Hybrid Vehicle Simulation System with Discrete-Event Modeling and Simulation

Shafagh Jafer, Jeanette Benjamin
Department of Electrical Computer Systems and Software Engineering
Embry-Riddle Aeronautical University
Daytona Beach, FL
jafers@erau.edu, benjamj2@my.erau.edu

***Abstract*— Embry-Riddle Aeronautical University (ERAU) is one of fifteen schools across the United States and Canada that qualify to compete in the EcoCAR competitions. One of the major tasks of the ERAU EcoCAR team, besides to produce a working vehicle that fits the required hybrid vehicles, is to ensure predictions of the car's performance. In this paper we present the use of the Discrete-Event System Specification (DEVS) formalism to simulate the physical hybrid car in its current state. We have modeled the car and manipulated various inputs and conducted simulation results that in theory match with those of the physical car. The proposed Hybrid Vehicle Simulation System produces scientifically sound results in theory that can be verified and validated. Development is performed at a system level, identifying the major contributing components, connecting those components as it is in the physical car, and simulating the various inputs and outputs of each component. The behavior of each component is analyzed as data is injected into it and the results are documented. Test plans are formulated to test each component (unit) under different conditions. The produced simulation will be incrementally replaced with actual hardware components of the car. This work benefits from a number of modeling and simulation concepts such as: component-oriented model-based development, model-continuity, and incremental replacement.**

Keywords*— *Model-based design; hybrid electric vehicle simulation; discrete-event modeling and simulation; EcoCAR; automobile modeling

I. Introduction

The increasing fuel costs and related economical, political, and environmental concerns have raised the demands for auto-manufacturers to produce more efficient vehicles. Among the different technologies which attempt to improve vehicle's fuel efficiency and reducing emission of greenhouse gasses, the hybrid electric vehicles have been recognized as the technology of choice. A hybrid vehicle is one that stores energy on board in two or more forms. The two sources of energy for a typical hybrid electric vehicle are battery and gasoline. Hybrid electric vehicles reduce fuel consumption in several was. One method is eliminating idle fuel use. In conventional vehicles, fuel keeps burning when the car is in idle mode and not moving. Hybrid electric vehicles eliminate fuel consumption at this stage by immediately shutting the engine off, and re starting it quickly as needed once again. The other method of fuel consumption reduction is recapturing brake energy that otherwise is wasted. This is done by using a bidirectional energy storage device on the vehicle. Another method which is used in plug-in vehicles is to offset fuel with stored grid electricity to allow recharging and some level of electric-only driving capability.

Embry-Riddle Aeronautical University (ERAU) is participating in EcoCAR 2: Plugging in to the Future [2], an advanced vehicle competition run by Argonne National Labs sponsored by General Motors and the United States Department of Energy. ECOCAR2 is a three-year collegiate engineering competition. The goal of this competition is to reduce fuel consumption, reduce well-to-wheel greenhouse gas emissions, and maintain consumer acceptability in the areas of performance, utility, and safety. This competition challenges 15 schools from USA and Canada. EcoCAR vehicle is a 2013 Chevrolet Malibu with hybrid-vehicle technologies capabilities. The competition, running from 2011 to 2014, requires the competing schools to drastically alter the architecture of their vehicle, including removing and replacing the drivetrain components and completely rewriting the vehicle control systems.

With the rise of complex hybrid powertrains and their corresponding complex control systems, model-based design has become the excellent technology in the design and validation of hybrid systems [1]. To fulfill the EcoCAR competition rules and specifications, here we attempt to use model-based design to construct our EcoCAR hybrid-electric vehicle. We demonstrate how discrete-event modeling and simulation techniques can be used as the model-based design approach to effectively design a hybrid electric vehicle powertrain and the corresponding control system. Our modeling and simulation methodology is the Discrete-Event System Specification (DEVS) formalism [7] for dynamic systems. DEVS allows defining an entire system through collaborative and interacting components. A DEVS-based system is composed of coupled (structural) and atomic (behavioral) entities. Atomic components can be viewed as entities communicating with each other by sending/receiving messages via ports. The behavior of each atomic component is given by a state machine. DEVS promotes component-based model-driven development. It supports model continuity where an entire system can be built upon existing components. With DEVS incremental replacement capability, simulation blocks are replaced by actual hardware, providing an excellent solution to modeling and simulating complex and risky systems.

In this work, using DEVS theory, the overall vehicle architecture is modeled and simulated in an incremental development approach, focusing on a single component at a time. Software-in-the-loop and hardware-in-the-loop simulations is integrated with our proposed model in the future to examine fuel efficiency and validate the vehicle control system for performance, safety, and fuel economy.

