Proceedings of the 2006 Winter Simulation Conference
L. F. Perrone, F. P. Wieland, J. Liu, B. G. Lawson, D. M. Nicol, and R. M. Fujimoto, eds.

A SIMULATION ALGORITHM FOR DYNAMIC STRUCTURE DEVS MODELING

Hui Shang
Gabriel Wainer

Dept. of Systems and Computer Engineering
Carleton University
1125 Colonel By Drive Ottawa, ON
K1S 5B6 CANADA

ABSTRACT

Real-Time System (RTS) correctness and timeliness are critical. Modeling and Simulation techniques have been widely used for testing particular conditions on these systems. Recently, the DEVS formalism has been successfully used as a framework for RTS validation. Nevertheless, we need to address dynamic adaptation to dynamic changes in the environment. Dynamic Structure DEVS focuses on the possibility to change system structure dynamically according to the system real requirements, which is useful for RTS (in which sometimes it is impossible to interfere with the running of the system, and auto-adaptation is needed). We present a new algorithm derived from the DSDE and the dynDEVS formalisms. We use the DSDE formal specifications, and parts of the dynDEVS simulation algorithms.

1 INTRODUCTION

Hard Real-Time Systems (RTS) are highly reactive artificial systems that deliver data from/to devices interacting with the surrounding environment (another artificial/natural system) within deadlines ranging at millisecond scales. As the decisions taken by these applications can lead to catastrophic consequences for assets or lives, correctness and timeliness are critical. Modeling and Simulation (M&S) techniques, instead, have proven to be adequate for testing particular conditions, regardless of the application's size. M&S is also an alternative method of analysis for natural systems, which is convenient to study the environment the RTS is controlling.

Recently, the DEVS formalism (Zeigler et al. 2000) has been used as a framework for RTS validation (Cho et al. 2000; Cho et al. 2001; Hong et al. 1997; Kim et al. 2001). The DEVS simulation provides a good framework for these purposes, because it is a mathematical paradigm with well-defined concepts of coupling of components, hierarchical, modular model construction, support for discrete event approximation of continuous systems and an object-oriented substrate supporting repository reuse. Real-Time DEVS (Hong et al. 1997) helps to expand each model of the system for executing in a real-time environment. DEVS has also been implemented to execute over CORBA to address the requirements of scalable and efficient model execution (Cho et al. 2000). Nevertheless, we need to address the dynamic adaptation to dynamic changes in the environment.

Dynamic Structure DEVS (Barros 1995; Barros 1997; Barros 1998; Uhrmacher 2001; Uhrmacher 2004) allows addressing some of these issues. Dynamic structure systems focus on the possibility to dynamically change the system structure according to the system real requirements, which is useful for real time systems (in which sometimes it is impossible to interfere with the running of the system manually, and auto adaptation is needed). In this way, the system can adapt internal/external environments automatically.

Dynamic structure is one of the measurements to improve the flexibility and reliability of systems. By detecting and revising the current states and the layouts of involved models, a more reasonable structural organization of the system can be achieved automatically. The Dynamic structure algorithms based on DEVS support the structural changes to full extent, ranging from simple model/connection add/deletion to the exchange of models between coupled models. The structural changes can be divided into three levels:

1. System level: The structural change happens between coupled models (i.e., a new link between coupled models is added);
2. Component level: The structural change happens within a coupled model but including two or more atomic models;
3. Sub-component level: The structural change only happens within a single atomic model.

We will present a new approach based on the previous dynamic structure algorithms. We will show that, when compared with the existing algorithms, our approach is context-oriented and more applicable to real-time simulations.

2 THE DEVS FORMALISM

DEVS is a formal modeling and simulation framework based on system theory. DEVS has well-defined concepts for coupling of components and hierarchical, modular model composition. DEVS defines a complex model as a composite of basic components (called atomic model), which can be hierarchically integrated into coupled models. A DEVS atomic model is defined as M = <X, S, Y, $\delta_{int,}$ δ_{ext}, λ, ta>, where X is a set of input events of the atomic model; Y is a set of output events of the atomic model; S is a set of partial states associated with the atomic model; ta represents the lifetime of each state in S; δ_{ext} is the external transition function which is triggered when an input event in X is received; λ is the output function; and δ_{int} is the internal transition function. If there is no external event comes, the current state will keep for its lifetime ta. Then, the output event is determined by λ and produce output events Y; at the same time, the internal state change will happen determined by the internal transition function.

