DEVS Tutorial--
DEVS and Distributed DEVS

Ming Zhang, PhD
Paradise Research Laboratory
School of Information Technology and Engineering
University of Ottawa
Outline

- How to Find DEVS Materials:
  - Google: ACIMS
- DEVS Key Concepts
- DEVS Tools and Distributed DEVS Tools:
  - DEVS/Grid, DEVS/P2P, DEVS/SOA...
- DEVS/RMI—A Reconfigurable Distributed DEVS Framework
- Solving large-scale simulation models using DEVS/RMI
- DEVS in the near future
What is and Why Use DEVS

DEVS—Discrete Event System Specification

Strictly Defined Modeling and Simulation Framework based on DEVS Formalism.

Flexible Hierarchical Modeling and Simulation Structure.

Support Model Reuse by Model Repository.

Support both discrete event and continuous system modelling and simulation.

Can be used for formalized design and system design verification and validation.

Can be used for agent-based simulation.

And more...
DEVS and Non-DEVS based Simulation

- DEVS is formalized.
- DEVS is hierarchical.
- Clearly separating modeling and simulation framework.
- Models' behavior and their inter-relation are separated.
Concept View of Entity Relationship[*]
Separate Model and Simulator [*]

- Single processor
- Distributed Simulator
- Real-Time Simulator
- Non DEVS
- DEVS Simulation Protocol
- C++
- Java
- Other Representation

The DEVS simulation protocol is the agreement between the DEVS modeler and the implemented simulator.
DEVS Formalism

- DEVS stands for Discrete Event System Specification.
- DEVS Formalism is used to strictly define the model component behaviour.
- DEVS Formalism has many extensions to satisfy the emerging requirements.
Basic DEVS Formalism[*]

A Discrete Event System Specification (DEVS) is a structure
\[ M = (X, S, Y, \delta_{\text{int}}, \delta_{\text{ext}}, \delta_{\text{con}}, \lambda, ta) \]
where
- \( X \) is the set of input values
- \( S \) is a set of states,
- \( Y \) is the set of output values
- \( \delta_{\text{int}}: S \rightarrow S \) is the \textit{internal transition function}
- \( \delta_{\text{ext}}: Q \times X^b \rightarrow S \)
  is the \textit{external transition function}, where
  \( Q = \{(s, e) \mid s \in S, 0 \leq e \leq \text{ta}(s)\} \) is the \textit{total state set}
- \( e \) is the \textit{time elapsed since last transition}
- \( X^b \) denotes the collection of bags over \( X \)
  (sets in which some elements may occur more than once).
- \( \delta_{\text{con}}: Q \times X^b \rightarrow S \)
  is the \textit{confluent transition function},
- \( \lambda: S \rightarrow Y^b \) is the output function
- \( \text{ta}: S \rightarrow \mathbb{R}_{0,\infty}^+ \) is the \textit{time advance function}.
Basic Concept in DEVS[*]
Internal Transition[*]

- Make a transition
- Time advance
- Generate output

Using the output function
Using the internal transition function
External Transition[*]

Make a transition using the external transition function.

Time advance

elapsed time

input

Make a transition

using the external transition function
Confluent Function[*]

Input

Make a transition

Elapsed time

Time advance

Generate output

Output

Using the confluent transition function
Basic DEVS Formalism Example[*]

\[ \text{DEVS}_{\text{scuba}} = (X, Y, S, \delta_{\text{ext}}, \delta_{\text{int}}, \lambda, \tau_a) \]

where

\[ X = \{ \} \]
\[ Y = \{ 1 \} \]

\[ S = \{ "sixty", "forty", "five", "surface1", "surface2" \} \times R_{\geq 0}, \infty^+ \]

\[ \delta_{\text{int}} ("sixty", \sigma) = ("surface1", 60) \]
\[ \delta_{\text{int}} ("surface1", \sigma) = ("forty", 25) \]
\[ \delta_{\text{int}} ("forty", \sigma) = ("five", 5) \]
\[ \delta_{\text{int}} ("five", \sigma) = ("surface2", \infty) \]

\[ \lambda (\text{phase}, \sigma) = 1 \]
\[ \tau_a (\text{phase}, \sigma) = \sigma \]

\[ \delta_{\text{ext}}(\text{phase}, \sigma, e, x) = \begin{cases} "five", 5 & \text{if } \text{phase} \neq "surface1", "surface2", \text{or } "5" \\ (\text{phase}, \sigma-e) & \text{otherwise} \end{cases} \]

\[ \delta_{\text{con}}(\text{phase}, \sigma, e, x) = \delta_{\text{ext}}(\text{phase}, \sigma, e, x) \quad \text{//pay attention to external event (call)} \]
Hierarchical Model Construction[*]

Atomic: lowest level model, contains structural dynamics -- model level modularity