II. BACKGROUND

For the EcoCAR 2 competition, ERAU is building a series plugin hybrid electric vehicle (PHEV). This architecture was chosen by the EcoEagles team for its versatility and high simulated efficiently. The vehicle uses a Remy [3] HVH250-090-P [4] electric motor to drive the vehicle's front wheels. The motor will be powered by an A123 (lithium-ion battery) [5] donated energy storage system (ESS) running at a nominal 292V. When the battery is depleted, a General Motors [6] donated 1.7L diesel engine coupled to a second Remy motor will act as a generator to supply power to the battery and traction motor. This architecture allows the vehicle to have an estimated all-electric range of approximately 40 miles, with an overall range of over 350 miles. A high-level overview of the vehicle architecture is shown in Figure 1.

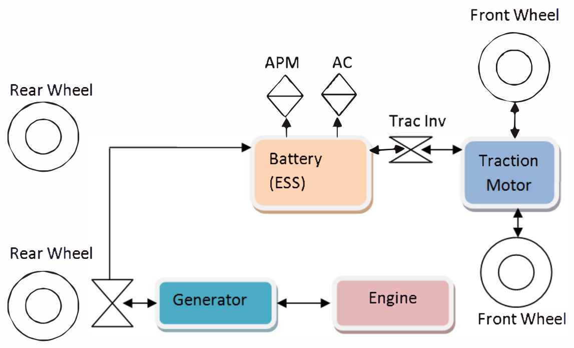


Figure 1. Vehicle architecture.

We are interested in modeling the components outlined in Figure 1, given their following descriptions:

- *ESS – Energy Storage System* – is the battery pack built by the EcoCAR students, with 16.2 kWh capacity. It is a high voltage energy.
- *Generator* – Converts mechanical energy to electrical energy in order to charge the ESS from the diesel.
- *Traction Motor* – Converts electrical energy to mechanical energy. It is attached to the E-Transmission that powers the wheel.
- *Engine* – 1.7 Liter turbo Diesel that runs on B20 Bio diesel. The engine is not connected to the wheels, its sole purpose is to generate charge for the ESS when the ESS charge depletes to 30%. When this occurs the diesel will start to maintain charge levels in the ESS.
- *APM* – is the Auxiliary Power Module. Its main purpose is to convert 300 Volts to 12 Volts to maintain the charge on the battery since there is no alternator.
- *Inverters* – There are two inverters: a Traction Motor inverter that converts DC to AC and vice versa and a Generator inverter that converts AC to DC.

Our model-driven design approach is to take one component at a time and model it based on the specifications for that component and the appropriate mathematical equation(s) embedded. All of the necessary inputs for the component will be itemized, utilizing DEVS-based components. DEVS uses state machines to isolate the different states that the component can be in and simulates its behavior. The embedded mathematical equations will aid in producing output that can then be verified and validated. Each component is built using the same framework and methodology. Once all components have been modeled, test plans will be formulated for each component. Then, the next step would be combining all of the components into the system level and formulate test plans for the system as a whole, then conducting integrated and system tests for the purpose of verifying and validating the overall system behavior and performance.

III. RELATED WORK ON MODEL-BASED DESIGN

With the increased demand and complexity of hybrid vehicles, model-based design has gained a lot of attention from automobile industry and related researchers. Traditionally, model-based design was only used for developing controllers. Recently, there has been noticeable effort among researchers demonstrating how model-based techniques can be used throughout the design process as well. The key focus has been on highlighting the ability of model-based design techniques in continually verifying that design requirements are being met at each step throughout the process. In [1] a model-based design of a hybrid vehicle is given using the Mathworks SimDriveLine and SimPowerSystems features. They also used Mathworks Stateflow for developing the control strategy of the vehicle based on state machine. Universal modeling languages such as SysML and UML were used in [9] in developing control architecture and strategy for hybrid electric vehicle. Mathworks Simulink was then used to implement the actual control strategy. Pisu and Rizzoni [10] compared four supervisory control techniques for hybrid electric vehicles. They techniques presented were Finite-State Machine (FSM), Equivalent Consumption Minimization Strategy (ECMS), H_∞ Control, and Dynamic Programming. Using each of the methods, first the vehicle model used for simulating the control strategies was outlined. Then mathematical models for all the vehicle components were then presented. And finally a curve fit model was used for engine fuel consumption. Unfortunately those methods are not practical for in-vehicle use due to extreme offline computational complexity and prior knowledge of the drive cycle in developing the optimal solution. A number of efforts [11][12] used hardware-in-the-loop (HIL) to simulate the vehicle's control system. The HIL system provides a risk-free and cost-effective experimental environment, as errors are found earlier in development, reducing correction cost. Such testing strategy provides progression from desktop to HIL, dynamometer, and finally to on-road testing. Each step included incremental replacement of simulated models with real components.