A DEVS coupled model is defined as CM = <X, Y, D, $\{M_i\}$, $\{I_i\}$, $\{Z_{ij}\}$>, where M_i ($i \in D$) is a set of basic DEVS models (atomic or coupled) interacting through their interface (X, Y) ; I_i is the set of influencees of model i; for each $j \in I_i$, Z_{ij} is the i to j translation function to convert the output of M_i to the input of M_j. Due to closure under coupling, coupled model can be taken equally as atomic model, which enables model reuse.

The DEVS models are executed by abstract simulators that are independent from the models themselves. Consequently, separated concerns between models and implementations of simulation can be achieved and enhance the verification of each layer independently.

DEVS is a popular method to simulate a variety of systems. However, the regular DEVS simulation is difficult to adapt to dynamically changed systems. The Dynamic structure algorithm in DEVS simulation is an optional solution for real world simulation. The two most popular dynamic DEVS structure algorithms are DSDE (Barros 1997) and dynDEVS (Uhrmacher 2001).

DSDE divides models into two groups: basic and network models. The basic models are atomic structure units which cannot be split. The network models are coupled components, composed of multiple basic structure models and interconnections that involve structural changes. A Network Executive is a modified basic model to conduct structural changes in network models. The Network Executive stores all possible states of structural changes and their corresponding component sets in each structural state. The two parts are associated together through an index function in the Network Executive. A DSDE network is a 4-tuple $DSDEN_N = (X_N, Y_N, \chi, M_\chi)$, where X_N is the network input value set; Y_N is the network output value set; χ is the name of the dynamic Network Executive; and M_χ is the model of the Network Executive χ, which is a modified basic model and is defined by $M_\chi = (X_\chi, s_{0,\chi}, S_\chi, Y_\chi, \gamma, \Sigma^*, \delta_\chi, \lambda_\chi, \tau_\chi)$. Here, $\gamma: S_\chi \rightarrow \Sigma^*$ is the structure function, and Σ^* is the set of network structures. A structure $\Sigma_\alpha \in \Sigma^*$ associated with the executive partial state $s_{\alpha,\chi} \in S_\chi$ is given by $\Sigma_\alpha = \gamma(s_{\alpha,\chi}) = (D_\alpha\ \{M_{i,\alpha}\}, \{I_{i,\alpha}\}, \{Z_{i,\alpha}\})$, where D_α is the set of component names associated with the executive partial state $s_{\alpha,\chi}$; for all $i \in D_{\alpha,}$ $M_{i,\alpha}$ is the model of the component i; for all $i \in D_\alpha \cup \{\chi, N\}$, $Z_{i,\alpha}$ is the set of component influencers of i; for all $i \in D_\alpha \cup \{\chi\}$, $Z_{i,\alpha}$ is the input function of the component i; and $Z_{N,\alpha}$ is the network output function. Changes of a basic model include structural changes within the basic model or changes on transition/output functions of this basic model. A Network Executive should be used together with the basic model to form a network model (only the Network Executives can conduct structural changes). In (Barros 1998), three abstract simulators are defined:

1. *Simulators are able to use the implicit behavior contained in basic models;*
2. *Network Simulators can simulate dynamic structure network models;*
3. *The Synchronizer manages the simulation time.*

The dynDEVS formalism (Uhrmacher 2001) does not introduce an extra component to conduct dynamic structural changes. Instead, ρ_α, a model transition function, is included. There are two kinds of dynamic DEVS models: dynDEVS (atomic) and dynNDEVS (coupled). The dynDEVS models are atomic structural components with the structure *dynDEVS =df <X*, *Y*, *minit*, M(*minit*)>, where *X*, *Y* are the structured sets of inputs and outputs; *minit* ∈ M(*minit*) is the initial model, where M(*minit*) is the least set having the structure{<*S*, *sinit*, δ*ext*, δ*int*, ρα, λ, *ta*> }. dynNDEVS models are coupled structural components with the structure *dynNDEVS = df <X*, *Y*, *ninit*, N(*ninit*)>, where *X*, *Y* are the structured sets of inputs and outputs; *ninit* ∈ N(*ninit*) is the start configuration and N(*ninit*) is the least set having the structure {<*D*, ρ_N, {*dynDEVSi*}, {*I}*, {*Zi*, *j*}, *Select*>}. A model's state space, internal and external transition, output, time advance, and model transition functions are subject to change during simulation. A dynDEVS can be interpreted as a set of DEVS models with the same interface plus a transition function that determines which DEVS model succeeds the previous one. Agents associated with dynDEVS or dynNDEVS models hold the worldview knowledge of their corresponding models and environments. Agents are responsible for launching structural changes and conducting the changing process.

