A Flexible Dynamic Structure DEVS Algorithm towards Real-Time Systems

Hui Shang
Gabriel A. Wainer

Department of Systems and Computer Engineering
Carleton University
1125 Colonel By Drive Ottawa ON.
K1S 5B6, Canada
email: {shanghui, gwainer} @ sce.carleton.ca

Keywords: Dynamic Structure, DEVS, Real-Time Systems, Experimental Environment

ABSTRACT

A dynamic structure DEVS algorithm – Flexible Dynamic Structure DEVS is introduced in this paper. The dynamic structure is a salient supplementary of the Real-Time DEVS-based experimental environment we proposed before. By adapting the system organization to the changing internal/external environments while the systems are in progress, the functionality of the experimental environment is improved greatly. Consequently, it is possible to build more reliable and flexible Real-Time systems. This paper describes the abstract simulators for dynamic structure DEVS; moreover, the implementing process is presented to show how the dynamic structure abstract simulators work seamlessly with the regular DEVS simulating engine. The Real-Time DEVS-based experimental environment with the dynamic structure function exploits a new space to build reliable and adaptive Real-Time systems.

1. INTRODUCTION

Real-Time systems are extremely noted for the critical timeliness requirement and rigorous correctness of results. Development of Real-Time systems calls for a unified process to bridge the gaps between design and implementation.

Discrete EVent System Specification (DEVS) [1] is a popular approach in the simulation domain. Modularity and hierarchy of DEVS make it easy to represent systems in a structured way. DEVS formalism has been extended to the Real-Time domain, which is helpful to build a DEVS-based framework to enhance the seamless transformation from the system design to the implementation of Real-Time systems [2-5]. However, they did not mention that how the seamless transformation between the different development stages. MDA (Model Driven Architecture) technology proposed in [6-8] exhibits a Real-Time DEVS-based experimental environment, in which substantial examples are offered to show how the large-scale Real-Time systems are built up from the virtual DEVS models. In the experimental environment, DEVS models are integrated as a portion of a Real-Time system. The Real-Time DEVS-based experimental framework makes it possible to analyze and test models in a risk-free environment. In addition, the DEVS models can be replaced by real world processes gradually after they are safely tested in the Real-Time simulation environment. The experimental environment provides a sound platform to analyze the reliability. Accordingly, the performance of the Real-Time systems is improved.

In common, the experimental environment is represented by DEVS models statically. For instance, the elevator system [7] and the AMS system [8] are both of static structures. These systems interact with their external worlds statically by the specified input and output ports. The static structured systems are not reasonable to represent the systems residing in a changing environment. It is essential to introduce a dynamic structure function to fit the varied requirements of responding to the changing external environment or recovering from errors automatically. Flexibility and reliability, therefore, are reached by adjusting the structures of models dynamically.

This paper aims to provide a flexible dynamic structure DEVS algorithm and integrate it into the Real-Time DEVS-based experimental environment. The proposed dynamic structure DEVS not only concerns Real-Time context but also cooperates with the regular simulation engine

seamlessly. The algorithm derives from the existing dynamic structure DEVS specifications but employs a different message set and abstract simulators. The Flexible Dynamic Structure DEVS Algorithm carries out a more elegant Real-Time development framework.

2. BACKGROUND

Dynamic structure DEVS is a dynamic structure algorithm based on DEVS theory. It is a new simulation paradigm supporting structural changes to full extent, ranging from simple model/connection addition/deletion to the exchange of models between networks of models [9]. Moreover, the structure should be dynamically adjusted according to internal or external changes of the systems. Therefore, the system structures are able to adapt to the system real requirements. Dynamic structure DEVS is a promising solution to the changing environments, such as Real-Time systems and embedded systems.

