Develop DEVS Models using DEVSJAVA

Dr. Xiaolin Hu
Outline

• Review of how DEVS models work
• DEVSJAVA Modeling: the first pass
  – A first look at the DEVSJAVA environment.
  – Basic DEVSJAVA methods and Some sample models
• DEVSJAVA Modeling: more details
  – The GenDevs Architecture
  – Message passing in DEVS
• A modeling example: the CarWash example
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
How an Atomic Model Work

- **Input**
- **State**
  - External
  - Internal
  - Time advance
- **Output**
  - Hold for some time
  - Send an output

- **Actions**
  - Make a transition (external)
  - Make a transition (internal)

- **Connections**
  - X \(\rightarrow\) \(S,e\) \(\rightarrow\) Y
An DEVS Coupled Model

Coupling (internal)

Output port

Input port

State

output

external

internal

time advance

output

external

internal

time advance
DEVJAVA Modeling: the first pass
A first look at the DEVSJAVA environment

- Set up the DEVSJAVA project in Eclipse
- Packages structure
- How to develop models – programming
- How to run simulations – the SimView Interface
  - SimView configure
**Basic Atomic Variables**

**phase**

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

**elapse time:** the time that elapsed in the current state
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

Basic Atomic Methods

public double ta(){return sigma; }  

public message out(){return new message();}  
    //override

public void deltint() {} //override

public void deltext(double e,message x) {} //override

public void deltcon(double e,message x){}//default
deltint();
deltext(0,x);
}

public void Continue(double e){
    if (sigma < INFINITY)
        sigma = sigma - e;
}

public void holdIn(String phase, double sigma) {
    this.phase = phase;
    this.sigma = sigma;;
}

public void passivateIn(String phase) {holdIn(phase,"passive");}

public boolean phasels(String phase){
    return this.phase.equals(phase);
}
Other commonly used methods

- The following methods are defined in devs, but commonly used by 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 in `deltext(double e, message x)`

```java
for (int i = 0; i < x.getLength(); i++)
    if (messageOnPort(x, "in", i)) {
        entity job = x.getValOnPort("in", i);
        // ..... process job
        holdIn("busy", 20);
    }
```
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;
}
public boolean somethingOnPort(message x,String port) : is there a value on the given port, e.g., somethingOnPort(x, "in")

public double getEntityOnPort(message x,String port): get the entity on the given port, e.g., getEntityOnPort(x,"in")

public double getIntValueOnPort(message x,String port): get the integer value on the given port, e.g., getIntValueOnPort(x,"in") Note – assumes that the port only receives integer values

public double getRealValueOnPort(message x,String port): get the real (double) value on the given port, e.g., getRealValueOnPort(x,"in") Note – assumes that the port only receives real values

public double sumValuesOnPort(message x,String port): sum up all the real values on the given port, e.g., sumValuesOnPort(x,"in") Note – assumes that the port only receives real values

public double getNameOnPort(message x,String port): get the name (string) value on the given port, e.g., getNameOnPort(x,"in") Note – assumes that the port only receives name values

public message outputNameOnPort(String nm,String port): output a name (string) on the given port, e.g., outputNameOnPort("hello","out")

public message outputRealOnPort(double r,String port): output a real value (double) on the given port, e.g., outputRealOnPort(5,"out")

public message outputIntOnPort(int r,String port): output an integer value on the given port

public void addNameTestInput(String port,String name,double elapsed): add a test which inputs the given name on the given port after an elapsed time, e.g., addNameTestInput("in","hello",5)

public void addRealTestInput(String port,double value,double): add a test which inputs the given value on the given port after an elapsed time, e.g., addRealTestInput("in",10,5)
**friendlyAtomic Methods - Relation to Full-up Atomic Methods**

Besides being easy to use, these definitions suggest how to use the full-up methods

```java
public boolean somethingOnPort(message x, String port){
    for (int i=0; i< x.getLength();i++)
        if (messageOnPort(x, port, i))
            return true;
    return false;
}

public entity getEntityOnPort(message x, String port){
    for (int i=0; i< x.getLength();i++)
        if (messageOnPort(x, port, i))
            return x.getValOnPort(port, i);
    return null;
}

public double getRealValueOnPort(message x, String port){
    doubleEnt dv = (doubleEnt)x.getEntityOnPort(port);
    return dv.getv();
}

public int getIntValueOnPort(message x, String port){
    intEnt dv = (intEnt)x.getEntityOnPort(port);
    return dv.getv();
}

public double sumValuesOnPort(message x, String port){
    double val = 0;
    for (int i=0; i< x.getLength();i++)
        if (messageOnPort(x, port, i))
            val += dv.getv();
    return val;
}
```

This only puts one value on one port

Note how this is a strong restriction on full-up Devs: it pulls off only one value on the port regardless of how many there may be.