Designed for generic dynamic systems, DEVS is a powerful technology for incremental model-based designs and development. The work presented here is the first attempt to using discrete-event continuous-time system theory for the full design of a hybrid vehicle.

IV. VEHICLE COMPONENTS MODELING AND SIMULATION WITH DEVS

Among the existing modeling and simulation techniques, the DEVS (Discrete Event System Specification) formalism [1] is regarded as one of the most developed general-purpose M&S frameworks for Discrete Event Dynamic Systems (DEDS). In DEVS, a real system is decomposed into behavioral (atomic) and structural (coupled) components. The system under study is modeled as a top coupled component encapsulating atomic or other internal coupled components. Components are linked through their input/output ports and interact with each other by sending/receiving event messages. Events can arrive at any time through input ports, but due to the discrete-event nature of DEVS, acceptable data can only be processed in a discrete fashion. The behavior of an atomic component is given by a state machine. Through their life time, atomic entities go through various states when transitions are triggered by incoming events.

Component-based DEVS provided us with a step-by-step design approach. First the system's overall architecture is defined within the DEVS framework. Then the internal behavior of each component is defined using a state machine. Fig. 2 illustrates the DEVS-based high-level System Architecture of our hybrid vehicle composed of four coupled components (*Generator*, *Traction Motor*, *Engine*, and *Battery*) and four atomic entities (Inverter - *inv*, *APM*, *AC*, Traction Inverter – *Trac inv*.).

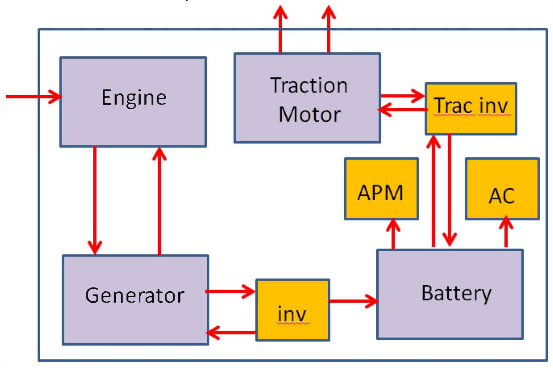


Fig. 2. Hybrid vehicle DEVS-based system architecture.

A. Energy Storage System

The Energy Storage System (ESS) is an electrical lithium-ion battery system that produces power to power the PHEV. The ESS is comprised of two major components: Battery and a battery management system. As defined in Figure 3, it is composed of two internal components.

- The **Battery** (an atomic component) is the slave to the entire system, where its main function is to do what the battery management system tells it to do.
- The **Battery Management System** (a coupled component) is considered the brain of the battery because it monitors, controls and connected to the micro-autobox of the car via the Controller Area Network (CAN bus).

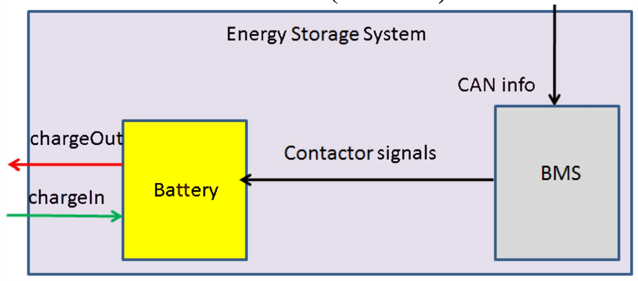


Fig. 3: Energy Storage System.

The internal behavior of the Battery component is defined using the state diagram in Figure 4. Here we present the details for the Battery component. EcoCAR is a plug-in hybrid electric vehicle (PHEV) which provides battery recharge capability to restore to full charge by connecting a plug to an external electric power source. As shown in Figure 2, our PHEV shares the characteristics of both a conventional hybrid electric vehicle, having an electric motor and an internal combustion engine, and of an all-electric vehicle with a plug to connect to the electrical grid. We model the Battery component as a coupled DEVS system with an input indicating charge arriving, and an output representing charge departure.

The abstract architecture of the Battery component is given in Fig. 4.

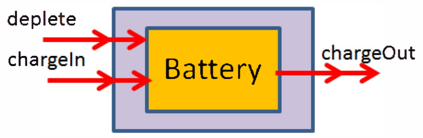


Fig. 4. Battery coupled component.