There are two kinds of dynamic structure DEVS. DSDE [10-11] (Dynamic Structure Discrete Event System Specification) divides models into two groups: basic models and network models. The basic models are atomic structure units which cannot be split. Network models are coupled components, consisting of multiple basic structure models and interconnections that involve structural changes. A Network Executive is a modified basic model to conduct structural changes in the network models. Network Executive stores all possible states of structural changes and their corresponding component sets in each structural state. Network Executive is a structure control component that knows all possible structural states within its action domain. In a network model, Network Executive is the only component to conduct the structural changes. The centralized Network Executive ensures that the structure transition is executed sequentially without any conflicts between structural change functions of the models. Differently, the dynDEVS formalism [12] does not introduce an extra component to conduct dynamic structural changes. dynDEVS and dynNDEVS present atomic and coupled dynamic structure models respectively. Structural changes can be conducted both in atomic model and coupled model separately. $\rho\alpha$, the atomic model transition function, and ρN, the coupled model transition function, are included to conduct the structural changes. However, structural conflicts might occur due to the independency of the model transformations. Proper constraints should be added to avoid the conflicts. These constraints make the structural transition functions more complicated.

CD++ [13] is a modeling and simulation tool that implements DEVS models simulation based on an abstract simulator mechanism. Atomic models are defined using a state-based approach (encoded in C++ or an interpreted graphical notation); while coupled models contain atomic models composition and interconnecting information of those atomic models. CD++ has been widely used in various applications from simple queuing systems to complex urban traffic systems or physical systems. CD++ employs the abstract simulators proposed in [14]. Message drives the simulation according to the scheduled time points. Different versions of CD++ have been developed to facilitate various applications.

- Stand alone CD++ implements DEVS and Cell-DEVS simulation.
- Parallel CD++ is aiming to enhance the performance of Cell-DEVS simulation by distributing calculation of different cells over multiple processors.
- Distributed CD++ is developed to facilitate the coordination of the different simulating engines in different sites through the standard distributed computing protocols.
- Real time embedded CD++ is constructed especially for Real-Time embedded system. A timing feature of the Real-Time systems has been included in CD++ to check the timing deadlines of given points of the systems, based on which the scheduability of the Real-Time system can be judged.

3. REAL-TIME CONTROL CONTEXT

"A real time computer system is the one in which the correctness of the system behaviour depends not only on the logical results of the computation, but also on the physical instant at which these results are produced" [15].Real-Time systems are well known for their critical timeliness. Besides, 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). As improper decisions may lead to catastrophic consequences for assets or lives, correctness is another distinct characteristic.

With sensitive timeliness and rigorous correctness, Real-Time systems pose critical challenges to software design and development. Fortunately, DEVS-based experimental environment with dynamic structure function provides a sound underlying platform for the Real-Time systems. Dynamic structure DEVS, to some extent, makes it possible for the system designers and developers to tackle those challenges. The following points out how dynamic structure DEVS works to improve the reliability and performance of the Real-Time systems.

- Sensitive timeliness of Real-Time systems requires higher precision on schedulability. Explicit formal specification of DEVS presents a solid base for the accuracy of schedulability. Sometimes the critical timing constraints are achieved only by dynamic scheduling. The dynamic structure function brings more space for the development of dynamic scheduling.
- Fault tolerance is a distinct trait of Real-Time systems. Due to the rigorous requirements on the result correctness, the Real-Time systems should be able to recover from the faults automatically. If the structure is static, it is difficult to rectify these errors dynamically. Dynamic structure allows more possibilities for fault tolerance design. For example, if a DEVS model or a piece of hardware is crashed, the error would be detected and reported to the controller, a standby model or hardware can be initiated to replace the crashed one without interrupting the system running.
- In most cases, it is difficult to predict the situation of the surrounding environment. Some unexpected events from the surrounding environment may cause overheads or oscillations of the systems. The dynamic structure function is capable of dealing with these unpredicted situations by adjusting the system organization to the situation dynamically.
- Embedded systems are an important group of Real-Time system. Memory in embedded systems is an expensive resource and should be managed properly during running time. Dynamic structure can remove the unused parts of the systems and keep the minimum usage of memory dynamically.

