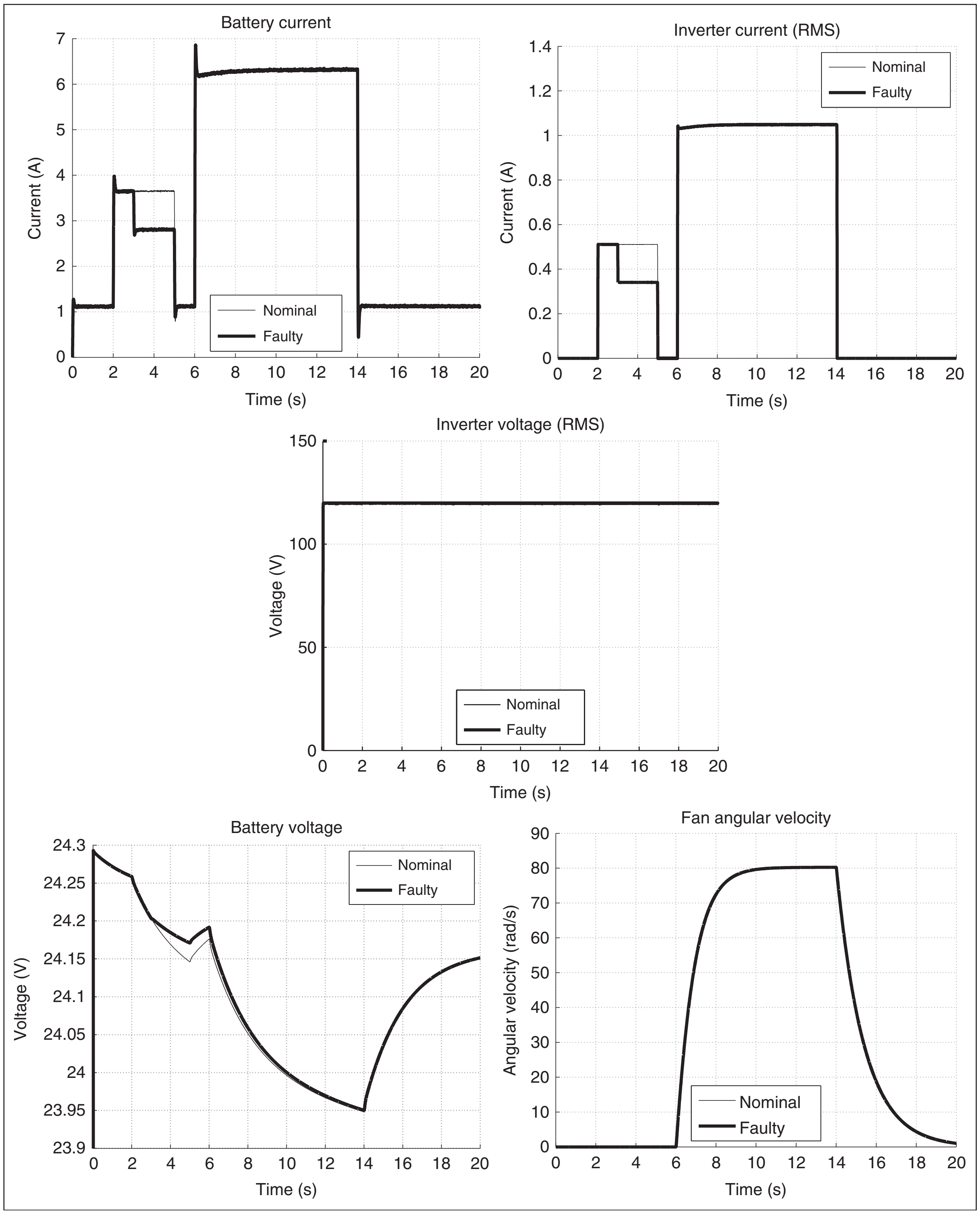


Figure 22. Simulation results.

