Outline

• Review of last class
• DEVSJAVA common classes
• JEV$JAVA$A3.0 and Simview
• An carwash example – work together
  • Note: all the code provided in the PPT slides are “pseudo” code. To develop your own model, you should follow the examples provided in the DEVSJAVA source code packages.
DEVS Atomic Model

Elements of an atomic model:

- input events
- output events
- state variables
- state transition functions
  - External transition
  - Internal transition
  - Confluent transition
- output function
- time advance function
Internal Transition / Output Generation

Time advance

Make a transition

Generate output

Using the internal transition function

Using the output function

output
Response to External Input

Make a transition using the \textit{external} transition function

Input

ElapsedTime

Time Advance
Response to Simultaneous External Input and Internal Event

Make a transition

 elapsed time

Time advance

Generate output

using the confluent transition function

input

output
DEVS Coupled Model

Elements of coupled model:

- Components
- Interconnections
  - Internal Couplings
  - External Input Couplings
  - External Output Couplings
Coupling in Action

Coupling (internal)

Output port
Input port

external

state

internal

time advance

output
Basic Atomic Variables

**phase**

**sigma**: the scheduled remaining time in the current phase

**elapse time**: the time that elapsed in the current state
Basic Atomic Methods

**public double ta()**: the time advance function; returns the value of sigma for atomic models

**public message out()**: the output function; releases the message just before an internal transition

**public void deltint()**: the internal transition function

**public void deltext(double e, message x)**: the external transition function

**public void deltcon(double e, message x)**: the confluent transition function

**public void Continue(double e)**: subtract e from sigma; use to retain the same time of next event after an external event

**public void holdIn(String phase, double sigma)**: set the phase and sigma values as given in the arguments

**public void passivateIn(String phase)**: set the phase as given in the argument and sigma to INFINITY

**public void passivate()**: set the phase to passive and sigma to INFINITY

**public boolean phasels(String phase)**: return true if the current phase equals the argument

---

Sigma: time left to the next planned internal event

e: time elapsed from the last event
Other commonly used methods

- The following methods are also commonly used in an atomic model
  - void Initialize()
  - boolean messageOnPort(message x, String p, int i)
  - ContentInterface makeContent(PortInterface port, EntityInterface value)

Below is a typical code segment used in deltext(e, x)

```java
for (int i = 0; i < x.getLength(); i++)
    if (messageOnPort(x, "in", i)) {
        entity job = x.getValOnPort("in", i);
        System.out.println("my current job is: "+ job.getName());
        holdIn("busy", 20);
    }
```
realDevs – a template for handling simple real valued data

realDevs – an example of working with real number inputs, states and outputs

Its behavior – wait for an input, store it, and output it immediately

realDevs

output

realVar

passive

wait in phase passive until receive input

receive a number as input and store it in realVar

Transition to phase "output"

Write your own DEVSJAVA code!!
realDevs – implementation in DEVSJAVA
realDevs illustrates how to receive a real number, store it, manipulate it, and output the result

```java
public class realDevs extends ViewableAtomic{
    protected double realVar;

    public realDevs(String nm){
        super(nm);
        addInport("in");
        addOutport("out");
        addRealTestInput("in",10);
        addRealTestInput("in",10,5);
    }
    public realDevs(){
        this("realDevs");
    }
    public void initialize(){
        realVar =0;
        super.initialize();
        passivate();
    }
    public void deltint(){
        if (phaseIs("output"))
            passivate();
    }
    public void deltcon(double e,message x){
        deltint();
        deltext(0,x);
    }
    public message out(){
        if (phaseIs("output"))
            return outputRealOnPort(realVar,"out");
        else return new message();
    }
    public static int signOf(double x){
        if (x == 0) return 0;
        else if (x > 0) return 1;
        else return -1;
    }
    public static double inv(double x){
        if (x == 0) return Double.POSITIVE_INFINITY;
        else if (x >= Double.POSITIVE_INFINITY ) return 0;
        else return 1/x;
    }
}
```

Some useful methods that can be used outside the class:

- `signOf(double x)`: This method returns 0 if `x` is 0, 1 if `x` is positive, and -1 if `x` is negative.
- `inv(double x)`: This method returns the reciprocal of `x` if it is not zero and positive infinity if `x` is positive infinity.