Both of the above formalisms introduce new structure transition functions to conduct structural changes. In DSDE, the structural changes are carried out by χ (the Network Executive) and the structure function γ (which maps the network structure state's set S_χ and the network structure models' set Σ^*) is applied in the Network Executive. The centralized Network Executives make sure that the structure transition is executed sequentially without any conflicts between structural change functions of the models. In dynDEVS, agents associated with the models conduct structural changes. ρ_α and ρ_N are structure transition functions in dynDEVS and dynNDEVS models respectively which execute structural changes concurrently and independently. However, executing ρ_α and ρ_N of different models may cause conflicts between models due to different worldviews of the models. To avoid conflicts, "omnipotent" models are used to conduct structural changes sequentially and once per simulation step. Constraints in certain functions are applied to avoid conflicts (i.e., not allowing a model to delete other models, nor adding new models into other coupled models, etc.). These constraints make the structural transition functions more complicated.

3 FLEXIBLE DYNDEVS SIMULATOR

In order to integrate the dynamic DEVS simulation into the regular DEVS simulation algorithm, we inherit the message-driven mechanism, which is applied in the regular DEVS simulation algorithm. Besides the two main message types (@, t) and (*, t) used in the regular DEVS simulation algorithm, extra message types are introduced to facilitate dynamic DEVS simulation, as we explain in this section.

Our proposal stems from both DSDE and dynDEVS algorithms. We apply the DSDE formal specifications and parts of the dynDEVS simulation algorithm. In DSDE, a Network Executive conducts the dynamic structural changes. We follow the same idea to provide ground for user-defined model design and simulation (state transition functions, structural transition functions and output functions). However, we do not attach a Network Executive to a network model. A different mechanism is devised to launch dynamic structural changes, and to link regular state transitions of models and structural transitions of some models in simulation processes. First, we introduce message types used in the regular DEVS simulation algorithm and the dynamic algorithm respectively; second, we define how to launch a dynamic structural change within a regular state transition message from which the dynamic structural changes may raise; third, we identify the steps within a single structural change, and what its associated message types are (Uhrmacher et al. 2004); and finally, we present the proposed abstract simulation algorithm.

Message types in the regular DEVS simulation algorithm:

1. (@, t): A collecting message for collecting outputs of each component and routing the outputs to their corresponding input ports according to the links between models;
2. (*, t): An internal message that is routed down to all atomic components by Coordinators and is used for synchronizing the three different transitions of the atomic models;
3. (q, t): Input message;
4. (y, t): Output message;
5. (done, t): Finishing message.

The last three are intermediate messages produced during the delivery of (@, t) and (*, t) messages.

Extra message types in the dynamic algorithm:

6. (sc*, t): A structural change requesting message sent from Simulator to its supervised Coordinator, or from a Coordinator to its parent Coordinator to indicate that the model asks for a structural change. Any Simulator or Coordinator can issue this message;
7. (sc, t): A structural change message sent from a Coordinator to its children, who sent out a request for structural changes, indicating that the children can carry out the structural change;
8. (start, t): An initializing message sent by Root Coordinator after a structural change. After receiving the (done, t) messages from the models experiencing structural changes, Root Coordinator sends the (start, t) message to start a new simulation phase. This message is used to initialize new-added models, and to get the next imminent event time for all the new models.

According to Chow et al. (1994), Root Coordinator executes a message loop while simulation time steps forward as time points when simulation events happen. As shown in Figure 1, Root Coordinator executes (@, t) and (*, t) in each message loop. In the regular DEVS simulation, the layouts of the simulation components are static. However, the layouts cannot stay unchanged if the dynamic structure is introduced in the simulation. These changes might be triggered by the states and the inputs of the models or by global simulation time. The triggering effectors might be found in the execution of (*, t) message. Therefore, we conclude that the (*, t) message is the place where structural changes might happen. Figure 2 shows the case that a dynamic structure change happens during a regular simulation process. Message sequence for structural change is shown in the dotted-line frame. Figure 2 presents a good expression on how the dynamic DEVS algorithm migrates from the regular DEVS simulation algorithm.