The formal DEVS specifications of the Battery atomic component are given as following:

$$\textbf{Battery} = <\mathbf{X}, \mathbf{Y}, \mathbf{S}, \delta_{int}, \delta_{ext}, \lambda, \mathbf{ta}>$$

Where **X** is the set of inputs, **Y** is the set of outputs, **S** defines possible states, δ_{int} is the internal transition function (defines the behavior upon state change), δ_{ext} is the external transition function (defines the behavior upon arrival of input), λ describes the output generation, and **ta** provides the states durations. According to the definitions and terminologies of PHEV, the vehicle's battery can be in one of the following modes of operation:

- **Charge-depleting (CD) mode**: An operating mode in which the energy storage state-of-charge (SOC) may fluctuate but, on average, decreases while the vehicle is driven.
- **Charge-sustaining (CS) mode**: An operating mode in which the energy storage SOC may fluctuate but, on average, is maintained at a certain level while the vehicle is driven. This is the common operating mode of commercial HEVs.

- **All-electric range (AER) mode**: The vehicle is driven with motor only (with the combustion engine off), range is the total miles driven electrically before the engine turns on for the first time.
- **Blended or charge-depleting hybrid (CDH) mode**: An operating mode in which the energy storage SOC decreases, on average, while the vehicle is driven; the engine is used occasionally to support power requests.
- **Zero-emission vehicle (ZEV) range**: The same as AER; there are no tailpipe emissions when the vehicle is in electric vehicle mode.

These modes define the internal behavior of the Battery component. Thus, the state diagram is given based on two generic states (CD and CS) as follows:

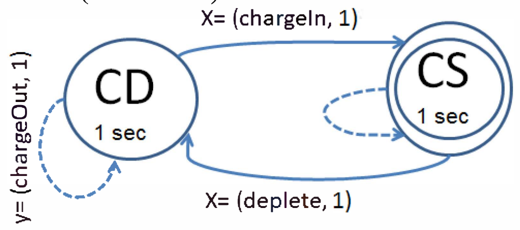


Fig. 5. Battery state diagram.

The state diagram is interpreted as following: the Battery model is initially at state CS (initial state is denoted by double border) indicating charge sustaining. Given a state duration of 1 second for CS state, this means that every one second the state of the Battery is reset to CS allowing for periodic increment of the charge (this can be modeled using a state variable that is incremented by one every time the Battery enters CS state). However, this behavior can be interrupted when an input arrives through the *deplete* port indicating usage of the Battery and thus causing state change to CD. As the Battery enters the CD state, periodic charge depletion will occur, indicated by sending an output event to the destination component (e.g. AC, APM, or the Traction Inverter). The depletion phase will continue by depleting an amount of charge every one second, until the battery receives charge from a source (engine or through plugging-in). This is when an event arrives through the *chargeIn* port, causing an immediate state transition to the CS state.

With the Battery structural diagram from Fig. 4 and the state diagram above, the Battery formal DEVS specification is given below.

```
Battery = <X, Y, S, δint, δext, λ, ta>
X = {(chargeIn, 1), (deplete, 1)}
Y = {(chargeOut, 1)}
S = {CD, CS}
δint (S -> S'){
        if (S = CD) -> S = CD
        if (S = CS) -> S = CS
        }
λ (S -> Y) {
        if (S = CD)  → out(chargeOut, 1)
        }
δext (S,e,X -> S'){
        if (S == CD && e = anytime && X == (chargeIn,1))  → S = CS
        if (S == CS && e = anytime && X == (deplete,1))   → S = CD
        }
ta (S -> R+) { CD → 1,  CS → 1}
```

Fig. 6. Battery formal DEVS specification.

B. Battery Management System

The Battery Management System (BMS) contains two major components: a battery control unit (bcu) and a battery monitoring unit (bmu). The overall responsibility of the BMS is to protect the battery from unsafe temperature, voltage and current, and to monitor the status of the battery itself such as its temperature, calculate state of charge (SOC) and read CAN incoming CAN messages and send outgoing CAN messages. Figure 7 shows the conceptual diagram of the BMS. The BMS receives information through CAN bus, Read sensors and Contactor Signals. Contactor signals are the only means the battery can protect itself in case of unsafe parameters. The contactors are controlled by the battery control unit (bcu). In the event that the battery monitoring unit (bmu) within the bmu receives information from the CAN bus that indicates an error or unsafe parameters, it then sends a signal to the bcu. Then, the bcu sends one of two possible signals: close contactors or open contactors. In the event of key ignition the signal will close contactors provided that all operational parameters are safe and then charge will flow in or out of the battery. In the event of unsafe parameters the signal will open contactors which will break the circuit connection and stop charge flow in or out of the battery.