Coupled: composed of one or more atomic and/or coupled models

SimpA rc.efa
DEVS Tools

- ADEV (by Dr. Nutaro, ORNL)
- CD++ (by Dr. Wainer, Carleton Univ.)
- Tools Developed at ACIMS, Univ. of Arizona:
  - DEVSJAVA
  - DEVS/CORBA
  - DEVS/Grid
  - DEVS/P2P
  - DEVS/SOA
  - DEVS/HLA
  - DEVS/RMI
DEVSJAVA

- Java Implementation of DEVS
- Support discrete and continuous system modeling and simulation
- Support As-Fast-As-Can, Real Time DEVS simulation.
- Support Distributed Simulation, but with limited functionalities.
- Support dynamic structure changes through “variable structure”.
Distributed DEVS--Why?

- Reducing model execution time.
- Overcoming limited memory for a single machine to handle large models.
- Obtaining scalable performance.
- Handling geographically distributed users and/or resources (e.g., databases, specialized equipment).
- Integrating simulations running on different platforms.
- Dealing with fault tolerance.
DEVS/P2P[1]

- Use JXTA Pipe Interface as middleware to support distributed Execution of DEVS.
- Need additional layer for simulation time management.
- Prototype developed, not see application on complex and large-scale models.
DEVS/P2P-architecture
DEVS/P2P-communication between simulators

- DEVS
  - Model
  - (DEVS protocol)
  - Output port
  - DEVS msg

- DEVS/J XTA
  - P2P Simulator Proxy
  - (DEVS/P2P Protocol)
  - J XTA msg

- J XTA
  - Output pipe
  - (J XTA Protocol)

- P2P Simulator Proxy
  - Input pipe listener
  - J XTA msg

- DEVS msg
  - Input port
  - Output port
DEVS/Grid [2]

- Use Grid infrastructure to run DEVS in distributed fashion.
- Rely on existing Grid management framework, such as Globus.
- Not see application on solving real-world simulation models.
DEVS/Grid [2]-architecture
DEVS/SOA—recent advance [3]

- Use SOA as the basis architecture.
- Use most current web service technology.
- DEVS can be run on internet!
- Performance is the big issue?
DEVS/SOA-architecture
DEVS/RMI--Motivation

- Portable distributed simulation framework
- Support dynamic re-configuration of simulation in a distributed environment
- Eliminate the model code change when mapping models to computing nodes
- Flexible to implement partition algorithm in a distributed environment
- Toward very large-scale distributed simulation.
Java RMI(1)

- Maintaining the original object architecture built for a single processor, which is important for building large-scale scalable system.
- Task or computing workload distribution is at object level, which helps on solving load-balance, fault-tolerance problems in distributed computing in an easier way.
- Make the design of highly dynamic and reconfigurable distributed framework easier; Systems integration can be performed to a higher degree.
Java RMI(2)
DEVS/RMI Architecture

Model Structure

Configuration Engine

Simulation Controller

Machine 1
- RMI Naming Server
- Remote Simulator
- Model

Machine 2
- RMI Naming Server
- Remote Simulator
- Model

Machine 3
- RMI Naming Server
- Remote Simulator
- Model

Simulation Monitor
DEVS/RMI-A Flexible Framework

- Integrate Java RMI to existing DEVS/JAVA objects framework.
- Using both local and remote simulators.
- No additional simulation time management.
- No model code change except adding a new attribute for model code to assign model to computing node.
- Flexible and dynamic re-configurable.
DEVS/RMI--Simulator Relations
DEVS

Parsing as parameters

RMICoordinator

Decomposing Model

Sub-Models (Atomic or Coupled)

If sub_model.putwhere = “remote host”

Create and Assign Local Simulators

Dynamic Creation of Remote Simulators

Static Creation of Remote Simulators

Load the model to “Remote Host”, create the simulators there, obtains the Remote References of the Remote Simulators.

Create Remote References of the Remote Simulators using pre-defined RMI URL.

Initialize RMICoordinator

Start Execution of Distributed Simulation

DEVS/RMI Simulation Controller
DEVS/RMI-- RMI Overhead Test

Generator

Machine 1

Processor10
Processor11
............... 
Processor1n

Machine 2
DEVS/RMI--RMI Overhead

Simulation Execution Time vs. No. of "Processors"

- Single Machine
- 3 Machines
DEVS/RMI-Reconfigurable Framework(1)

- Supports run-time model migration across machines.
- Model states are kept persistent.
- Model structure can evolve during a distributed simulation execution.
- High-level support of run-time model re-partition.
DEVS/RMI-Reconfigurable Framework(3)
Model Partition in DEVS/RMI(1)

```java
ViewableAtomic A1 = new generator("A1","node2");
add(A1);
ViewableAtomic A11 = new generator("A11","node3");
add(A11);
```
Model Partition in DEVS/RMI(2)
Model Partition in DEVS/RMI(3)
--Dynamic model repartition