This shows how multiple values on a single port can be captured. This case does so by summing them up.

This allows accumulating any number of contents (port, value pairs) in a message

```java
public message outputNameOnPort(String nm, String port){
    message m = new message();
    m.add(makeContent(port, new entity(nm)));
    return m;
}

public message outputRealOnPort(double r, String port){
    message m = new message();
    m.add(makeContent(port, new doubleEnt(r)));
    return m;
}

public message outputRealOnPort(message m, double r, String port){
    m.add(makeContent(port, new doubleEnt(r)));
    return m;
}

public void addNameTestInput(String port, String name, double elapsed){
    addTestInput(port, new entity(name), elapsed);
}

public void addNameTestInput(String port, String name){
    addTestInput(port, new entity(name), 0);
}

public void addPortTestInput(String port, double elapsed){
    addTestInput(port, new entity(), elapsed);
}

public void addRealTestInput(String port, double value, double elapsed){
    addTestInput(port, new doubleEnt(value), elapsed);
}
```
Class `pulseGenr` -- generates pulses whose size and frequency can be specified

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

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

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

What happens if both start and stop arrive at the same time?

Note: stop and start do not remember the time already elapsed in last period; this can be done by employing the elapsed time, e.
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

wait in phase passive until receive input
receive a number as input and store it in realVar
Transition to phase “output”
wait for 0 seconds before outputting.
realDevs – implementation in DEVSJAVA

realDevs illustrates how to receive a real number, store it, manipulate it, and output the result.

declare an instance (state) variable to hold the received input

add input and output ports

add test inputs for use in the viewer

specify how the variable will be initialized and re-initialized

specify how to get and manipulate the input

specify what to do after receiving and outputing

transition then get input

specify how to generate output

specify the tooltip contents for the viewer

how to simulate a model outside the viewer

some useful methods that can be used outside the class

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 deltext(double e, message x){
        Continue(e);
        if (phaseIs("passive") && somethingOnPort(x,"in")){
            realVar = getRealValueOnPort(x,"in");
            if (signOf(realVar) == 1)
                realVar = inv(realVar);
            holdIn("output",0);
        }
    }
    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 String getTooltipText(){
        return super.getTooltipText() + "realVar:" + realVar; }
    public static void main(String args[]){
        realDevs re = new realDevs("real");
        re.initialize();
        atomicSimulator s = new atomicSimulator(re);
        s.simInject(0,"in",new doubleEnt(10));
        s.simulate(2);
        //should print
        //Time: 0.0 ,input injected:
        //port: in value: 10.0
        //port: out value: 10.0
        //Terminated Normally at ITERATION 2 ,time: Infinity
    }
    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;
    }
}
RealDevs -- illustrating how to use DEVS primitive methods in Internal Transition /Output Generation

Time advance =
\[ \tau_a(\text{"output"}) = 0 \]
\[ \tau_a(\text{"passive"}) = \infty \]

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

passivate()
RealDevs (continued) – illustrating how to specify the response to external input

Dr. Xiaolin Hu
Fire-once Neuron

This neuron fires exactly once after a fixed delay while ignoring all subsequent inputs.

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

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

public void deltint(){
    //if (phaseIs("fire"))
    passivateIn("refract");
}

public message out(){
    //if (phaseIs("fire"))
    return outputNameOnPort("pulse","out");
}
public void initialize(){
    super.initialize();
    holdIn("active",0);
}

public void deltext(double e,message x){
    Continue(e);
    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
        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");
}
A diver drops immediately to 60 ft depth, stays there for 20 min., then rises to 40 ft, depth staying 20 min., and finally spends 10 minutes at 15 ft before rising to the surface. Modify the basicGenr to represent this dive plan. Your model should output the new depth to which the diver goes at each stage.
Coupled Model Example:
using a basic generator to dynamically set the period of a pulse generator

```java
public class boxCar extends ViewableDigraph{

public boxCar(){
    super("boxCar");

    ViewableAtomic basic =
        new basicGenr("basic",15,20,14, 2,1,1000);
        // d1,d2,d3 ,p1,p2,p3

    add(basic);

    ViewableAtomic pg = new pulseGenr("pg");

    add(pg);

    addCoupling(basic,"out",pg,"setInterPulseTime");
}
```

DEVSJAVA Modeling: more details
The GenDevs Architecture

GenDevs
GenCol
java.util, java.lang

DEVS Models
Simulators and Interfaces

• Bag, Relation, Function
• ensembleCollection classes and interfaces

See the GenDevsDoc file

• Collection classes and interfaces
• Threads
• Sockets
• reflect
# Collections, Maps, Relations