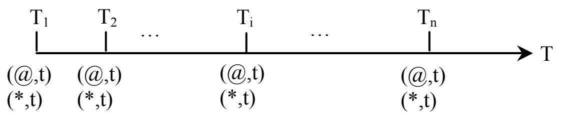

Figure 1: Sequence without Structural Change Function

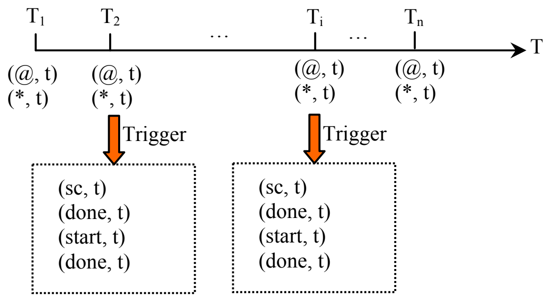

Figure 2: Sequence with Structural Change Function

The dynamic DEVS simulation algorithm is described in different hierarchical levels: Root Coordinator, Coordinator and Simulator.

Root Coordinator

```
t := tN of the topmost Coordinator
While t ≠ ∞
  Send (@, t) message to topmost Coordinator
  Wait for (done, t) message
  Send (*, t) message to topmost Coordinator
  if (sc*,tN) received from topmost Coordinator {
        Send (sc, t) to topmost Coordinator
        Wait for (done, t) message
        Send (start, t) to topmost Coordinator
        Wait for (done, tN) message      }
  else
        Wait for  (done, tN) message
```

Coordinator

```
When  (@, t) is received
    // Use Chow et al. (1994) algorithm.
End when

When (*, t) is received  {
  if tL <= t <=  tN    {
  ...//use Chow et al. (1994) procedure for (*,t)

  Wait (done, tN) from all i in synchronize set.
   if (sc*, tN) received and tN=minimum  {
       Retrieve model-changing list (modelname1,
         tN1),(modelname2, tN2),... supervised by
           current Coordinator.
         // modelname1 launches the structural
         // change procedure ("initial model").
       Send (sc*, tN) to its parent Coordinator.}
    Wait for (done, tN) messages
    tL := t
    tN := minimum of components' tN's
    if current imminent component is the one
          among the model-changing list {
         Send (sc*, tN) to parent Coordinator }
        Else {
            Clear the synchronize set
            Send (done,tN) to parent Coordinator }
    } else  { raise an error }
} // End when

When receive (sc, t) message from its parent  {
  Case: (t=tN and current message type of
    Coordinator is a structural change) {
      Backup current model set & links supervised
            by the Coordinator.  }
      Case: (t=tN is for the initial model) {
         route (sc,t) down to initial child who
                   sent (sc*,tN) previously.
          wait for (done,t) message from it.
      Case: (t=tN is ! for the initial model)
         send (sc,t) down to corresponding models
         wait for (done,t) message from them }
      Case: (t < tN)
        forward this message to its children
      Case: (t<tL or t>tN)
        raise error     }  // End Case
      Wait (done, t) from corresponding children.
      Current Coordinator processes structural
           change at its level.
      Send (done, t) to parent Coordinator
    } // End when

    When receive (start,t) from parent {
       Send (start, t) message to i
       ...  // {i| i∈ D-D'}. D' is the model set
    before structural change, while D is current
    model set after structural change.
       Wait (done, tN) from all components i
       Select tN = minimum of components' tN's
       Send (done, tN) to parent Coordinator
       if (component whose tN is minimum is in
    model-changing list in the Coordinator) {
        Coordinator sends (sc*,tN) to parent to
            trigger next step of structural change}
    } // End when
```

Simulator

```
// Use Chow et al. (1994) algorithm for (@,t)
          and (*,t) messages.

When receive (*, t) message {
 if (current state, elapsed time, input bag or
          global time) reach critical points {
      Calculate tN for the structural change
      Send (sc*, tN) to its parent Coordinator  }
 else
      execute regular simulations
} // End when

When receive (sc,t) from parent Coordinator
 if t ≠ tN { raise error }
  else  {
    S = δ_st(s, t, e, bag)
    // structural transition function calculates
         new structure state.
    Send (done,t) message to parent Coordinator }
} // End when

When receive (start, t) message from parent {
    initialize the new models and new ports
    tL := t
    tN := tL + ta
    s := s_0 (the initial state of the model)
    Send (done, tN) to parent Coordinator
} // End when
```