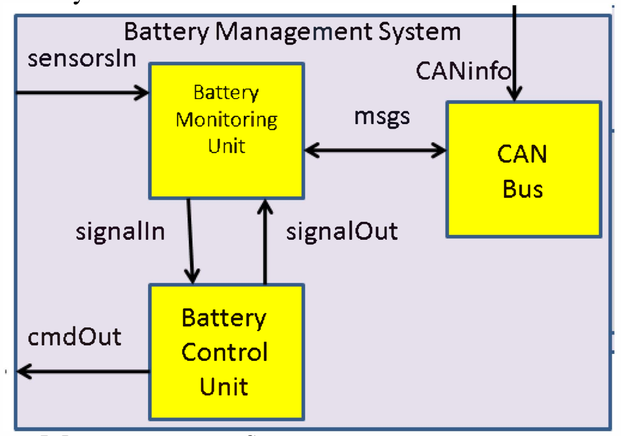


Figure 7. Battery Management System.

The internal behavior of bmu and bcu is defined next using the state diagram in Figure 8 and Figure 9.

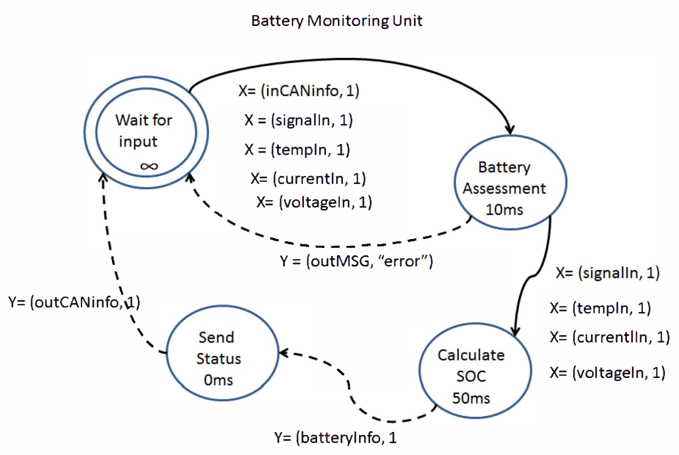


Figure 8. bmu state diagram.

The Battery Monitoring Unit (bmu) has 4 states: the initial state is "Wait for Input", "Battery Assessment", "Calculate SOC" and "Send Status". The initial state has duration of (∞) and it will remain it this state unless it receives the input: X = (inCANinfo, 1), X = (signalIn, 1), X = (tempIn, 1), X = (currentIn, 1), X = (voltageIn, 1), where the state changes to "Battery Assessment". The "Battery Assessment" has duration of 10ms where it can produce an error message if the anomalies are detected. If no anomalies were detected the state changes to "Calculate SOC" which has a duration of 50ms. In this state the current state of charge (SOC) of the battery is calculated in percentages and an output of Y = (batteryInfo,1) which will include the calculated SOC, voltage, current, and temperature of the battery and after 50ms the state changes to "Send Status". The "Send Status" produces an output of Y= (outCANinfo,1) which is the battery information that will be transmitted over Controller Area Network (CAN). For duration of 0 ms and changes state to the initial state of "Wait for Info".

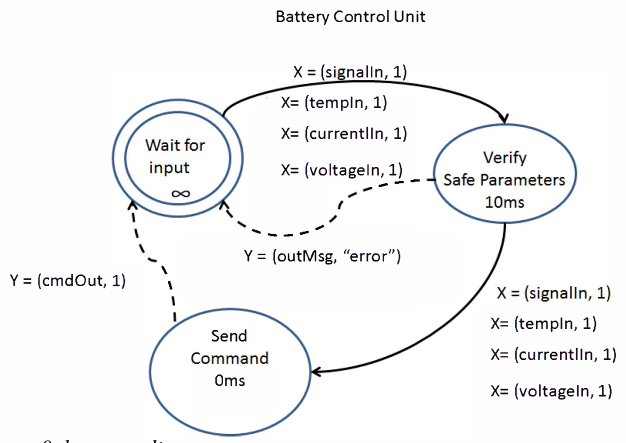


Figure 9. bcu state diagram.

The Battery Control Unit state diagram has three states: the initial state of "Wait for Input", "Verify Safe Parameters", and "Send Command". The initial state has duration of (∞) since the state will not change unless it receives the required input as shown in the state diagram. When the bcu receives the input, it changes its state to "Verify Safe Parameters" for duration of 10ms. In this state the battery runs comparisons of the data being received from the battery sensors against the safe ranges specified for the battery. If the comparison shows the parameters are unsafe then an "error" message is sent out as an internal output Y= outMsg. After 10ms "Verify Safe Parameters" state changes to "Send Command" which has duration of 0ms. In this state a command is sent out as output: Y = (cmdOut, 0 /1) and returns to the initial state "Wait for Input". The command can be one of two possible commands: open contactors (0) or close contactors (1). In an unsafe situation an "open contactor" signal will be sent; however in other cases where the parameters were safe and charge wants to enter the battery then the command sent out would be close contactors".

