



DEVS Training Course

DEVS Programming – Part 1

Prof. Dr. Valdemar Vicente Graciano Neto

From the last class...




- Hello World
- Atomic Models
- Coupled Models
- Message exchange between systems

In this class



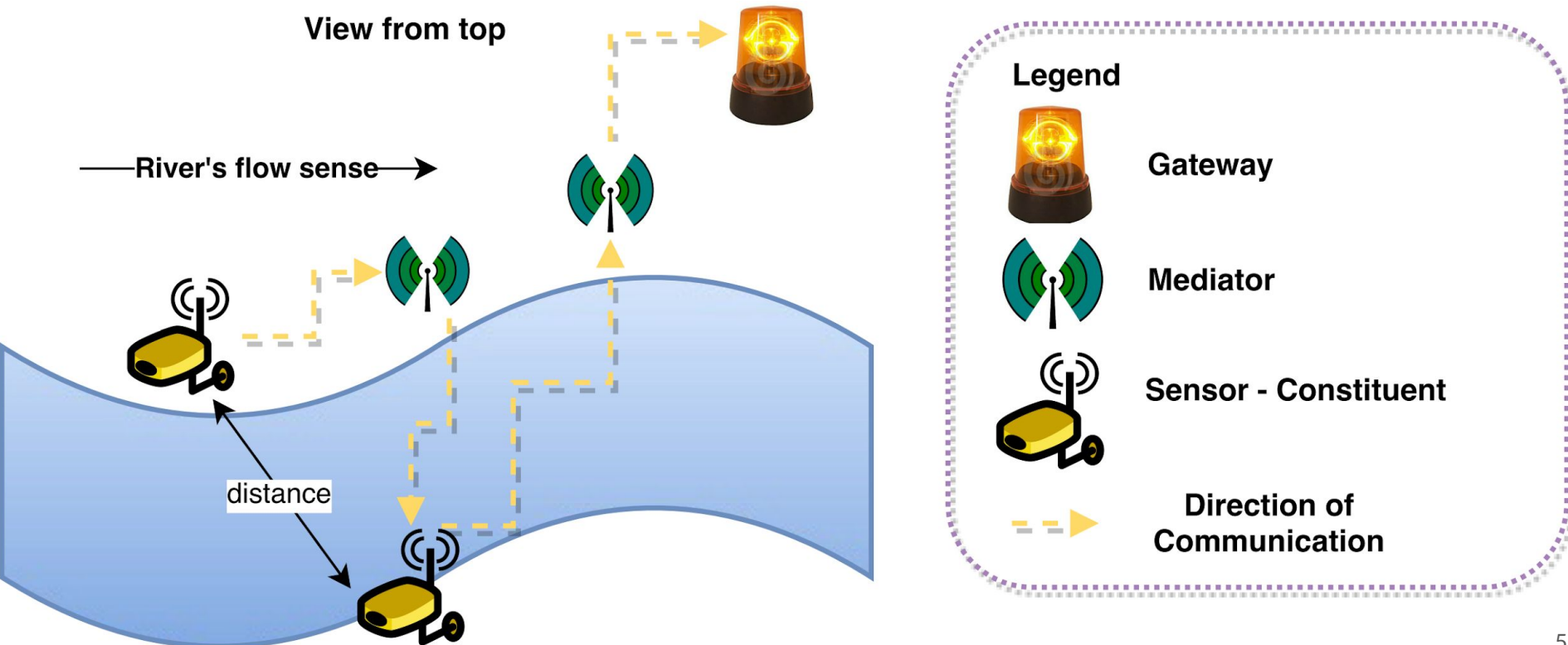
- Data Types
- Specifying ports
- Changing the messages content
- Accessing message content
- Events
- Processing with Atomic Models

DEVS

- 
- Every Atomic Model generates a corresponding Java code;
 - Non-primitive data types (ie, from Java libraries) must be explicitly imported.