4. ABSTRACT SIMULATORS FOR DYNAMIC STRUCTURE DEVS

Barros [9] [10] introduced a model of executive to execute structural changes of network models. The executive, as the only structural change conductor at each level of the model hierarchy, prevents ambiguity resulting from different components' structural change requirements. We defined a model of Structure Agent at each level to undertake the structural changes. Different from the Network Simulator in DSDE, which is a combined processor for both a coupled model and a model of executive, we assigned an independent abstract simulator to each model of Structure Agent, called Revised Simulator for Structure Agent. Hence, the Coordinator, the processor for a coupled model, skips the details of the structural changes. In addition, a different message set is employed to accommodate the existing regular simulation algorithm (Chow's algorithm [14]). Accordingly, the Root Coordinator, the Coordinator and the Simulator should be modified to link the dynamic structural changes with the regular simulation. The Coordinator delivers the structural change message to its children or to the corresponding Revised Simulator for Structure Agent. The Simulator informs the structural changes when the conditions of the structural changes are satisfied.

Root Coordinator

```
t := tN of the topmost Coordinator
Structure change value: request = 0
While t ≠ ∞
  Send (@, t) message to topmost Coordinator
  Wait for (done, t) message
  Send (*, t) message with the value of
'request' to topmost Coordinator
  Wait for (done, tN) message
  Request = msg.value() (structural change
value)
End while
```

Coordinator

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

When (*, t) is received {
if(msg.value() != 0) // structural change
 {
  if (the coupled model is a structure
  component)
  {
    catch the structure agent model of the
    coupled model into the structured set.
   } //end if
  for the component i of the coupled model
  {
    if (the component is a structure component)
    {
```

```
      catch the component into the structured
      set.
    } end if
  } // end for
  for the component i in the structured set
  {
    send (*, t) to the component i
  } // end for
  Wait until all (done, tN)'s are received from
  the components in structured set
  if(i ∈ D – D')
  //where D is the component set before the
  structural change while D' is the component
  set after the structural change.
  if(I is in the synchronized set) erase i.
  // the deleted models are removed from the
  synchronized set
  if(i ∈ D' - D) send (St, t) to i.
  // the newly added models are initialized
  Wait until (done, tN)'s are received from all
  new models

  tL := t
  tN := minimum of components' tN's
  send (done, tN) to the parent Coordinator
  } // end if (msg.value() != 0)
else //regular simulation
{
  ...//use Chow et al. (1994) procedure for
  (*,t)
   Wait until all (done, tN)'s are received
  If(the msg.value != 0) add the component into
  structure request set.

  If the structure request set is not empty
  {
    if the component is also in the imminent
    children set
    {
      capture the component into the
      structured set
      send (done, tN) with the structural
      change value to the parent Coordinator
      clear structure request set.
    }
  }
  else
    send (done, tN) with (vaue = 0) to the
    parent Coordinator
}//else
} end when
```

Simulator

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

When receive (*, t) message {
If (msg.value() == 0)
{
     case tL≤ t < tN
           e := t – tL
           s := δext( s, e, bag )
           if structure request is not set
              empty bag
     end case
     case t = tN and bag is empty
           s := δint( s )
     end case
     case t = tN and bag is not empty
           s := δcon( s, bag )
           if structure request is not set
              empty bag
     end case
tL := t
tN := ta(s)
if(structure request is set)
send (done, tN) with the structural change
value to the Coordinator
else
send (done, tN) with (value = 0) to the
Coordinator
}
} // End when
When receive (start, t) message from parent
{
tL := t;
tN := tL + ta
s := s0 (the initial state of the model)
Send (done, tN) to parent Coordinator
} // End when
```

Revised Simulator for Structure Agent

```
When receive (*, t) messages
{
if (msg.value() != 0) // structural change
message
    {
    value = msg.value()
    tL = t
    tN = Infinity;
    s := δint( s, value, t )
    send (done, t) to the Coordinator
    }
}// end when
```