efficiency of causal reassignment using `Hybrid SCAP` over `SCAP`. In our current example, `Hybrid SCAP`, on average, performed causal reassignment about 13 times faster than `SCAP`. Our simulation approach resulted in considerable improvements in the efficiency of simulation of a number of other configurations, especially for large systems such as the VIRTUAL ADAPT simulation testbed.[9] Further increases in

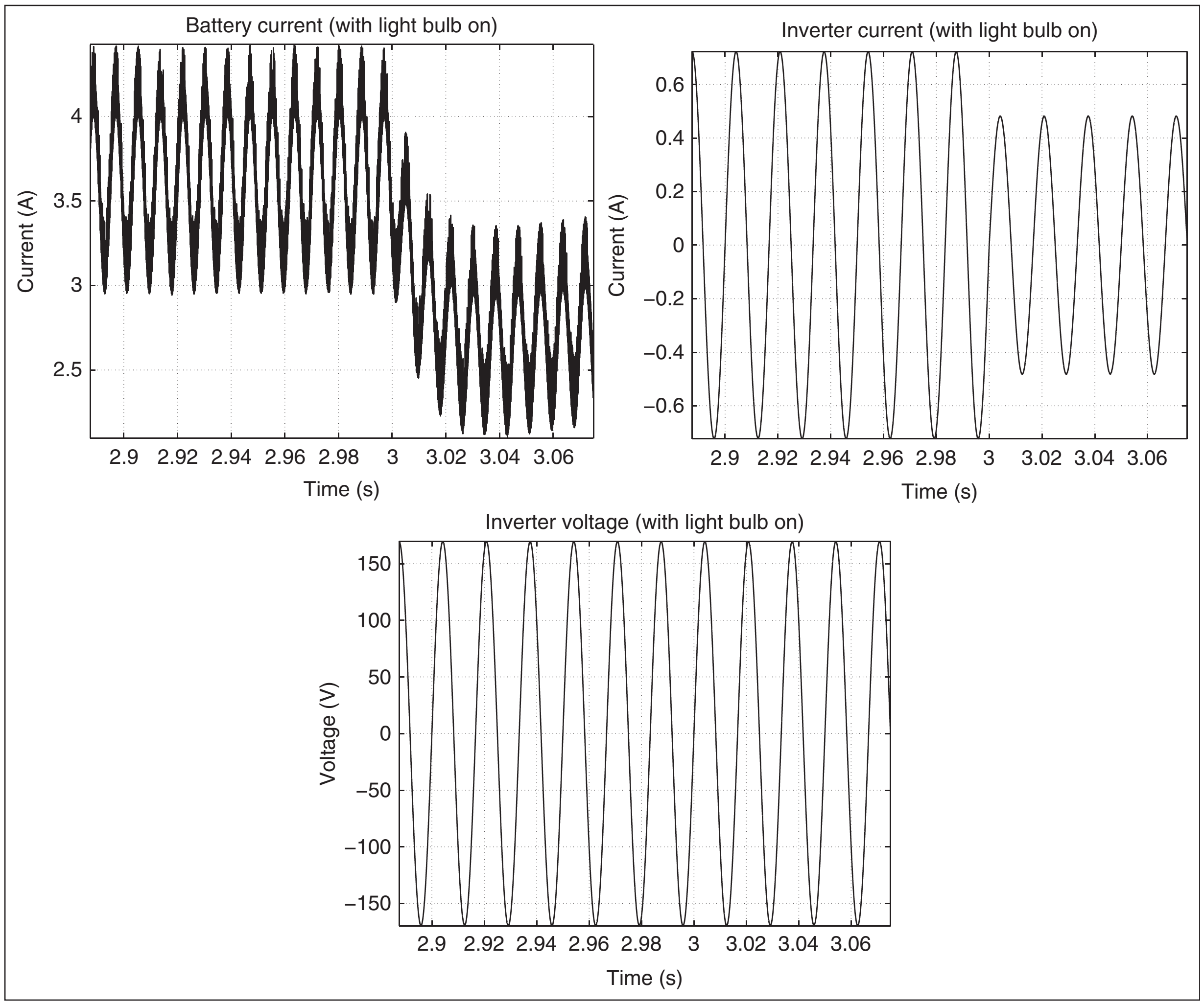


Figure 23. Detailed plots of selected measurements near the time of fault injection.