Root Coordinator, Coordinator and Simulator described above contain extended concepts comparing with those defined in the regular DEVS simulation. Root Coordinator is able to process the structural change requests and to issue structural change commands. We incorporated the function of the Network Executive mentioned in DSDE into the two abstract simulators: Coordinator and Simulator. In this way, the dynamic structure algorithm can be integrated into the regular simulation processes seamlessly. Coordinator is liable to know all possible states of structural changes and migrations between those states. The structure transition function in Coordinator is applied to execute those migrations. The structure transition function in Simulator executes structural changes within the associated atomic model. Our algorithm presents the launching mechanism of the structural changes. As we have shown, there are three steps in the structural change process:

- Requests for structural change: They always rise in an internal message (*, t). When receiving a (*, t) message, Each Simulator evaluates its states or/and its simulation time. If they are critical, the Simulator will send a request message (sc*, t) to its parent Coordinator for a structural change. Structural change is a chain of activities and these activities may span a period. Some changes can be initiated by Simulator while others cannot, such as adding a new atomic model, deleting a existed model or adding a new link between two models, etc. For the latter cases, the corresponding Coordinator launches structural changes instead of Simulator. As a conclusion, either Coordinator or Simulator can launch structural changes according to different situations.
- Structural change processing: Both Simulator and Coordinator perform structural change processes employing structural transition functions, which are introduced specially for dynamic structural simulations. In Simulator, structural transition function δst(s, t, e, bag) is used to calculate the next structural state of an atomic model. A new structural state is determined by the current model state (critical state), elapsed time since the immediately preceding state, input bag and global simulation time. In Coordinator, the structural change message (done, t) from the initial model triggers the structure transition function. When a simulation involves multiple levels, the structural change should be executed from bottom to top.
- Structural change finishing: At this stage, the simulation returns to the regular DEVS simulation process without losing any unprocessed information. It is under the control of Root Coordinator. After receiving all (done, t) messages in response to their corresponding (start, t) messages, Root Coordinator knows the time for the next imminent event. Then global time is advanced and simulations are stepped to a new stage.

4 CASE STUDY: AN AUTOMATED MANUFACTURING SYSTEM

In this section, we discuss how to apply the above concept to the model of an Automated Manufacturing System (AMS), an extension of the models presented in Wainer et al. (2005), Glinsky et al. (2004a), and Glinsky et al. (2004b). The AMS is composed of dedicated stations that perform tasks on products being assembled and conveyors that transport the products to/from those workstations.

The proposed AMS is a flow shop of autos. It consists of five parts:

1. Scheduler: Production cycle organization;
2. Display Controller: AMS status display;
3. Controller Unit: Conveyors control according to the production cycle provided by Scheduler;
4. Conveyors: Products transportation;
5. Workstations: Taking care of task implementation and quality control. (ES: Engine Assembly Workstation; PS: Painting Workstation; BS: Baking Workstation; QC: Quality Control Center; SS: Store Workstation).

The Scheduler knows the production cycle and indicates Controller Unit to execute the production cycle on both conveyors. Autos being manufactured are delivered to each workstation (in order to be served step by step) by the conveyors. The ES workstation takes charge of engine parts assembly; the PS workstation undertakes the painting task and other special painting tasks; baking is the subsequent procedure in the BS workstation after painting; the QC workstation serves as a quality center to evaluate the quality of autos; and the Store workstation distributes the autos to their corresponding warehouses. Figure 4 shows the layouts of the AMS system.

4.1 Dynamic Components of AMS System

During the system running, the structure of the AMS would not stay unchanged all the times. Some times, structural adjustment should be made to adapt to the changes of external environment. Two kinds of system adjustments are considered:

1. Workstation duty shifts (the workstations has different working capacity during day and night.);
2. In the PS workstation, two possible tasks are performed: color painting and chrome painting. Autos need color painting or both color painting and chrome painting. Painting selection is determined by the ‘control’ model residing in the PS workstation.