Fig. 1 Abstract Simulators

5. FLEXIBLE DYNAMIC STRUCTURE DEVS ALGORITHM

In CD++, simulation is driven by the message passing among processors. In the regular abstract simulators, six kinds of messages are used: @ (collect message used to collect output of each model), I (initial message indicating the start of the simulation), *(internal message used to signal a state change due to an internal event), X (external message used when an external event arrives), Y (output message carrying out the models' output) and D (done message indicating the completion of the task. According to different waiting modes of the abstract simulators, three kinds of done messages are defined: an initial done message is sent back when the initial message is finished processing; a collect done message indicates the processing of collect message has been finished; an internal done message follows an internal message to complete the internal message processing.). In the regular simulation, both the internal message and the done message do not need specific message values. However, in order to keep consistence with the existing abstract simulators, we extended them to carry structural change values which indicate different structural organizations. The following

messages are used in the structural change process:

- Structural Change Request ((done, t) (sc)): as shown in the abstract simulator for atomic models - Simulator (Fig.1), a (*, t) message triggers the state calculation of a model. A value which indicates a structural change request could be assigned to the internal done message if the structural change conditions of a model are satisfied. The non-zero value in the internal done message will be detected by the Root Coordinator and the Coordinator. And then the process of structural change starts.
- Structural Change ((*, t) (sc)): according to Chow's algorithm, an internal message triggers the model's external/internal transition functions. If we consider all possible model structures as the state space of a model of Structure Agent (SA), we can use an internal message to trigger the structural transition function of the SA. The Root Coordinator issues an internal message with a structural change value to execute the required structural changes.
- Start Message (St, t): When the structural changes are completed and all the structural done messages has been returned, the Coordinator will issue a (St, t) message to initialize the newly added models and retrieve their tNs. And then a new regular simulation phase starts.

Message-driven mechanism is the key in DEVS simulation. Each abstract simulator executes a specific message sequence to advance the simulation process. Fig. 2 shows the message sequences of each abstract simulator. The Root Coordinator, the Coordinator and the Simulator execute the regular simulation. If there is no structural change request, the Simulator returns an internal done message with a value of zero, indicating a regular process for the next simulation cycle. Whereas, when a non-zero structural change value (sc) is sent with the internal done message (a structural change request), the structural change process is triggered. As shown in Fig. 2, when the Root Coordinator detects a structural change request, it issues an internal message (*, t) with a structural change value (sc) to launch a structural change process. The Coordinator passes the message to the RevSimulator (Revised Simulator for Structure Agent) to execute the structural change process at each level of the model hierarchy.

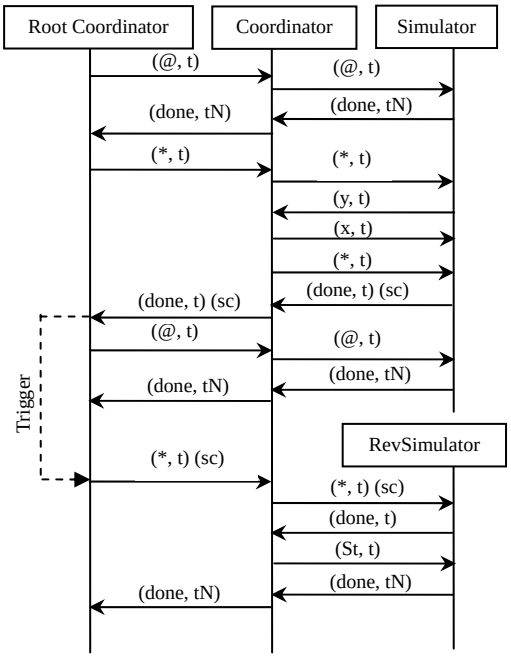

Fig. 2 Message Sequence with a Structural Change

The Structural Change process contains three stages: creating structural change requests, structural change processing and structural change post-processing.

If the state of an atomic model satisfies certain structural change conditions, the atomic model initiates a structural change process by sending a structural change request to its Coordinator.

Coupled models are compositions of atomic models and links between them. They do not contain any states or execute the transitional functions. The outputs from coupled models stem from the corresponding atomic models. Hence, atomic models are the only entities that can raise structural change requests.

The Root Coordinator invokes a structural change process by sending an internal message with a structural change value (*, t) (sc).

Structure Agent is an exclusive atomic model that records structural states at each model level. RevSimulator is inserted into the simulating processor tree as a structural change conductor to take charge of structural changes. The RevSimulator only accepts and executes structural change messages while skips other regular

simulation messages. Fig. 3 shows the structure of the simulating processor tree with RevSimulators. SAXs represent the RevSimulators. CXs denote Coordinators and SXs are Simulators.