Table 5. Comparison of causal reassignment methods

Method	Real time taken for 1 s of simulation time	Average time for causal reassignment per mode change
`SCAP` called at every mode change	13,121.75 s	0.1723 s
`Hybrid SCAP` called at every mode change	7,428.99 s	0.0136 s
No causal reassignment procedure called	58.30 s	0.0000 s

simulation efficiency can be obtained by running our simulation models in the Rapid Accelerator mode of Simulink. Note that the improvements in simulation time using `Hybrid SCAP` depend on the switching characteristics of an HBG. Hence, the more the switchings during the whole simulation, the more the improvement in simulation performance, since the efficiency improvements due to efficient causal reassignment will be leveraged.

9. Related work

Explicitly pre-enumerating the continuous system dynamics for each global system is the most naive approach to modeling hybrid systems. Such approaches include modeling languages, such as hybrid automata, the most general computational models for hybrid systems.[22] However, pre-enumeration is only feasible if the number of system modes is small, which, for real systems, is seldom the case. A solution to this issue is presented in Borutzky,[23] where a hybrid system is modeled using a Petri net or finite state machine to pre-enumerate only the more important operating modes of the system, and defining the continuous behavior of the system in each of these modes using a separate BG. The generation of the complete BG for each mode, and its causality assignment can be tedious. Also, the selection of important modes and the transitions between these modes can be error-prone. Hierarchical, component-based approaches, such as those presented by Edström,[24] Roychoudhury et al.,[11] Liu et al.,[25] Hofbaur and Wotawa,[16] avoid the pre-enumeration of system modes, thereby enabling concise representation of hybrid systems. The global system mode is determined by composing the modes of each individual component. Hence, the issue of not modeling important modes does not arise in component-based approaches.

A general, hierarchical hybrid system modeling and simulation framework using the Ptolemy environment was presented in Liu et al.[25] Hierarchical modeling of hybrid systems is supported by executable *actors*, where each actor models a subsystem. Actors can be atomic or composite, with arbitrary levels of nesting. In Ptolemy, a hierarchical hybrid automaton can be modeled as a continuous-time actor at the top-most level containing one or more finite state automata. The simulation is executed in continuous time, and the continuous time model is a composition of the continuous models of the current state of each automaton. Guard conditions are evaluated at discrete time points, and if they are enabled, a state transition is executed, and continuous time simulation continues with a new model with the initial state being defined by the reset conditions. In this approach, the modeler is required to explicitly enumerate the continuous-time behavior in each subsystem mode at the lowest level of abstraction. Also, since Ptolemy is a general modeling approach, the modeler has to determine the causal relations between the different variables in different states of the finite state automaton. In BGs and HBGs, the causality information in inherent in the model structure, and systematic procedures can be used for automatically assigning causality and deriving computational models from BGs and HBGs. Moreover, BGs and HBGs are designed for modeling physical systems, and so, such systems are easier to model with HBGs, than general languages, such as hybrid automata.

Several approaches have been proposed for simulating continuous system behavior using BGs. Some of these approaches are developed for simulating hybrid system behavior and, hence, implicitly simulates continuous system behavior. The CAMP-G system[26,27] compiles the bond graph equations into code form for execution as MATLAB M-functions or Simulink S-functions. In Dymola,[28] bonds with variable causality can be implemented as acausal bonds, which are converted into a set of differential algebraic equations for simulation.[29] The simulation software[30,31] SYMBOLS 2000 makes use of *capsules* (encapsulated objects), which include derived equations of subsystems, with either fixed or generic causality. The notion of capsules, therefore, is similar to our HBG components. SYMBOLS 2000 performs causality assignment by solving the equations symbolically. Our modeling and simulation methodology, although designed primarily for HBGs, applies to BGs as well. Our approach, in the absence of algebraic loops, exploits causality to express the system dynamics as a system of ODEs, which are then implemented using BDs. If algebraic loops are present, our approach generates BD structures that correspond to DAEs. Solving DAEs requires more sophisticated iterative computational solvers, and in some cases, may face numerical convergence problems. Our modeling and simulation methodology depends entirely on the solvers of existing simulation applications for the implementation of the procedures for generating the ODEs or DAEs from the BD and simulating them.