Dr. Xiaolin Hu
RealDevs (continued) – illustrating how to specify the response to external input

```
if something is on the "in" port{
    treat it as a real and store it in realVal;
    phase = "output"
    sigma = 0
}
otherwise
    sigma = sigma - e
```

```
SomethingOnPort("in")
holdIn("output", 0)
continue()
realVar = getRealValue OnPort("in")
```
RealDevs -- illustrating how to use DEVS primitive methods in Internal Transition/Output Generation

Time advance = 
\[ ta("output") = 0 \]
\[ ta("passive") = \infty \]

Generate output

Make a transition
phase = "passive"
sigma = "infinity"

passivate()

outputRealOnPort(realVar, "out")

output = realVar
Fire-once Neuron

public void initialize(){
    super.initialize();
passivateln("receptive");
}

public void deltext(double e, message x){
    Continue(e);
    if (phases("receptive")&&!somethingOnPort(x,"in"))
        holdIn("fire", fireDelay);
}

public void deltint(){
    // if (phases("fire"))
    passivateln("refract");
}

public message out(){
    // if (phases("fire"))
    return outputNameOnPort("pulse","out");
}
Class pulseGenr -- generates pulses whose size and frequency can be specified

```java
public void initialize(){
    super.initialize();
    holdIn("active",interPulseTime);
}

public void deltint(){
    holdIn("active",interPulseTime);
}

public message out(){
    return outputRealOnPort(pulse,"out");
}

public void initialize(){
    super.initialize();
    holdIn("active",interPulseTime);
}

public void deltint(double e,message x){
    Continue(e);
    if (somethingOnPort(x,"setInterPulseTime"){
        interPulseTime = getRealValueOnPort(x,"setInterPulseTime");
        if (interPulseTime > 0)
            holdIn("active",interPulseTime);
        else passivate();
    } else if (somethingOnPort(x,"setSize"){
        size = getRealValueOnPort(x,"setSize");
        holdIn("active",interPulseTime);
    } else if (somethingOnPort(x,"start")
        holdIn("active",interPulseTime);
    else if (somethingOnPort(x,"stop")
        passivate();
```
public void initialize()
function
super.initialize();
holdIn("active",0);
}

public void deltext(double e, message x)
if (somethingOnPort(x,"start"))
  holdIn("first",firstDuration);
else if (somethingOnPort(x,"stop"))
  passivate();
}

public void deltint()
if (phaseIs("active"))
  holdIn("first",firstDuration);
else if (phaseIs("first"))
  holdIn("second",secondDuration);
else if (phaseIs("second"))
  holdIn("third",thirdDuration);
else //if (phaseIs("third"))
  passivate();

public message out()
if (phaseIs("active"))
  return outputRealOnPort(firstOutput,"out");
else if (phaseIs("first"))
  return outputRealOnPort(secondOutput,"out");
else if (phaseIs("second"))
  return outputRealOnPort(thirdOutput,"out");
else //if (phaseIs("third"))
  return outputRealOnPort(0,"out");

What about both start and stop arrives at the same time?

Dr. Xiaolin Hu
Demonstrate coupled DEVS models:
GenDevsTest HierarModels
Discussion of Modeling

• The classroom example
  – Two rooms connected by a door
  – port: door  message: people
  – a door that can pass only one person?
  – A person with his/her behavior?

