Discrete Event and Hybrid System Simulation with SimEvents

Michael I. Clune and Pieter J. Mosterman
The MathWorks
3 Apple Hill Drive
Natick, Mass. 01760
Michael.Clune@mathworks.com,
Pieter.Mosterman@mathworks.com

Christos G. Cassandras
Center for Information and Systems Engineering
Boston University
Brookline, MA 02446
cgc@bu.edu

***Abstract*— A new simulation product for discrete event and hybrid systems is overviewed and some examples of application areas where it can be used are briefly described.**

I. Introduction

SimEvents [1] has been designed to simulate Discrete Event Systems (DES), but it is embedded in Simulink® [2], a traditional time-driven simulator, so that it is equipped with functionality that enables an effective co-existence of time-driven and event-driven components in complex hybrid systems. In addition, this design allows SimEvents to take advantage of a rich collection of visualization, data processing, and computation tools in both Simulink and MATLAB® while operating as a pure DES simulator if there are no time-driven system components involved. It can also readily incorporate Stateflow®, another useful tool for simulating untimed DES.

II. Architecture

Figure 1 highlights the main functional components of the overall architecture. As a DES simulation engine, SimEvents is driven by an Event Calendar where all future events to occur are listed in ascending order of their scheduled time. SimEvents always processes the first event in this list and updates the DES state accordingly. When such an event takes place, the Cooperative Event Driver is responsible for translating it into a Simulink signal which the Data Exchange module passes on to Simulink so that it may trigger a time-driven process or update various model parameters. Conversely, as a time-driven process evolves under the control of Simulink, it may generate events in the form of level-crossing points (from above, from below, or either) that the Data Exchange module appropriately translates so they may be processed by SimEvents blocks. The most challenging aspect of coordinating time-driven and event-driven dynamics is that of proper timing. In the architecture of Fig. 1, the system "clock" is maintained by Simulink and the Cooperative Event Driver is responsible for ensuring consistency between Simulink blocks and SimEvents blocks which interact with the Event Calendar. Note that when a pure DES is simulated, the only interaction between SimEvents and Simulink is a simple link to the system clock through the Cooperative Event Driver which ensures that the sample times applied are consistent with times in the Event Calendar.

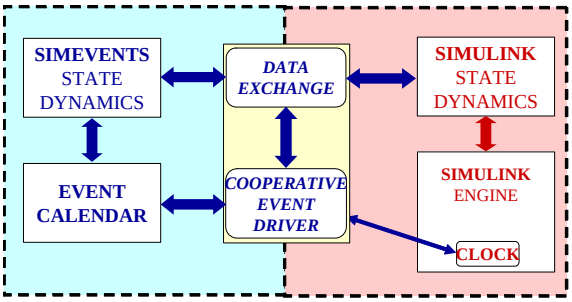

Fig. 1. SimEvents and Simulink collaborative functionality

III. SimEvents Functionality

In Simulink, communication across blocks is based on signals. In SimEvents, it is based on both signals and entities. The "entity" concept is motivated from the view of a DES as an environment consisting of "users" and "resources": users request resources in order to perform various tasks, occupy these resources for a certain amount of time, and then relinquish them so that other users may access them. Examples of users are messages in a communication network and parts in a manufacturing system. Examples of resources are switches in a network and machines in a factory. A typical hybrid system scenario arises when an entity accessing a resource initiates a physical process (thus, defining an event in SimEvents) which is carried out until some termination condition is satisfied (defining another event in Simulink). Based on this approach, SimEvents consists of a number of libraries containing blocks with different system functionalities. The main libraries are the following:

1. **Generators**: Blocks which generate entities, or function calls (i.e., events that call Simulink blocks), or random variates.
2. **Queues**: Blocks where entities can be temporarily stored while waiting to access a resource.
3. **Servers**: Blocks that model various types of resources.
4. **Routing**: Blocks that control the movement of entities as they access queues and servers.
5. **Gates**: Blocks that control the flow of entities by enabling/disabling access of entities to certain blocks.
6. **Event Translation**: Blocks that enable communication between SimEvents and Simulink by translating events into function calls.