Extending BGs to model hybrid systems has been studied by a number of researchers.[23,32,33,34] Karnopp and Rosenberg,[35] Garcia et al.,[36] and Granda et al.[26,27] introduced non-linear resistances and Boolean-valued modulated transformers to model discontinuous behavior, but the conditions for idealized, lossless switching are violated. An ideal switch, *Sw*, was introduced as a new BG element by Buisson et al.[37] and Strömberg et al.,[38] and switching bonds were proposed by Broenink and Wijbrans.[39] HBGs augment BGs by including switching junctions for modeling discrete mode changes. Switching junctions do not violate the basic energy relations in the system for handling idealized mode changes.[4,40] Furthermore, the implementation of the switching junction maintains a separation between the switching scheme and the energy transfer principles that govern the behavior of continuous systems.

The inclusion of *Sw* elements, switching bonds, or switching junctions within BGs may necessitate reassignment of causality when mode changes occur. A number of different approaches have been proposed for updating the causal assignment of the reconfigured HBG model after a mode change occurs. Some approaches recompute the causal assignments in their entirety and regenerate the model. Edström[24] and Manders et al.[41] applied causal assignment to the entire model at each mode change to generate the new equations. Incremental reassignment is more efficient because in most cases, as we have shown, only a small part of the computational model will change from one mode to the next. Therefore, in our approach, we implement an incremental causal reassignment procedure that uses the causal assignment in the previous mode to efficiently assign causality in the new mode and reconfigure the BD structure according to the new causal assignment.

HBGs capture a set of ODEs or DAEs in each mode of system operation. The equations can be represented in implicit or explicit form. CAMP-G formulates the constituent system equations explicitly, but Dymola models the system equations implicitly. HyBrSim[42] captures the system equations in implicit form, and uses algebraic solvers for simulating the system. Our modeling and simulation approach represents the equations in explicit form, and relies on the solvers used by existing simulation applications, for various hybrid simulation algorithms, such as zero-crossing detection.

10. Discussion and conclusions

In this paper, we have presented an approach for modeling and simulation of complex systems with switching behaviors using HBGs. Our approach for building computational models from HBGs provides a comprehensive framework starting from component-based physical system models and deriving efficient computational models in the form of reconfigurable block diagrams for hybrid systems. Typically, simulation of hybrid systems requires pre-enumeration of models for different operational modes, which is space-inefficient, or regeneration of models when mode changes occur during simulation, which is time-inefficient. In this work, we retain the space-efficiency of simulation-time model generation, but make the incremental model regeneration process time-efficient by recognizing that only parts of the model change causal assignments, and only these parts need to be reconfigured when mode changes occur. The notion of fixed causality of bonds and elements, derived from model topology, facilitates identification of model fragments that retain the same configuration in all modes of system operation. In addition, the notion of forced causality forms the basis of our causality reassignment procedure, `Hybrid SCAP`, applied when mode changes occur. The algorithm, an extension of the traditional `SCAP` algorithm, incrementally updates the causality assignment from the previous mode to that of the new mode. Reducing the number of possible block diagram configurations, and minimizing the number of changes that need to be made when mode changes occur results in efficient computational models for simulation as has been demonstrated in our case study of the ADAPT system.

