INTRODUCTION TO DEVS AND FDDEVS

Lahiru Ariyananda
ACIMS
UNIVERSITY OF ARIZONA
Outline

• Introduction to DEVS
  – DEVS semantics
  – Atomic and Coupled Models

• DEVS implementation
  – Defining DEVS models within a restricted class
  – Using DEVSJAVA

• FD-DEVS
  – Features
Acknowledgement

Some of the slides presented here are from Prof. B. Zeigler’s collection
http://www.acims.arizona.edu
Recall: Tennis Player Example

PlayerA

- Send message
  - ShotA

- Receive message
  - ShotB

play, 0.5
wait, infinity

inShotB ➔ outShotA
Recall Classical DEVS Specification

• How many tuples are there in classical DEVS?

• What are the tuples?
DEVS Semantics in a nutshell

- Assume System is in state “play”
  - With No External Event
    - system will stay in play for defined time $ta(\text{play})=0.5$
  - When $ta(\text{play})=0.5$ expires
    - system outputs ShotA
  - With External event when in state “wait”
    - goes to state play
Transition Functions in a nutshell

1) Internal Transition
   • generates Output
   • The system remains in a particular state for time before
     – generate Output and Internal Transition

2) External Transition
   – happen in response to external Inputs.
   – does not generate output.
More on DEVS

• In the DEVS formalism
  • *Atomic DEVS models*
    – are basic models from which larger ones are built
    – In Tennis Example, PlayerA

• *Coupled DEVS models*
  – how atomic (or lower level coupled) models are connected together in hierarchical fashion describing the structure of system.
  – Basic models can be coupled to build more complex models.
    • How PlayerA model and PlayerB model interacts to form the system called the Tennis Match
Atomic Models

- Basic models from which larger ones are built by coupling.
- Atomic Models Contain ....

- Set of input ports and output ports
- Phase/Sigma variables.
  » State/Next transition time.
- Time advance function.
- Transition functions
  – Internal, External
- Output function.
PlayerA Atomic model mapping to DEVS

Tennis Player A Model

DEVS Mappings

\( S = \{ \text{play} \} \)
\( ta(\text{play}) = 0.5 \)

Output Function : \( \lambda(\text{play}) = \text{ShotA} \)

\( Y = \{ \text{outShotA} \} \)

\( S = \{ \text{wait} \} \)

Internal Transition : \( \delta y(\text{play}) \rightarrow \text{wait} \)
\( ta(\text{wait}) = \text{infinity} \)

\( X = \{ \text{inShotB} \} \)

External Transition: \( \delta x (\text{wait}, \text{shotB}) \rightarrow \text{play} \)

\( ta(\text{play}) = 0.5 \)
\( ta(\text{wait}) = \text{infinity} \)
PlayerA Atomic model mapping to DEVS in summary…

Note: below mapping strictly does not follow the DEVS mathematical specification structure. But will provide an intuitive idea needed for modeling

- $X = \{\text{inShotB}\}$
- $Y = \{\text{outShotA}\}$
- $S = \{\text{play, wait}\}$
- $ta(\text{play}) = 0.5$, $ta(\text{wait}) = \infty$
- External Transition: $\delta x(\text{wait, shotB}) \rightarrow \text{play}$
- Internal Transition: $\delta y(\text{play}) \rightarrow \text{wait}$
- Output $\lambda(\text{play}) = \text{ShotA}$
Tennis Match Coupled Model

PlayerA
  Atomic Model

PlayerB
  Atomic Model

TennisMatch

TennisPlayerA
  play
  \( \sigma = 0.500 \)

TennisPlayerB
  wait
  \( \sigma = \infty \)
Coupled Models

Components couplings
- Internal Couplings
- External Input Couplings
- External Output Couplings

Tennis Match - Coupled Model

PlayerA
Atomic Model

PlayerB
Atomic Model

Couplings:
- inShotA -> outShotB
- outShotA -> inShotB
- inShotA -> outShotB
- outShotB -> inShotA
Coupled Models

• Basic models may be coupled in the DEVS formalism
  => form a **Coupled model**.

• A coupled model connects several component models together
  => New larger model.

• This new model can be employed as a component in a larger coupled model
  => giving rise to **hierarchical construction**.

• A coupled model contains the following information:
  – the set of component models
  – the set of input ports through which external events are received
  – the set of output ports through which external events are sent
  – set of possible Internal couplings within a coupled model
Atomic Models, Coupled Models and Hierarchical Coupling

+ coupling

Hierarchical Coupling
DEVS as M&S Framework and its Computational implementations

- **DEVS structural model is**
  
  i) expressed as Mathematical Formalism

  • provides a means of specifying a mathematical object called a System with a time base, inputs, states, and outputs, and functions for determining next states and outputs, given current states and inputs.

  ii) Implemented as a set of Computational instructions, such as in Java or C++
Understanding DEVS – Define DEVS models within a restricted class

http://www.cs.gsu.edu/DEVSTutorial/
Implementing PlayerA Atomic Model in DEVSJAVA

Understand Devs formalism to map your model into DEVS structure

Implement the model in DEVSJAVA or using another DEVS implementation environment

Simulate

SimView
Visual Simulator environment for DEVSJAVA

Text Based Simulation
DEVSJAVA

- Is based on DEVS formalism
  - an Object-Oriented Modeling and Simulation environment
- Includes a Simulation Viewer for animating simulation(simView)
- Provides Java base classes for implementing Atomic and Coupled models.
- Allows adding ports, phases/states, and internal, external and confluent functions.