• A network example
  – Two computers connected by network
  – Network delay?
DEVSJAVA Main Classes

• Chapter 3
  – two groups of classes
  – Atomic class
  – Digraph class
public interface EntityInterface{
    public String getName();
    public Object equalName(String name);
}

public class entity
    extends Object implements EntityInterface{
    protected String name;

    public entity(){name = "anEntity";}
    public entity(String nm){name = nm;}
    public boolean eq(String nm){
        return getName().equals(nm);
    }
    public Object equalName(String nm){
        if (eq(nm)) return this;
        else return null;
    }
    //overrides pointer equality of Object
    public boolean equals(Object o){
        if (!(o instanceof entity)) return false;
        else return eq(((entity)o).getName());
    }
    public String toString(){
        return getName();
    }
    //overrides hashCode of Object
    /*
    public int hashCode(){
        return name.hashCode();
    }
    */
    public String toString(){
        return name.hashCode();
    }
}
Major Class Relationships

- entity
  - devs
    - atomic
      - ViewableAtomic
        - User Defined Atomic Model
    - digraph
      - ViewableDigraph
        - User Defined Coupled Model
    - intEnt
    - doubleEnt
      - User Defined Classes, e.g., carEntity
public message out() {
    message m = new message();
    if (phaseIs("busy")) {
        m.add(makeContent("out", job));
    }
    return m;
}
Sending/Receiving/Interpreting Messages

how to use casting to receive instances of arbitrary entity subclasses

coupled model

\[
\begin{array}{c}
\text{A} \\
\text{double}
\end{array}
\quad \xrightarrow{\text{out}} \quad \\
\quad \xrightarrow{\text{in}} \\
\begin{array}{c}
\text{B} \\
\text{double}
\end{array}
\]

coupling: (A,"out",B,"in")

entity

doubleEnt

doubletEnt(double)

casting the received entity down to the doubleEnt subclass

deltex(double e,message x){
    for (int i = 0; i < x.getLength(); i++)
        if (messageOnPort(x, "in", i)) {
            entity val = x.getValOnPort("in", i);
            doubleEnt f = (doubleEnt)val;
            double v = f.getv();
        }
}

public message out( ){
    message m = new message();
    m.add( makeContent("out", new doubleEnt(1.2));
    return m;}

Dr. Xiaolin Hu
entity job;

public void initialize(){
    passivate();
}

public void deltext(double e, message x){
    Continue(e);
    if (phaseIs("passive"))
        for (int i = 0; i < x.getLength(); i++)
            if (messageOnPort(x, "in", i)) {
                job = x.getValOnPort("in", i);
                holdIn("busy", 20);
            }
}

public void deltint() {
    passivate();
}

public message out() {
    message m = new message();
    if (phaseIs("busy")) {
        m.add(makeContent("out", job));
    }
    return m;
}
Avoiding Some Common Formalism Violations

\[ X^b \text{ is a bag of inputs whose elements are in } X, \]

\[ \delta_{\text{ext}}: S \times X^b \rightarrow S \]

This means that the state after receiving a bag of inputs is uniquely determined by the current state, the elapsed time, and in particular the bag of inputs. Since a bag is an unordered collection, the result cannot depend on the order used in examining the inputs. In the example on the left, the order of examining the bag \{"a","b"\} matters since if "a" is in the first content to be examined the result is to do A, while if "b" is the first content, the result is to do B. The example on the right completely examines the bag for occurrence of "a" and then occurrence of "b". The results always do A first and B next for the bag \{"a","b"\}.

```java
public void deltext(double e, message x) {
    Continue(e);
    for (int i = 0; i < x.getLength(); i++) {
        if (messageOnPort(x, "a", i)) <do A>
    } else
    if (messageOnPort(x, "b", i)) <do B>
}
```

result is undefined

```java
public void deltext(double e, message x) {
    Continue(e);
    for (int i = 0; i < x.getLength(); i++){
        if (messageOnPort(x, "a", i))
            <do A>
    } for (int i = 0; i < x.getLength(); i++)
        if (messageOnPort(x, "b", i))
            <do B>
}
```

result is uniquely defined
Avoiding Some Common Formalism Violations

\[ \lambda: S \rightarrow Y \]

This means that the output function does not have an effect on the state – it can’t change it, it can only look at it. The following violates this requirement.

```java
public message out( )
{
message m = new message();
if (phaseIs("transmit")){
m.add(makeContent("out", new entity("packet " + count+';'+destination)));
count = count + 1;
}
}
```