We implemented our framework as the MoTHS tool suite, with a two-stage model interpreter. The first stage converts component-based HBG models to efficient computational models implemented as reconfigurable block diagrams, and the second translates the BD models to executable Matlab Simulink models. We used MoTHS to build Virtual ADAPT, a high-fidelity simulation testbed for ADAPT, a large electrical power distribution system with complex hybrid and non-linear behaviors. Simulation results demonstrated the capability of our framework to correctly capture both nominal and faulty operation scenarios. Experimental results also demonstrate the efficiency improvements our methodology can offer.

The integral causality assumption for energy storage elements is important for our time- and space-efficient simulation approach. Allowing HBG elements to be in derivative causality may introduce numerical problems during simulation, for example, under derivative causality, the current through a capacitance will be infinity when the voltage across the capacitance increases stepwise. Simulating HBG elements in derivative causality will also imply knowing a future value to correctly compute the derivative at the current time point, and derivatives of signals are noisier than their integrals. Hence, the integral causality assumption is more practical for our simulation purposes. However, if the integral causality assumption is relaxed, and derivative causality assignment is allowed, some of the gains in space efficiency of the reconfigurable BD would be lost, since C or I elements would not be assigned fixed causality, and the assignment of fixed causality for HBG elements would depend on how far the fixed causality of energy source elements would propagate, which is typically not much. Fewer fixed causality assignments results in a decrease of space-efficiency of the reconfigurable BD models. For the online causal reassignment procedure, the C or I elements would be treated similarly to R elements with preference given to integral causality. We would still obtain considerable savings in this scenario compared with running `SCAP` on the entire HBG every time a mode change occurred, because causality reassignment would still not likely propagate very far.

As part of future work, we will apply our modeling approach and computational model generation schemes to other large, real-world hybrid systems. We also believe the main idea of our HBG-based approach can be extended to other component-based hybrid systems modeling frameworks. For example, causal information extracted from the modeling paradigm, or made available otherwise, can lead to the assignment of fixed causal forms on equation fragments used in modeling languages such as Modelica.[43,44] In addition, we also wish to build additional interpreters in the MoTHS framework to generate executable simulations in other graphical simulation environments. We would also like to extend our simulation model generation approach to include intelligent model validation algorithms (such as that presented by Hofbaur and Wotawa[16]) to evaluate the correctness of the model across all system modes before simulating. Finally, we would like to further improve simulation efficiency by including a caching mechanism that avoids having to recalculate causal assignment updates for system modes that have occurred previously.

11 Acknowledgements

We gratefully acknowledge the help from Scott Poll and Ann Patterson-Hine for supporting our efforts with the ADAPT system, and Pieter Mosterman, whose comments helped improve the technical content of the paper. We would also like to thank the anonymous reviewers for their constructive comments in improving the technical quality and presentation of this paper.

This work was supported in part by the National Aeronautics and Space Administration (NRA NNX07AD12A and USRA 08020-013), and the National Science Foundation (grant numbers CNS-0615214 and CNS-0347440).