<table>
<thead>
<tr>
<th>Table</th>
<th>Defining Property</th>
<th>Useful For</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Collection</strong></td>
<td>- Indefinite size</td>
<td>- Variable sized collections</td>
</tr>
<tr>
<td>List</td>
<td>- Indexed elements</td>
<td>- sequencing</td>
</tr>
<tr>
<td></td>
<td>- insert/remove anywhere</td>
<td>- basis for queues/stacks</td>
</tr>
<tr>
<td>Set</td>
<td>- admission based on equality (no duplicates)</td>
<td>- unique tracking of object occurrences</td>
</tr>
<tr>
<td>Bag</td>
<td>- no admission criteria</td>
<td>- word frequency counts</td>
</tr>
<tr>
<td></td>
<td>- multiplicities are counted</td>
<td></td>
</tr>
<tr>
<td>Map</td>
<td>- one-one correspondence (keys to values)</td>
<td>- dictionary (one meaning per word)</td>
</tr>
<tr>
<td>Relation</td>
<td>- many-many correspondence</td>
<td>- dictionary (multiple meanings per word)</td>
</tr>
</tbody>
</table>
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 int hashCode(){
    return name.hashCode();
}
GenDevs Collection Classes

- Relation
- Bag
- entity
  - intEnt
  - doubleEnt
  - DrawEntity
  - User Defined Classes
DEVS Interfaces

interface IODevs {
    public void addInport(String portName);
    public void addOutport(String portName);
    public ContentInterface makeContent(PortInterface port, EntityInterface value);
    public boolean messageOnPort(MessageInterface x, PortInterface port, ContentInterface c);
}

interface basicDevs {
    public void deltext(double e, MessageInterface x);
    public void deltcon(double e, MessageInterface x);
    public void deltint();
    public MessageInterface Out();
    public double ta();
    public void initialize();
    public void showState();
}

interface coupledDevs {
    public void add(basicDevs d);
    public void addCoupling(basicDevs src, String p1, basicDevs dest, String p2);
    public basicDevs withName(String nm);
}

interface atomicDevs {
    public void Continue(double e);
    public void passivate();
    public void passivateIn(String phase);
    public void holdIn(String phase, double time);
    public void holdIn(String phase, double time, Activity a);
    public boolean phaseIs(String phase);
}

interface IOBasicDevs {
    public void deltext(double e, MessageInterface x);
    public void deltcon(double e, MessageInterface x);
    public void deltint();
    public MessageInterface Out();
    public double ta();
    public void initialize();
    public void showState();
}

interface AtomicInterface {
    public void Continue(double e);
    public void passivate();
    public void passivateIn(String phase);
    public void holdIn(String phase, double time);
    public void holdIn(String phase, double time, Activity a);
    public boolean phaseIs(String phase);
}

interface Coupled {
    public void add(basicDevs d);
    public void addCoupling(basicDevs src, String p1, basicDevs dest, String p2);
    public basicDevs withName(String nm);
}

interface DevsInterface {
    public void Continue(double e);
    public void passivate();
    public void passivateIn(String phase);
    public void holdIn(String phase, double time);
    public void holdIn(String phase, double time, Activity a);
    public boolean phaseIs(String phase);
}
DEVS-Canonical Implementation

Message Handler

coupledDevs

IODevs

basicDevs

atomicDevs

IOBasicDevs

Atomic

EntityInterface

devs

coupled

entity

digraph

atomic
Port, Content and Message

PortInterface

ContentInterface

MessageInterface

content

message

port

value

ensembleBag

ensembleCollection

Collection

Dr. Xiaolin Hu
Message Interface/Using EnsembleBag

```java
public interface MessageInterface extends Collection{
    public boolean onPort(PortInterface port, ContentInterface c);
    public Object getValOnPort(PortInterface port, ContentInterface c);
    public void print();
    /* examples of using ensembleBag approach */
    //public ensembleBag getPortNames();
    //public ensembleBag valuesOnPort(String portName);
}
```

```java
ensembleBag b = x.getPortNames();
if (b.size()>= 2) //both stop and start arrive
    holdIn("active",10);
else if (b.contains("stop")){
    if (phaseIs("active"))
        passivate();
}
else if (b.contains("start")){
    if (phaseIs("passive"))
        holdIn("active",100);
}
```

```java
public void deltext(double e,message x){
    Continue(e);
    for (int i=0; i< x.getLength();i++)
        if (messageOnPort(x,"in",i))
            entity ent =  x.getValOnPort("in",i);
            passivate();
}
```
Sending/Receiving/Interpreting Messages
how to use casting to receive instances of arbitrary entity subclasses

coupled model

\[ \text{coupling: (A,"out",B,"in")} \]

double entity
doubleEnt(double)
getv() \rightarrow \text{double}

doubleEnt

deltetEnt(double)
getv() \rightarrow \text{double}

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

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

This assumes there is only one message on port in.
casting the received entity down to the doubleEnt subclass

deltet(double e,message x){if
(somethingOnPort(x,"in"){
entity val = getEntityOnPort(x,"in");
doubleEnt f = (doubleEnt)val;
double v = f.getv();}}