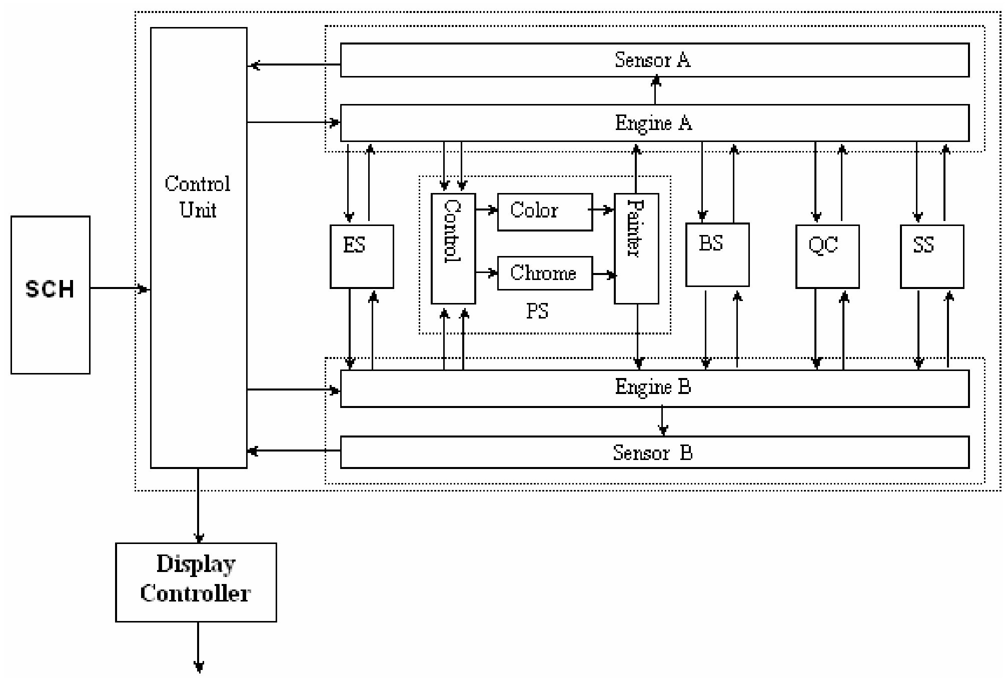

Figure 4: The Automated Manufacturing System Model

4.2 Case 1: Workstation Duty Shifts

According to the DSDE formal specification, the behaviors of the basic model ς are shown in figure 5 and figure 6. ES and ES', representing engine workstation during daytime and night respectively, can be considered as two structural states of the basic model ς. Zes,0 and Zes,1 represent the input functions of ES and ES'; while Zς,0 and Zς,1 represent the output functions of the structural model ς. χ is the Network Executive described in DSDE.

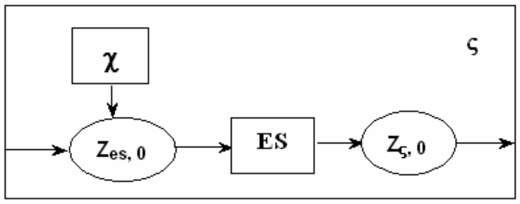

Figure 5: ES Structure Layout During Daytime

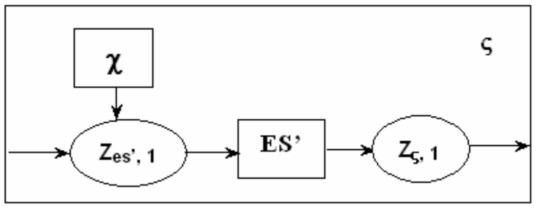

Figure 6: ES' Structure Layout During Night

Assume that the working time is 30mins during daytime while 40mins during night. Daytime duty is from 8:00am to 5:00pm everyday, the rest is the nighttime duty. The structural change is implemented as follows:

1. When ES reaches the critical time points (critical time point t_c = {t| t∈[8:00am, 5:00pm], 5:00pm – t <= 30mins} or {t| t∈[5:00pm, 8:00am], 8:00am – t <= 40mins}), calculates tN for the coming structural change, sends (sc*, t) message to Top Coordinator.
2. If the tN is the minimum one, Top Coordinator sends (sc*, t) message to Root Coordinator.
3. Root Coordinator forwards the current simulation time to tN and issues (sc, t) message.
4. Top Coordinator receives (sc, t) and sends it to the basic model ς.
5. Simulator associated with the basic model ς calculates the new structural state using the structural transition function δ_{st}, a (done, t) message is sent back.
6. Root Coordinator sends (start, t) message to initialize the new simulation stage. When (done, t) messages are received, a new regular simulation stage begins.

4.3 Case 2: Dynamic Components in PS Workstation

The PS workstation is a coupled model including four atomic models: Controller, Color, Chrome and Painter. The atomic model 'Chrome' is an optional component. Painting selection is determined by the 'Controller'. Figure 7 and 8 show the two structural states of the network model Θ. $Z_{i,\alpha}$ (i = Controller, Color, Chrome and Painting; α = 0,1)is the input function of the atomic models while $Z_{\Theta,\alpha}$ is the output function of the network model Θ. Z_χ is the input function of the Network Executive χ.