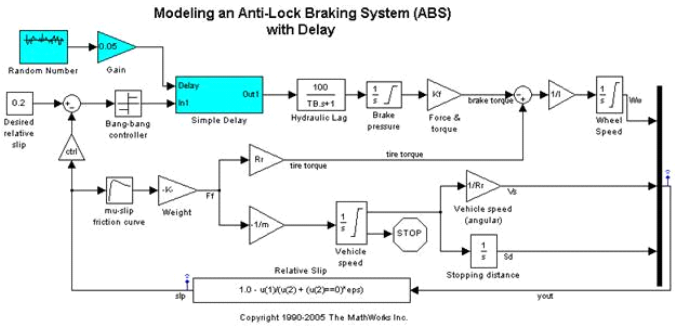

Fig. 2. Simulink model of Antilock Brake System (ABS).

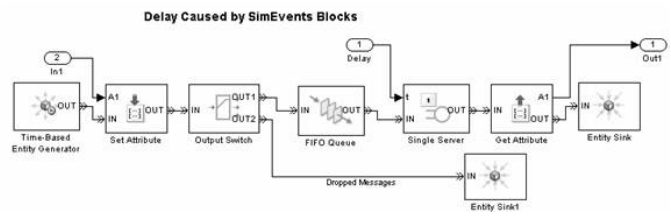

Fig. 3. Subsystem with SimEvents components representing simple delays.

7. **Attributes**: Blocks that assign and modify data to entities. Various control actions are then made based on the values of these data, allowing blocks to differentiate between entities they process.

8. **Subsystems**: These allow a combination of blocks to be executed upon occurrence of specific events (not upon Simulink sample times).

9. **Timers and Counters**: Blocks that measure event occurrence times or time elapsing between events, and blocks that count occurrences of particular event types. These data are supplied to standard display or scope blocks in Simulink or specialized scopes designed specifically for SimEvents.

IV. Some Application Areas for SimEvents

SimEvents has been successfully used to model two aspects of distributed systems: (i) delays in message delivery due to loading of busses and networks and (ii) delays in task execution due to processor loading. While delays can be introduced without SimEvents, it is difficult to accurately associate them with their causes. SimEvents allows modeling of these delays by mapping their causes to random processes used to describe (i) network traffic and congestion or (ii) task generation and execution.

As an example, SimEvents is used to demonstrate the effect that delays have on the control of brakes in an automotive Antilock Brake System (ABS). Figure 2 shows the ABS system including bang-bang control of the hydraulic braking system. The effect of the network delays is represented by the subsystem labeled "Delay" in Fig. 2. This subsystem can represent delays at a variety of levels of fidelity.

1) Delay represented by a simple queueing system. This queueing system can model either model (i) no queueing, (ii) unlimited queueing, or (iii) limited queueing, by setting the capacity of the queue to 0, infinity, or a finite positive value respectively. The subsystem is shown in Fig. 3.
2) Delay represented by traffic on the controller area network bus (CAN bus). The delay is caused by network congestion due to the priority-based arbitration on the CAN bus. The CAN bus subsystem is shown in Fig. 4. This subsystem contains CAN bus nodes for the transmitter and receiver of the controller commands and a node for generating background traffic. This subsystem allows the congestion on the network due to CAN bus arbitration to be modeled. An important difference between the network model and the simple delay model is that the input to the net subsystem is background traffic rate, allowing the performance of the ABS to be demonstrated as a function of traffic loading.

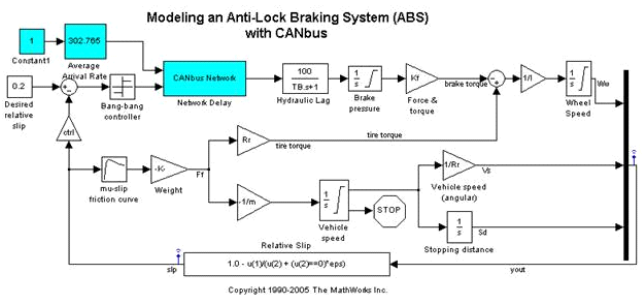

Fig. 4. Subsystem modeling delays due to CANbus network loading.

As shown in this ABS example, SimEvents can be used to effectively model the behavior of complex interactions between time-driven and event-driven components using an intuitive "queue and servers" paradigm.

More information about SimEvents can be found at *http://www.mathworks.com/products/simevents/index.html*. A free trial version can be downloaded from *http://www.mathworks.com/products/simevents/tryit.html*.

References

[1] MathWorks. *SimEvents User's Guide*. The MathWorks, Inc., 2005.
[2] MathWorks. *Simulink: A Program for Simulating Dynamic Systems, User Guide*. The MathWorks, Inc., 2001.