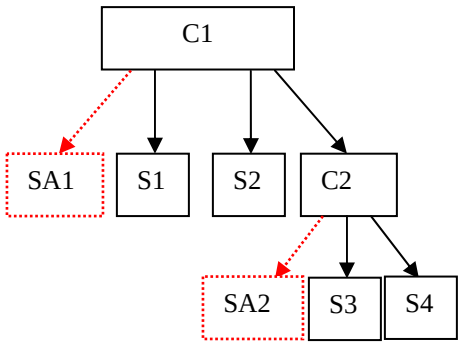

Fig. 3 Simulating Processors Tree

The Coordinator sends (St, t) and asks for the tNs of the newly added models when the structural change of this level is done. When the Root Coordinator gets the imminent tN, a new regular simulation cycle begins.

6. OPERATION BOUNDARIES

In the dynamic structure process, the operation boundary is defined as a safe scope to conduct a meaningful operation [17]. The operation boundary ensures clear and determined operations among models in dynamic structural process. Different design perspectives of the dynamic structure algorithms lead to different operation boundaries. There are four basic structural change types: (1). Addition of a component; (2) Removal of a component; (3) Addition of a link between components; (4) Removal of a link between components. In the DEVS-based systems, a component refers to an atomic model or a coupled model. Most structural change processes are the composition of the four basic forms. In each structural change operation, the following operation boundaries should be complied with:

1. A model cannot add/remove itself.
2. A model can only be added or removed by the Structure Agent residing at this level.
3. A model cannot add/remove a link to other peer models. Models are hierarchically coupled. Each model is independent from others at the same level and cannot be controlled by its peers.
4. A link between models at the same level can only be added/removed by the Structure Agent residing at this level.

7. CONCLUSIONS

Real-Time systems are a kind of systems with sensitive timeliness and rigorous correctness of results. The proposed Real-Time DEVS-based experimental environment facilitates the seamless development of Real-Time systems. Introducing a dynamic structure function provides a more reasonable platform for the transformation from design to implementation.

The Real-Time context concerns explicitly unveil the advantages brought by the DEVS-based experimental environment together with the dynamic structure DEVS function. The proposed abstract simulators and the structural change process of Flexible Dynamic Structure DEVS Algorithm show the cooperation between the dynamic structural changes and the regular simulation. Undoubtedly, dynamic structure function is an important supplementary of the previous Real-Time DEVS-based experimental environment. Structural change levels and operation boundaries restrict the dynamic structural changes within a clear and determined range. The case study for the proposed dynamic structure DEVS algorithm is ongoing to verify the correctness. The proposed Real-Time DEVS-based experimental environment permits a unified process for designing and developing Real-Time systems. As a result, DEVS models can be easily tested and smoothly transformed to the real hardware benefited from the delicate Real-Time DEVS-based experimental environment.

8. REFERENCES

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

[2] 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.

[3] 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.

[4] Hong, J.S., 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.

[5] 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.

[6] 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.

[7] Glinsky, E., and G. Wainer. 2004. "Modeling and Simulation of Systems with Hardware-in-the-loop". In the Proceedings of the 2004 Winter Simulation Conference. Washington DC, USA.

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

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

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

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

[12] 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.

[13] Wainer, G. 2002. "CD++: a toolkit to define discrete-event models". In Software, Practice and Experience. Wiley. Vol. 32, No.3, pp. 1261-130.

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

[15] Kopetz, H. 2000. "Software Engineering for Real-Time: A Roadmap". In the Proceedings of the Conference on the Future of Software Engineering, pp.201-211, Limerick, Ireland

[16] Uhrmacher, A.M., and J. Himmeelspach. 2004. "Processing dynamic PDEVS models". In the 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.

[17] Hu, X.L., B.P. Zeigler, and S. Mittal. 2005. "Variable Structure in DEVS Component- Based Modeling and Simulation". Simulation, Vol. 81, Issue 2, February 2005, pp. 91-102.