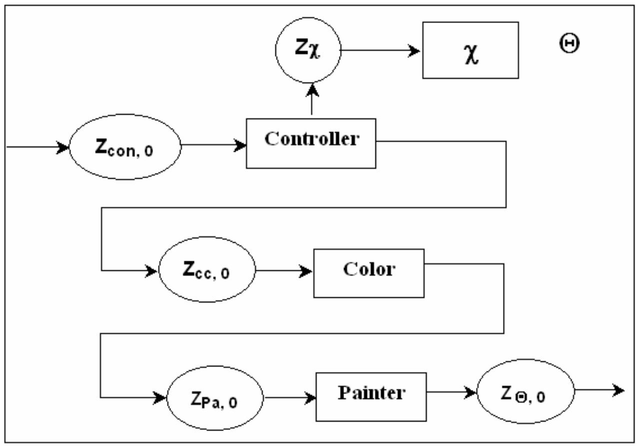

Figure 7: Painting Mode I in PS Workstation

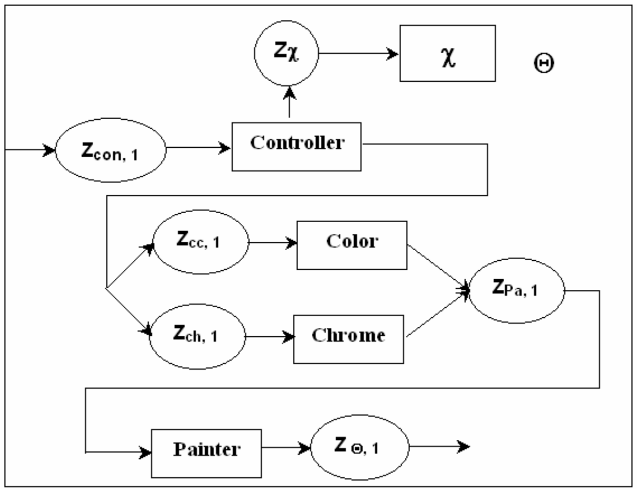

Figure 8: Painting Mode II in PS Workstation

Generally, the autos on the conveyor are painted with specific colors. Therefore only the atomic model 'Color' is needed in the PS. Assume that the current bulk of autos on the conveyor need to be painted both color and chrome. The atomic model 'Chrome' should be added into PS automatically. When the Simulator associated with the atomic model 'Controller' detects this change, the following structural change happens:

1. The Simulator associated with the basic model 'Controller' sends (sc*, tN) message (here tN can be the current time, which means structural change will happen right away) to its parent Coordinator, which is associated with the network model Θ.
2. The Coordinator retrieves the model-changing list according to the message value in (sc*, t) message from Simulator 'Controller'. The model-changing list should be: (Controller, tN1), (Chrome, tN2), (Painting, tN3). The Coordinator then sends (sc*, t) message upwards.
3. At time tN1, when the Coordinator receives (sc, t) message from its parent, it forwards the message to its corresponding Simulator (here are the Simulators associated with the basic models 'Controller', 'Chrome' and 'Painter' because tN1 = tN2 = tN3) under its supervision.
4. When the Simulators receive the (sc, t) message, corresponding structural changes are implemented and (done, t) messages are sent back to the Coordinator.
5. When all structural changes in the Simulators supervised by the Coordinator Θ finish, the Coordinator Θ begins to execute structural changes on its own level. In this case, new links are created between 'Controller', 'Chrome' and 'Painter'. A (done, t) message is returned to the upper level Coordinator.
6. A (start, t) is issued by Root Coordinator once the structural changes finish. After getting next imminent tN, global simulation time is stepped forward to this tN. Root Coordinator issues a (@, t) message and a new stage of simulation begins.

5 CONCLUSIONS

We presented a more practical simulation algorithm for Dynamic Structure DEVS modeling and simulation. Our proposal stems from both DSDE and dynDEVS algorithms. We used the DSDE formal specifications, and parts of dynDEVS simulation algorithms. We integrated the functions of Network Executives into the corresponding Simulators and Coordinators by introducing the structure transition functions in Simulators and Coordinators. We inherited the message types used in the regular DEVS algorithm while special message types conducting dynamic structural changes are devised. We also have a specific mechanism to launch dynamic structural changes, which links the regular state transitions of models and the structure transitions of some models in simulation processes. Hence, our proposed dynamic algorithm has been embedded into the regular DEVS simulation algorithm seamlessly.