Introduction to DEVS Modeling and Simulation with JAVA by B.P Zeigler and Hessam S. Sarjoughian, can be found at http://www.acims.arizona.edu/EDUCATION/education.shtml#tutorials
Example DEVS code for the Tennis PlayerA atomic model

```java
public class TennisPlayerA extends ViewableAtomic{

double Infinity = INFINITY;

public TennisPlayerA(){
    this("TennisPlayerA");
}

public TennisPlayerA(String nm){
    addInport("inShotB");
    addTestInput("inShotB",new entity());
    addTestInput("inShotB",new entity(),1);
    addOutport("outShotA");
}

public void initialize(){
    holdIn("play",0.5);
}

public void deltint(){
    if(phaseIs("play")){
        passivateIn("wait");
    }else{
        passivate();
    }
}

public void deltext(double e, message x){
    Continue(e);
    for(int i=0; i<x.getLength(); i++){  
        if(this.messageOnPort(x, "inShotB", i)){
            if(phaseIs("wait")){
                holdIn("Play",0.5);}
            }
        }
    }
}

public message out(){
    message m = new message();
    if(phaseIs("play")){
        m.add(makeContent("outShotA", new entity("ShotA")));
        return m;}
}

Note: You do not need to know this code. Showing as a code example only.

Note for the curious: passivate() is a shortcut to say stay in a state for ever or time infinity
```
Pros and Cons of DEVS

- Reduced development times
- Improved testing => higher quality models
- Improved maintainability
- Easy experimentation
- Verification/Validation facilities
- Models/frames stored in repositories with information to enable reuse.
- Discrete-Event formalism: time advances continuously as it does in the real world so that events can happen at any given time

- Modeling and Simulation Engineering is not a well established discipline
  - Many modelers lack the right experience

- Building DEVS models is not trivial
  - Most commercial tools are specialized
- DEVS more generic and harder compared to commercial tools
  - Commercial tools support only certain applications

- Extensive Training required to learn DEVS
  - System concepts
  - Object Orientation
  - This hurdle can be overcome in FDDEVS as we will see next

part resource: Dr Gabriel A. Wainer, Carleton University, Canada
FD-DEVS

- **Finite Deterministic-DEVS** is a sub-class of DEVS
- **From the user perspective**
  - Automation from a restricted Natural Language to DEVSJAVA mapping
    - Coupled models via automated port matching
  - Way to learn DEVS
  - Supplies a skeleton that can be extended to full DEVS as needed
  - Automated test frame development and Analysis
  - Not sufficient for Industrial strength applications

- Will reduce drastically learning curves, cut down on model development time and makes training easier
Finite Deterministic DEVS: FD-DEVS

FDDEVS = \langle \text{incomingMessageSet, outgoingMessageSet, StateSet, TimeAdvanceTable, InternalTransitionTable, ExternalTransitionTable, OutputTable} \rangle

where

- \text{incomingMessageSet, outgoingMessageSet, StateSet} \text{ are finite sets}
- \text{TimeAdvanceTable: StateSet} \rightarrow \mathbb{R}_{0,\infty}^+ \text{ (the positive reals with zero and infinity)}
- \text{InternalTransitionTable: StateSet} \rightarrow \text{StateSet}
- \text{ExternalTransitionTable: StateSet} \times \text{incomingMessageSet} \rightarrow \text{StateSet,}
- \text{OutputTable: StateSet} \rightarrow \text{the set of subsets of outgoingMsgSet}

Natural Language For FDDEVS

- to start hold in PHASE for time SIGMA
- when in PHASE and receive MSG go to PHASE’
- eventually
- hold in PHASE for time SIGMA
- after PHASE then output MSG
- from PHASE go to PHASE’

Semantics defined by mapping into DEVS

http://acims.arizona.edu/EDUCATION/education.shtml#turtorials
FDDEVS : TennisPlayerA from NL to DEVSSJAVA mapping

to start hold in play for time 0.5! after play output ShotA! from play go to wait!

hold in wait for time Infinity! when in wait and receive ShotB go to play!
Automated Test Frame Development

Legend:

= automated
References

- DEVSJAVA software, [http://www.acims.arizona.edu](http://www.acims.arizona.edu)
- DEVS Today: Recent Advances in Discrete Event-Based Information Technology Zeigler B.P, [http://www.acims.arizona.edu](http://www.acims.arizona.edu)
- *Introduction to DEVS Modeling and Simulation with JAVA*, Zeigler, B.P. and H.S. Sarjoughian, [www.acims.arizona.edu](http://www.acims.arizona.edu) 2001
- An Introduction to Modeling and Simulation with DEVS, Dr Gabriel A. Wainer, Carleton University, Canada.
- Object-Oriented Simulation/Discrete Event Models Lecture notes, Dr Moon Hwang, [http://www.acims.arizona.edu](http://www.acims.arizona.edu)
- [http://www.cs.gsu.edu/DEVSTutorial/](http://www.cs.gsu.edu/DEVSTutorial/)
- [http://acims.arizona.edu/EDUCATION/education.shtml#turtorials](http://acims.arizona.edu/EDUCATION/education.shtml#turtorials)