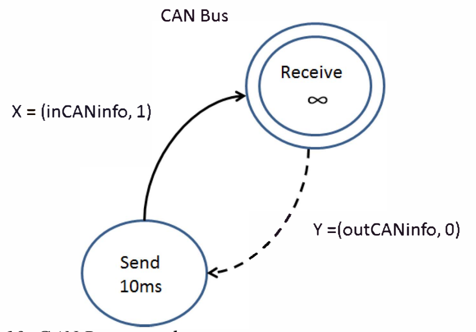


Figure 10. CAN Bus state diagram.

The CAN bus state diagram (Figure 10) depicts two basic states: "Receive" and "Send". The "Receive" state is the initial state which has duration of (∞) because it will not change unless is receives an input: X= inCANinfo. Once inCANinfo is received the CAN bus changes state to "Send" which has a duration of 0ms. In the "Send" state the battery transmits all of the information out to the micro-autobox of the car as the output: Y = (outCANinfo, 1) which then sends the relevant information to various components of the car that requires the information for operation. The CAN bus acts like a client server operation so it changes state based on demand and priority.

A. Traction Motor

The Traction Motor is a three-phase motor that receives input from the traction inverter. The Traction Motor produces torque and speed based on the input commands that are received from the micro-autobox of the car that is received by the inverter and passed to the motor. The traction motor converts AC to DC and vice versa and it contains a temperature sensor for the motor. The Traction Motor is comprised of a Temperature Sensor and a Motor. The traction Motor internal structure is presented in Figure 11.

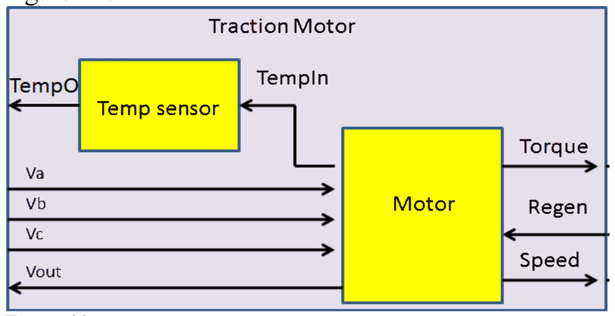


Figure 11. Traction Motor.

The internal behavior of the Motor and Temperature Sensor is presented in Figure 12 and Figure 13.

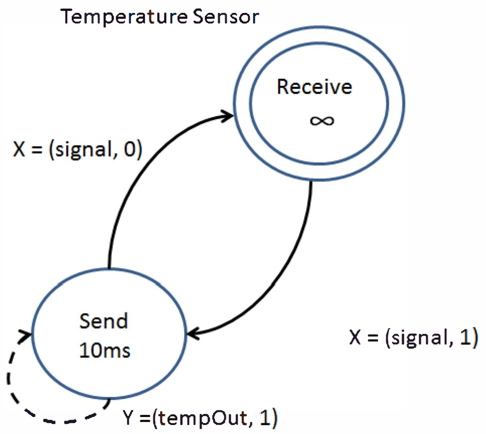


Figure 12. Temperature Sensor state diagram.

The Temperature Sensor state diagram has 2 states: the initial state is "Receive" and the other state is "Send". The "Receive" state has duration of (∞) and the "Send" state has duration of 10ms. The behavior of the Temperature Sensor is simple just as the CAN Bus, in that it stays in receive state unless it receives an input: X = (signal, 1) which changes the state to "Send". The "Send" state will send the acquired temperature sensor data as an output, Y = (tempOut, 1) and after 10ms it changes the state to its initial state of "Receive".

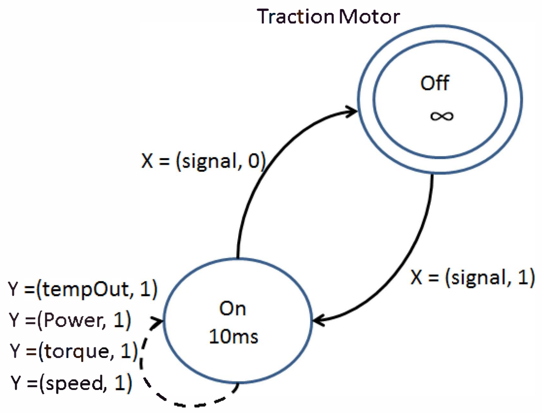


Figure 13. (Traction) Motor state diagram.