References

1. Alur R, Courcoubetis C, Henzinger TA, and Ho PH. *Hybrid Automata: An Algorithmic Approach to the Specification and Verification of Hybrid Systems*. Berlin: Springer, 1993, p.209–229.
2. Biel D, Guinjoan F, Fossas E, and Chavarria J. Sliding-mode control design of a boost-buck switching converter for ac signal generation. *IEEE Transactions on Circuits and Systems - I* 2004; 51(8): 1539–1551.
3. Mosterman PJ, Biswas G. A theory of discontinuities in physical system models. *Journal of the Franklin Institute* 1998; 335B(3): 401–439.
4. Mosterman PJ. Hybrid Dynamic Systems: A Hybrid Bond Graph Modeling Paradigm and its Application in Diagnosis. PhD thesis. Vanderbilt University, 1997.
5. Karnopp DC, Margolis DL, and Rosenberg RC. *Systems Dynamics: Modeling and Simulation of Mechatronic Systems*, 3rd edn. New York: John Wiley & Sons, Inc., 2000.
6. Manders E-J, Biswas G, Mahadevan N, and Karsai G. Component-oriented modeling of hybrid dynamic systems using the Generic Modeling Environment. *Proceedings of the 4th Workshop on Model-Based Development of Computer Based Systems*. Potsdam, Germany, March 2006. Los Alamitos, CA: IEEE Computer Society Press.
7. Karsai G, Sztipanovits J, Ledeczi A, and Bapty T. Model-integrated development of embedded software. *Proceedings of the IEEE* 2003; 91: 145–164.
8. MATLAB/Simulink. http://www.mathworks.com/products/simulink/
9. Poll S, Patterson-Hine A, Camisa J, Nishikawa D, Spirkovska L, Garcia D, et al. Evaluation, selection, and application of model-based diagnosis tools and approaches. In *AIAA Infotech@Aerospace 2007 Conference and Exhibit*, May 2007.
10. Buck J, Ha S, Lee EA, and Messerschmitt DG. Ptolemy: a framework for simulating and prototyping heterogeneous systems. *Readings in Hardware/Software Co-design* 2002; 527–543.
11. Roychoudhury I, Daigle M, Biswas G, Koutsoukos X, and Mosterman PJ. A method for efficient simulation of hybrid bond graphs. In: *Proceedings of the International Conference on Bond Graph Modeling and Simulation*, San Diego, CA. 2007, pp. 177–184.
12. Antic D, Vidojkovic B, and Mladenovic M. An introduction to bond graph modelling of dynamic systems. In: *4th International Conference on Telecommunications in Modern Satellite, Cable and Broadcasting Services*. Vol. 2, pp. 661–664.
13. Van Dijk J. On the Role of Bond Graph Causality in Modelling Mechatronic Systems. PhD thesis. University of Twente, 1994.
14. Joseph BJ, Martens HR. The method of relaxed causality in the bond graph analysis of nonlinear systems. *ASME Journal of Dynamic Systems, Measurement, and Control* 1974; 96(1): 95–99.
15. Karnopp DC. Lagrange's equations for complex bond graph systems. *ASME Journal of Dynamic Systems, Measurement, and Control* 1977; 99(4): 300–306.
16. Hofbaur MW, Wotawa F. A causal analysis method for concurrent hybrid automata. In: *Proceedings of the 21st AAAI Conference on Artificial Intelligence*. 2006, pp. 840–846.
17. Ceraolo M. New dynamical models of lead–acid batteries. *IEEE Transactions on Power Systems* 2000; 15(4): 1184–1190.
18. Barsali S, Ceraolo M. Dynamical models of lead–acid batteries: implementation issues. *IEEE Transactions on Energy Conversion* 2002; 17(1): 16–23.
19. Thaler GJ, Wilcox ML. *Electric Machines: Dynamics and Steady State*. New York: John Wiley & Sons, Inc, 1966.
20. Krause PC. *Analysis of Electric Machinery*. New York: John Wiley & Sons, Inc, 1986.
21. Karnopp D. Understanding induction motor state equations using bond graphs. In: *Proceedings of the International Conference on Bond Graph Modeling and Simulation*. Vol. 35, 2003, pp. 269–273.
22. Henzinger TA. The theory of hybrid automata. *Proceedings of the 11th Annual Symposium of Logics in Computer Science* 1996; 278–292.
23. Borutzky W. Discontinuities in a bond graph framework. *Journal of the Franklin Institute* 1995; 322B(2): 141–154.
24. Edström K. Simulation of Mode Switching Systems Using Switched Bond Graphs. PhD thesis. Linköpings Universitet, 1996.
25. Liu J, Liu X, Koo TJ, Sinopoli B, Sastry S, and Lee EA. A hierarchical hybrid system model and its simulation. In: *Proceedings of the 38th IEEE Conference on Decision and Control*. Vol. 4, pp. 3508–3513.
26. Granda JJ, Dauphin-Tanguy G, and Rombaut C. Power electronics converter-electrical machine assembly bond graph models simulated with CAMP/G-ACSL. In: *Proceedings of the IEEE International Conference*, France. 1993, October.