Here the intent is that the state variable, count is incremented by the call to the output function. But DEVS simulator is guaranteed only to use the return result of the call as specified in the DEVS simulation protocol, not to obey the side-effect of changing the count. The correct way to update the count is in the internal transition function which is called immediately after the output function:

```java
public void int( )
{
if (phaseIs("transmit"))
count = count + 1;
}
```
Sending/Receiving/Interpreting Messages (cont’d)

multiple copies of an object are needed to avoid hard-to-find dependencies.

A coupled model

Suppose A sends its instance of job directly to B:

```java
public message out( ){
message m = new message();
m.add( makeContent("out", job))
return m;}
```

and B stores it as its instance:

```java
deltext(double e,message) x){
if (somethingOnPort(x,"in"){
myJob  = getEntityOnPort(x,"in");
}
```

This instance of job is now shared in common by both A and B – if either makes a change to its state, then the other will also be subject to it.

For example, if B does:

```java
job.update(10);
```

then the instance at A will be similarly altered.

This can lead to mysterious effects (similar to quantum entanglement) where components of a model can influence each other outside of their interface couplings.

It is difficult to trace this kind of non-modularity.

The right way:

B stores a copy of its input:

```java
deltext(double e,message) x){
if (somethingOnPort(x,"in"){
job temp =
gentityOnPort(x,"in");
myJob = temp.copy();
```

where copy() is a method you define to create a new instance of job and give it values of an existing one.

The cure is simple: create a new instance as a local copy of an entity if it is to be altered (this happens automatically when using toString() and toObject(), see chap. 12)
SimView: Using `toString()` to display entities contents, and `getTooltipText()` to show component state information.

The modeler must define `toString()` for the simulator to use polymorphically. For example:

```java
public String toString() {
    return doubleFormat.niceDouble(x) + "," + doubleFormat.niceDouble(y);
}
```

```java
public String getName() {
    return toString();
}
```

```java
public message out() {
    message m = new message();
    m.add(makeContent("out", new vect2DEnt(x, y)));
    return m;
}
```

SimView displays the value returned by `toString()` in the moving window that represents transmitting a message.

SimView displays the value returned by `getTooltipText()`:

```java
public String getTooltipText() {
    return super.getTooltipText() + 
    "\n" + "Cost: " + linkCost;
}
```

Dr. Xiaolin Hu
DEVSJAVA3.0 and Simview

What are the packages?
How to use it?
How to add a new package in SimView?
An carwash example

• Chapter 4: exercise 1 part a  -- car wash
• To simplify the problem, let’s first assume the system will reject any incoming cars and trucks if it is busy
• Let’s work together!!
An carwash example

1. We first assume the system rejects any incoming cars/trucks if busy.
2. Test the model: addTestInput( )
3. Test the model: create a car generator and couple it to the carWashCenter
4. Add random numbers in generators
5. Create a truck generator and add to the system
6. Add queue
7. Display the queue size in simulation. -- the getTooltipText();
8. Each car/truck has its own processing time and other properties such as price and priority. --- add a vehicleEntity class.
9. Show the processing time of each car/truck. – the toString() method.
10. Add transducer. Introduce the getSimulationTime() method.
11. Make the whole system generate an output.
12. Test the model without SimView.
Exercise

• The car wash center can have at most 5 cars waiting in the queue.
  – Check the size of the queue
  – Use different phases, e.g., first, second…..

• The car wash time depends on the number of cars in the queue. For example, the car wash time $t = T \times (1/2)^n$, where $n$ is the number of cars in the queue.

• There are two service lanes.
  – Atomic model
  – Coupled model, similar to the simpArc example
Homework

• Read chapter 4 to learn some examples.
• Description of the hands-on project is posted on class webpage
• Form the project group and decide your project topic. Then email your group and project information to me by next class (March 10).
  – Type
  – A brief description of the project