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;
}
Sending/Receiving/Interpreting Messages (cont’d)

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

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 =
getEntityOnPort(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 an entity's 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 ) +
           "\n" + doubleFormat.niceDouble(y);
}
```

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

```java
public String getTooltipText()
{
    return super.getTooltipText() +
           "\n" + "Cost: " + linkCost;
}
```
Using `toString()` and `toObject()` to facilitate deploying models in distributed simulation

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

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

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

The modeler also needs to define `toObject()` and use this method in decoding the message.

```java
public static vect2DEnt toObject(String nm)
{
    int commaIndex = nm.indexOf(",");
    String xs = nm.substring(0,commaIndex);
    String ys = nm.substring(commaIndex+1,nm.length());
    return new vect2DEnt(Double.parseDouble(xs),Double.parseDouble(ys));
}

public static vect2DEnt toObject(entity ent)
{
    return toObject(ent.getName());
}
```

DEVS simulator uses `toString()` to encode the entity as a String which is sent across the wire.

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

```java
public void deltext(double e ,message x)
{
    for (int i=0; i< x.getLength();i++)
    {
        if (messageOnPort(x,"in",i)) {
            entity en = x.getValOnPort("in",i);
            position = vect2DEnt.toObject(en);
        }
    }
}
```

Using `toString()` and `toObject()` to facilitate deploying models in distributed simulation
Motivational Example: In DeKalb County, Georgia, near Atlanta, a diverging diamond interchange at Interstate 285 and Ashford-Dunwoody Road opened on June 3, 2012.

Figure 1. Photo. First U.S. DCD interchange at I-44 and Route 13 in Springfield, MO.

The DEVSJAVA Lab: carWashSys

Download the DEVSJAVA Lab package from iCollege
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
An carwash example
1. To simplify the problem, let’s first assume the system will reject any incoming cars and trucks if it is 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. Each car is unique – add vehicleEntity
8. Display message – implement the toString() in vehicleEntity
9. Show model state on SimView -- getTooltipText()
10. Add transducer – a transducer is a model that records job arrivals and departures and measures some results of interest (e.g., turnaround time and throughput).
11. Test the model without SimView
A Car Wash Center with schedule

- The car wash center can only work for 100 minutes before it needs to take a 20 minute break. After the break, the center works for another 100 minutes and then takes another 20 minutes break.
- The center will wash a new car (if there is car to be washed) as long as it is not break time. If the break time comes before a car is finished, the break time will be delayed until the car is finished. Note: A delayed break is still 20 minutes.
- During the break, cars can still come in and will be queued.
- This example shows how to keep track of time (e and \(\sigma\)) in DEVS. It will be useful for HW1 and HW2.
Time Tracking in DEVS

- The elapse time $e$ in deltext() is the time elapsed since the last event.
- The variable $sigma$ is the time remaining in the current phase. When $sigma$ expires, the out() and deltint() are invoked.
- Continue($e$) in the deltext() ensures the $sigma$ is correctly updated when the model is interrupted by external events. See the FireOnceNeuron example in the next slides.
- Whenever the model comes to its deltext(), it means the elapse time $e$ has passed.
- Whenever the model comes to its deltint(), it means $sigma$ time has passed (before the model sets a new $sigma$).
- Knowing how $e$ and $sigma$ work allows us to keep track of the time, e.g., keep track of how much time has passed since a specific time point.
This neuron fires exactly once after a fixed delay while ignoring all subsequent inputs.

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

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

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

public message out(){
    //if (phaseIs("fire"))
    return outputNameOnPort("pulse","out");
}
working_active: the schedule is “working” and the car washing center is washing a car.

working_passive: the schedule is “working” but there is no car to wash.

break: the schedule is “break” (and the car washing center is idle).
A Car Wash Center with schedule

• How to model the following situation?
  – **Original**: The center will wash a new car (if there is car to be washed) as long as it is not break time. If the break time comes before a car is finished, the break time will be delayed until the car is finished. Note: A delayed break is still 20 minutes.
  – **Variation #1**: The center will wash a new car (if there is car to be washed) as long as it is not break time. If the break time comes before a car is finished, the break time will be delayed until the car is finished. But the original working-break schedule will not be changed.
  – **Variation #2**: The center will wash a new car (if there is car to be washed) as long as it is not break time. If the break time comes before a car is finished, the center takes the break, and will continue washing the un-finished car after resuming work.
  – **Variation #3**: The center will wash a new car (if there is car to be washed) only when the car can be finished before the break time.
• Look into the DEVSJAVA packages
• The simpleArc Example