The Traction Motor state diagram has 3 states: The initial state is "Wait for Input", "Active" and "Handle Event". When the initial state which remains in duration of (∞) unless it receives an input: X = (signal, 1) which changes the state to "Active". The "Active" has duration for 10ms and produces an output of Y = tempOut which is received from the temperature sensor. In this state the Traction Motor may receive several requests which when received changes its state to "Handle Event". The inputs are: X = torque; X = speed; which are commands being received. Additionally an input of X= regen may occur if regenerative braking is achieved. In this state the traction motor will execute the request, for example: if the request was "torque" which is torque command that was received from the micro-autobox, then the "Handle Event" state will set the commanded torque, then the state is changed after 0ms to state, "Active". The output generated in "Handle Event" are Y = torque; Y = speed and if there is regenerative braking an output of Y = V_{out}. The state can remain in "Active" until an input of X = (signal, 0) is received which changes the state to the initial state.

V. Experiments

In order to execute our DEVS simulations, we implement our models using a DEVS development environment called CD++ [13]. The CD++ toolkit provides an easy-to-use framework to implement DEVS models in C++. In this paper we only present the simulation results from executing the Battery atomic component.

First a model file ("Battery.MA") is written defining the structure of the Battery coupled component in terms of encapsulated atomic entities and the linkage among them as well as input/outputs to the external environment (which could be another coupled or atomic component, e.g. Inverter or AC). The Battery Model file is defined as follows:

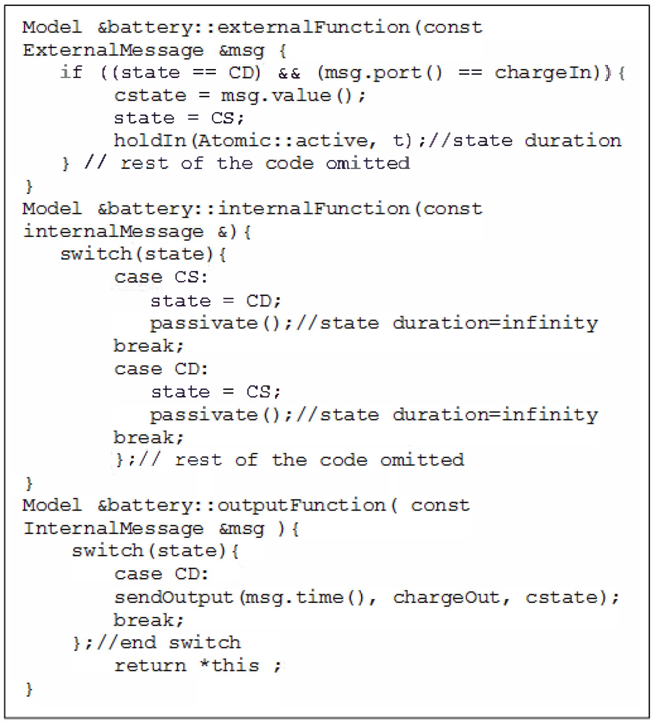
```
[top]
components : battery@Battery
out   : chargeOut
in    : chargeIn
Link : chargeIn chargeIn@battery
Link : deplete deplete@battery
Link : chargeOut@battery chargeOut
```

Next, a ".cpp" and a corresponding ".h" file are created for every atomic component to implement its behavior. The CD++ tool provides a template for these two files, making it a very simple and quick task. The ".h" file includes definition of that atomic component's states, the time duration for each state, and declarations for the four core DEVS methods: initial, internal transition, external transition, and output function. A DEVS developer provides the implementation for these four functions by mapping the DEVS atomic definition and behavior (as defined on the state diagram) to source code.

The "Battery.h" and "Battery.cpp" excerpts are shown in Figure 14 and Figure 15.

```
class battery : public Atomic{
...
protected:
 Model &initFunction();
 Model &externalFunction(const ExternalMessage &);
 Model &internalFunction(const InternalMessage &);
 Model &outputFunction( const InternalMessage  &);
private:
        const Port &chargeIn;  //input ports
        const Port &deplete;
        const Port &contactorSignal;
        Port &chargeOut;        //output ports
        Port &batteryInfo;
      enum State{ CD, CS }; //Battery states
        State state;
};      // class Battery
```

Fig. 14. Excerpts of "Battery.h".



```
Model &battery::externalFunction(const
ExternalMessage &msg {
   if ((state == CD) && (msg.port() == chargeIn)){
        cstate = msg.value();
        state = CS;
        holdIn(Atomic::active, t);//state duration
   } // rest of the code omitted
}
Model &battery::internalFunction(const
internalMessage &){
   switch(state){
        case CS:
           state = CD;
           passivate();//state duration=infinity
        break;
        case CD:
           state = CS;
           passivate();//state duration=infinity
        break;
        };// rest of the code omitted
}
Model &battery::outputFunction( const
InternalMessage &msg ){
    switch(state){
        case CD:
        sendOutput(msg.time(), chargeOut, cstate);
        break;
    };//end switch
        return *this ;
}
```