27. Granda JJ. The role of bond graph modeling and simulation in mechatronic systems, and integrated software tool: CAMP-G, MATLAB-simulink. *Mechatronics* 2002; 12: 1271–1295.
28. Dymola. http://www.dynasim.com/dymola.html
29. Cellier FE, McBride RT. Object-oriented modeling of complex physical systems using the Dymola bond-graph library. In: *Proceedings of the International Conference on Bond Graph Modeling and Simulation*, Orlando, FL. 2003, January 2003.
30. Mukherjee A, Samantaray AK. System modelling through bond graph objects on SYMBOLS 2000. In: *Proceedings of the International Conference on Bond Graph Modeling and Simulation*. 2001, pp. 164–170.
31. Bouamama BO, Samantaray AK, Medjaher K, Staroswiecki M, and Dauphin-Tanguy G. Model builder using functional and bond graph tools for FDI design. *Control Engineering Practice* 2005; 13(7): 875–891.
32. Borutzky W, Broenink JF, and Wijbrans KCJ. Graphical description of physical system models containing discontinuities. In: Pave A (ed.) In: *Modelling and Simulation, Proceedings of the European Simulation Multiconference*, Lyon, France. 1993, pp. 208–214, June 1993.
33. Söderman U, Top J, and Stromberg J. The conceptual side of mode switching. In: *Proceedings of the International Conference on System, Man and Cybernetics*. 1993.
34. Lorenz F, Haffaf H. Combinations of discontinuities in bond graphs. In: *Proceedings of the International Conference on Bond Graph Modeling and Simulation*, Las Vegas, NV. 1995, pp. 56–64, January 1995.
35. Karnopp D, Rosenberg RC. *Analysis and Simulation of Multiport Systems*. New York: John Wiley and Sons, 1975.
36. Garcia J, Dauphin-Tanguy G, and Rombaut C. Bond graph modeling of thermal effects in switching devices. In: *Proceedings of the International Conference on Bond Graph Modeling and Simulation*, Las Vegas, NV. January 1995, pp. 145–150.
37. Buisson J, Cormerais H, and Richard P-Y. Analysis of the bond graph model of hybrid physical systems with ideal switches. *Proceedings of the Institution of Mechanical Engineers Part I: Journal of Systems and Control Engineering* 2002; 216: 47–63.
38. Strömberg JE, Top J, and Söderman U. Variable causality in bond graphs caused by discrete effects. In: *Proceedings of the International Conference on Bond Graph Modeling*, San Diego, CA. 1993, pp. 115–119, 1993.
39. Broenink JF, Wijbrans KCJ. Describing discontinuities in bond graphs. In: *Proceedings of the International Conference on Bond Graph Modeling*, San Diego, CA. 1993, pp. 120–125.
40. Mosterman PJ, Biswas G. Behavior generation using model switching a hybrid bond graph modeling technique. In: *Proceedings of the International Conference on Bond Graph Modeling and Simulation*, Las Vegas, NV. 1995, pp. 177–182.
41. Manders E-J, Biswas G, Ramirez J, Mahadevan N, Wu J, and Abdelwahed S. A model integrated computing tool-suite for fault-adaptive control. In: *Proceedings of the 15th International Workshop on Principles of Diagnosis*. 2004.
42. Mosterman PJ. HybrSim—a modeling and simulation environment for hybrid bond graphs. *Journal of Systems and Control Engineering—Part I* 2002; 216(1): 35–46.
43. Broenink JF. Bond-graph modeling in Modelica. *Proceedings of the European Simulation Symposium* 1997; 137–141.
44. Modelica. http://www.modelica.org/