These arrow lines represent the coupling that will be removed.
Model Partition in DEVS/RMI(4)

- Increase locality whenever possible using model domain decomposition.
- Overhead incurred by dynamic repartition should be carefully evaluated.
- Load balance technique needs to be applied whenever necessary.
DEVS/RMI on large-scale model

Hilly Terrain Model

- Measure the shortest travel time for a traveller in a 2D space with hills.
- The “direction” of traveller is determined by the gradient of hill at certain point.
- Increasing the resolution results in using larger cell space.
Why use DEVS/RMI

- Express the continuous spatial model using DEVS Quantization Technique.
- 2D DEVS Cellular Space is used.

Problems:
- Cell space should be large enough to get necessary resolution, which results in “out of memory” in a single machine.
- Simulation execution time increases significantly when cell space increases.

Solution: Distributed Simulation using DEVS/RMI.
Hilly Terrain Model in DEVS
Computation Domain Decompose for Hilly Terrain Model
Measure the Travel Time

Travel Time vs. No. of Cells

Travel Time vs. No. of Hills

100 by 100 cell space
Speedup of Distributed Simulation

![Graph showing speedup of simulation with varying numbers of nodes and cells.](Image)

- 3 nodes one hill
- 6 nodes one hill
- 11 nodes one hill
- 3 nodes 100 hills
- 6 nodes 100 hills
- 11 nodes 100 hills

**Axes:**
- Y-axis: Speedup (single/multi-nodes)
- X-axis: No. of Cells
Studying Valley Fever Model

- Distributed simulation of Valley fever model, a highly dynamic 2D cell space, using DEVS/RMI
- Static model partition and “activity” based dynamic repartition are used
- Simulation execution performance is measured in terms of different computing workload
- Effects of “activity” based repartition is studied.
Valley Fever Java Model in DEVS

Valley fever model in Simview
Original model distributed simulation performance

Original model simulation execution performance in DEVS/RMI
Injecting workload

Injecting workload to partitioned cells (a sum of 1 to 100)

Injecting workload to partitioned cells (a sum of 1 to 150)
Dynamic repartition using “activity”

- “activity” metric is measured by counting the internal transitions of each individual cell.
- “activity” metric is used to repartition the model dynamically to achieve better load balance.
- High “activity” cells are assigned more computing power.
Valley Fever Model-using activity

<table>
<thead>
<tr>
<th>Using 5 computing nodes including 1 head node.</th>
<th>Static Blind Partition not considering model activities</th>
<th>Dynamic reconfiguration using “activity”</th>
<th>Performance increase by percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 by 4 cells with 400 simulation steps</td>
<td>28.124s</td>
<td>27.566s</td>
<td>1.98%</td>
</tr>
<tr>
<td>4 by 4 cells with 2000 simulation steps</td>
<td>113.977s</td>
<td>114.968s</td>
<td>-0.87%</td>
</tr>
<tr>
<td>8 by 8 cells with 400 simulation steps</td>
<td>256.49s</td>
<td>248.644s</td>
<td>3.06%</td>
</tr>
<tr>
<td>8 by 8 cells with 2000 simulation steps</td>
<td>1238.479s</td>
<td>1216.97s</td>
<td>1.73%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Using 9 computing nodes including 1 head node.</th>
<th>Static Blind Partition not considering model activities</th>
<th>Dynamic reconfiguration using “activity”</th>
<th>Performance increase by percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 by 4 cells with 2000 simulation steps</td>
<td>134.74s</td>
<td>110.49s</td>
<td>18%</td>
</tr>
<tr>
<td>8 by 8 cells with 2000 simulation steps</td>
<td>1348.17s</td>
<td>1199.87s</td>
<td>11%</td>
</tr>
</tbody>
</table>
Performance Issues

- Distributed Simulation performance of DEVS/RMI closely relates to the computation and RMI communication workload partitions.
- Load balance is a key factor.
- Locality should be increased whenever possible.
Advantages of DEVS/RMI

- DEVS/RMI provides an flexible and easy-to-use reconfigurable distributed simulation framework.
- Refactoring a distributed simulation becomes easier.
- Support run-time model structure evolution in a distributed environment.
- Achieves significant speedup when dealing with large-scale model.
DEVS in the near future

- SOA based architecture.
- Running on P2P network
- Towards to distributed execution.
- Towards running on grid.
- Keep its role as a formalized modeling framework.
References:


* http://www.acims.arizona.edu/EDUCATION/ECE575Fall03/Note/
Thank You!

Questions?
Email: mizhang@site.uottawa.ca
Google: ACIMS