Fig. 15. Excerpts of "Battery.cpp"

DEVS unit testing can be performed by analyzing the behavior of each atomic unit separately. For this purpose, an atomic component is injected with various inputs and timing sets, and the corresponding outputs and their timing are verified against the model specification as outlined on the state diagram. In order to test the Battery component, first an event file is created specifying the value, the port, and the time at which an input event is injected into the component. Table 1 illustrates the content of the event file (".ev"). The event file has the following format:

TABLE 1. "IN.EV" FILE FOR TESTING THE BATTERY UNIT.

eventTime	**port_name**	**eventValue**
00:00:50	chargeIn	1

Given this input, according to the Battery state diagram the following scenario should occur: assume the Battery is in state CD indicating that there will be a periodic loss of charge at a rate of *n_volts* per *second*. While in *charge-depleting* state, if a charge event arrives, the model immediately transits to *charge-sustaining* state where its charge will be incremented at a rate of *n_volts* per *second*. Since no output is generated while in state *CS*, we should expect nothing to be written to the ".out" file, indicating no output generation.

Once all atomic units are tested separately, their corresponding coupled components are then tested (incremental integrated testing). By grouping integrated tests, finally an overall system integrated test can be conducted verifying that the system works as a whole.

VI. Conclusion

This paper discussed a new model-driven design, simulation, and modeling technique developed at Embry-Riddle Aeronautical University for the EcoCAR 2 project. We presented an overall design of a hybrid-electric vehicle using the discrete-event dynamic system (DEVS theory). Given the component-oriented and highly-reusable capabilities of DEVS, the components defined in this work can be used "as-is" or in an extended version for designing and simulating any type of vehicle (electric, hybrid electric, hybrid, fuel-burning, etc.). Since DEVS is a mathematical formalism, it is backboned with guaranteed correctness and reliability. Any system implemented using DEVS is easily scalable, verifiable, and testable. This work attempts to provide a complete and freely-available DEVS-based package for modeling and simulation of vehicles.

Reference

[1] Mahapatra, S., Egel, T., Hassan, R., Shenoy, R., and Carone, M. "Model-Based Design for Hybrid Electric Vehicle Systems". SAE 2008 International World Congress, Detroit, MI. 2008.

[2] Argonne National Labs, "EcoCAR 2," [Online]. Available: http://www.ecocar2.org/ecocarchallenge.

[3] Remy, "Remy - Company Overview," [Online]. Available: www.remyinc.com/AboutOverview.aspx.

[4] Remy, "HVH250-090P," [Online]. Available: http://www.remyinc.com/docs/HVH250_r3_Sept_2010.pdf.

[5] A123, "About A123," [Online]. Available: http://www.a123systems.com/about-us.htm.

[6] General Motors, "About GM," [Online]. Available: http://www.gm.com/company/aboutGM.html.

[7] Zeigler, B., Praehofer, P. H., and Kim, T. G. "Theory of Modeling and Simulation". Academic Press. 2000.

[8] Smith, B, Raj, L., Wong, G., Stansbury, R. "Intelligent control system for improving the efficiency of a series hybrid for the EcoCAR 2 challenge". IEEE SoutheastCon. 2012.

[9] Marco, J, Cacciatori, E. "The Use of Model Based Design Techniques in the Design of Hybrid Electric Vehicles". 3rd Institution of Engineering and Technology Conference on Automotive Electronics, June 28-29, Warwick, UK. 2007.

[10]Pisu, P., Rizzoni, G. "A Comparative Study Of Supervisory Control Strategies for Hybrid Electric Vehicles", IEEE Transactions on Control Systems Technology, 15(3). 2007.

[11]Ramaswamy, D., et. al."A Case Study in Hardware-In-the-Loop Testing: Development of an ECU for a Hybrid Electric Vehicle", SAE Paper 2004-01-0303, SAE 2004 International World Congress, March 8-11, Detroit, MI. 2004.

[12]Deng, Y., Li, H., Foo, S. "Controller Hardware-In-the-Loop Simulation for Design of Power Management Strategies for Fuel Cell Vehicle with Energy Storage", Vehicle Power and Propulsion Conference, September 7-10, Dearborn, MI. 2009.

[13]G. Wainer, "CD++: A Toolkit to Develop DEVS Models", Software – Practice and Experience, 32(13), pp. 1261-1306, 2002.