Indranil Roychoudhury received the B.E. (Hons.) degree in Electrical and Electronics Engineering from Birla Institute of Technology and Science, Pilani, Rajasthan, India in 2004, and the M.S. and Ph.D. degrees in Computer Science from Vanderbilt University, Nashville, TN, USA, in 2006 and 2009, respectively. From September 2004 to July 2009, he was a Graduate Research Assistant with the Institute for Software Integrated Systems, Department of Electrical Engineering and Computer Science, Vanderbilt University, Nashville, TN, USA. During the summers of 2006 and 2007, he was an intern with Mission Critical Technologies, Inc., at NASA Ames Research Center. Since August 2009, he has been with SGT Inc., at NASA Ames Research Center as a Computer Scientist, Post Doctorate. His research interests include hybrid systems modeling, model-based diagnosis, distributed diagnosis, and Bayesian diagnosis of complex physical systems. He is a member of the IEEE.

Matthew J. Daigle received the B.S. degree in Computer Science and Computer and Systems Engineering from Rensselaer Polytechnic Institute, Troy, NY, in 2004, and the M.S. and Ph.D. degrees in Computer Science from Vanderbilt University, Nashville, TN, in 2006 and 2008, respectively. From September 2004 to May 2008, he was a Graduate Research Assistant with the Institute for Software Integrated Systems and Department of Electrical Engineering and Computer Science, Vanderbilt University, Nashville, TN. During the summers of 2006 and 2007, he was an intern with Mission Critical Technologies, Inc., at NASA Ames Research Center. Since June 2008, he has been with the University of California, Santa Cruz, at NASA Ames Research Center, where he is currently an Associate Research Scientist in the Intelligent Systems Division. His current research interests include physics-based modeling, model-based diagnosis and prognosis, and hybrid systems. He is a recipient of the 4.0 Award and Ricketts Prize from Rensselaer Polytechnic Institute, a University Graduate Fellowship from Vanderbilt

University, and a Staff Recognition and Development Award from the University of California, Santa Cruz. He is a member of the IEEE.

Gautam Biswas is a Professor of Computer Science and Computer Engineering in the EECS Department and a Senior Research Scientist at the Institute for Software Integrated Systems (ISIS) at Vanderbilt University. He has a Ph.D. degree in Computer Science from Michigan State University in E. Lansing, MI. He conducts research in intelligent systems with primary interests in hybrid modeling, simulation, and analysis of complex embedded systems, and their applications to diagnosis and fault-adaptive control. As part of this work, he was worked on fault-adaptive control of fuel transfer systems for aircraft, and advanced life support systems for NASA. He has also initiated new projects in distributed monitoring and diagnosis and prognosis and health management of complex systems. In other research projects, he is involved in developing simulation-based environments for learning and instruction and planning and scheduling algorithms for distributed real-time environments. His research has been supported by funding from NASA, NSF, DARPA, and ONR. He is an associate editor of the *IEEE Transactions on Systems, Man, and Cybernetics, Part A*. He has served on the Program Committee of a number of conferences. He is a senior member of the IEEE Computer Society, ACM, AAAAI, and the Sigma Xi Research Society.

Xenofon Koutsoukos received the Diploma in Electrical and Computer Engineering from the National Technical University of Athens, Athens, Greece, in 1993, M.S. degrees in Electrical Engineering and Applied Mathematics and the Ph.D. degree in electrical engineering from the University of Notre Dame, Notre Dame, IN, in 1998, and 2000, respectively. From 2000 to 2002, he was a member of Research Staff with the Xerox Palo Alto Research Center, Palo Alto, CA, working in the Embedded Collaborative Computing Area. Since 2002, he has been with the Department of Electrical Engineering and Computer Science, Vanderbilt University, Nashville, TN, where he is currently an Associate Professor and a Senior Research Scientist in the Institute for Software Integrated Systems. His research interests include hybrid systems, real-time embedded systems, sensor networks, and cyber-physical systems. He currently serves as Associate Editor for the *ACM Transactions on Sensor Networks* and for *Modeling Simulation Practice and Theory*. He is a senior member of IEEE and a member of ACM. He was the recipient of the National Science Foundation CAREER Award in 2004.