We provided ground for user-defined model design and simulation (state transition functions, structure transition functions and output functions). Users can focus on their application-oriented model design (how to devise the transition functions mentioned above for their own models) while some common parts can be reused. Model design and simulation architecture based on DEVS theory has been extended by involving the structure transition functions and our proposed launching mechanism. This also provides a sound base for the execution of real-time simulations with dynamic components.

REFERENCES

Barros, F. J. 1995. Dynamic Structure Discrete Event System Specifications: A New Formalism for Dynamic Structure Modeling and Simulation. *Proceedings of the 1995 Winter Simulation Conference.* pp.781-785. Arlington, USA.

Barros, F. J. 1998. Abstract Simulators for the DSDE Formalism. *Proceedings of the 1998 Winter Simulation Conference.* pp.407-412. Washington DC, USA.

Barros, F. J. 1997. Modelling Formalisms for Dynamic Structure Systems. *ACM Transactions on Modeling and Computer Simulation*. Vol. 7, No. 4, pp. 501-515.

Cho, S., and T. G. Kim. 2001. Real Time Simulation Framework for RT-DEVS Models. *Transactions of the Society for Computer Simulation International.* Vol. 18, No. 4, pp. 203 – 215.

Cho, Y. K., B. P. Zeigler, H. J. Cho, H. S. Sarjoughian, and S. Sen. 2000. Design Considerations for Distributed Real-Time DEVS. AIS 2000. Tucson, USA.

Chow, A.C., and B. P. Zeigler. 1994. Revised DEVS: A Parallel, Hierarchical, Modular Modeling Formalism. *Proceedings of the SCS Winter Simulation Conference*.

Glinsky, E., and G. Wainer. 2004a. Modeling and Simulation of Systems with Hardware-in-the-loop. *Proceedings of the 2004 Winter Simulation Conference.* Washington DC, USA.

Glinsky, E., and G. Wainer. 2004b. Model-Based Development of Embedded Systems with RT-CD++. *Proceedings of the WIP session, IEEE Real-Time and Embedded Technology and Applications Symposium.* Toronto, Canada.

Hong, J., H. Song, T. G. Kim, and K. H. Park. 1997. A Real-time Discrete Event System Specification Formalism for Seamless Real-time Software Development. *Discrete Event Dynamic systems: Theory and Applications,* Vol. 7, No. 4, pp. 355-375.

Kim, T.G., S. M. Cho, and W. B. Lee. 2001. DEVS Framework for Systems Development. *Discrete Event Modeling & Simulation: Enabling Future Technologies.* Springer-Verlag.

Uhrmacher, A. M. 2001. Dynamic Structure in Modeling and Simulation: A Reflective Approach. *ACM Transactions on Modeling and Computer Simulation*. Vol. 11, No. 2, pp. 206-232.

Uhrmacher, A. M., and J. Himmeelspach. 2004. Processing dynamic PDEVS models. *Proceedings of the IEEE Computer Society's 12th Annual International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems (MASCOTS'04).* Volenlam, Netherlands.

Wainer, G., E. Glinsky, and P. Macsween. 2005. Model-Driven Architecture of Real-Time Systems. *Model-driven Software Development - Volume II of Research and Practice in Software Engineering.* S. Beydeda and V. Gruhn eds., Springer-Verlag.

Zeigler, B. P., T. Kim, and H. Praehofer. 2000. Theory of Modeling and Simulation: Integrating Discrete Event and Continuous Complex Dynamic Systems. *Academic Press.*

AUTHOR BIOGRAPHIES

HUI SHANG is currently pursing her Master of Science degree at the Department of Systems and Computer Engineering at Carleton University. Her research interests are focused on DEVS simulation and dynamic structure simulation. Her e-mail address is <shanghui@sce.carleton.ca>.

GABRIEL WAINER received the M.Sc. (1993) and Ph.D.degrees (1998, with highest honors) of the Universidad de Buenos Aires, Argentina, and Université d'Aix-arseille III, France. He is Associate Professor in the Dept. of Systems and Computer Engineering, Carleton University (Ottawa, ON, Canada). He was Assistant Professor at the Computer Sciences Dept. of the Universidad de Buenos Aires, and a visiting research scholar at the University of Arizona and LSIS, CNRS, France. He is Associate Editor of the Transactions of the SCS. He is Associate Director of the Ottawa Center of The McLeod Institute of Simulation Sciences and a coordinator of an international group on DEVS standardization. His email and web addresses are <gwainer@sce.carleton.ca> and <www.sce.carleton.ca/faculty/wainer>.