Motivational example

- Flood Monitoring System with 2 Sensors and 1 Gateway



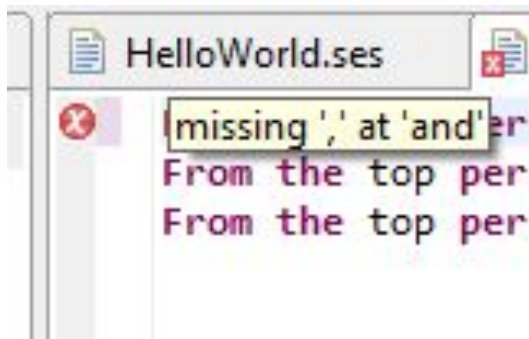
DEVS

- Create a Project named ReducedFM
- SimulationFM.ses

From the top perspective, SimulationFM is made of Sensor1, Sensor2 and Gateway!

From the top perspective, Sensor1 sends Depth to Sensor2!

From the top perspective, Sensor2 sends Depth to Gateway!



It happens if you don't put the comma before the 'and'.

DEVS



- Create a Project named ReducedFM
- SimulationFM.ses

Important
Remark:


From the top perspective, SimulationFM is made of Sensor1, Sensor2 and Gateway!

From the top perspective, Sensor1 sends Depth to Sensor2!

From the top perspective, Sensor2 sends Depth to Gateway!

Good Practice: The name used here (type of data transferred) should be the name of the output port of one and input to the other.

DEVS

- 
- Canonical Code of the Atomic Model
 - Declaration of data types
 - Instance Variables declaration
 - Ports declaration
 - Variables initialization
 - State machine specification
 - Events

DEVS



- Create three atomic models: Sensor1, Sensor2, and Gateway.
- Logic:
 - Generates the data
 - Put the data in the output port
 - Send data

DEVS



- Sensor1.dnl
 - Datatype declaration

A Depth has a value!
the range of Depth's value is Integer!

DEVS



Sensor1.dnl

- Respective Instance Variable

use measureData with type Depth!

DEVS



Sensor1.dnl

Output port specification

generates output on Depth with type Depth!

DEVS

Sensor1.dnl

- Variables initialization

Initialize variables

```
<%
```

```
java.util.Random gerador = new java.util.Random();
```

```
int number1 = gerador.nextInt(10);
```

```
measureData = new Depth(number1);
```

```
%>!
```

- You can also do it in an internal event

//On the transition for s1, perform the assignment!

Internal event for s0

```
<%
```

```
measureData = new Depth(new Integer (1));
```

```
%>!
```

DEVS



Sensor1.dnl

- State Machine - Sensor1
- First transition initializes variables once this was defined to happen via Internal Event. After, it sends the data and waits another second.

to start hold in s0 for time 1!
from s0 go to s1!
hold in s1 for time 1!
after s1 output Depth!
from s1 go to s0!

DEVS

Sensor1.dnl

- Output event - sensor outputs the data!

//On the transition going out from the state s0, perform the actions!

output event for s0

<%

output.add(outDepth, measureData);

%>!

DEVS

Sensor1.dnl code

A Depth has a value!
the range of Depth's value is Integer!

use measureData with type Depth!

generates output on Depth with type Depth!

Initialize variables

```
<%  
java.util.Random gerador = new java.util.Random();  
int number1 = gerador.nextInt(10);  
measureData = new Depth(number1);  
%>!
```

```
to start hold in s0 for time 1!  
from s0 go to s1!  
hold in s1 for time 1!  
after s1 output Depth!  
from s1 go to s0!
```

```
//On the transition for assigned, perform the assignment  
Internal event for s0
```

```
<%  
measureData = new Depth(new Integer (1));  
%>!
```

```
//On the transition going out from the state, perform the actions  
output event for s1
```

```
<%  
output.add(outDepth, measureData);  
%>!
```


DEVS



Sensor2.dnl

- It should be specified to receive the same type of data.

A Depth has a value!
the range of Depth's value is Integer!

use measureData with type Depth!

accepts input on Depth with type Depth! //environment
//sense
generates output on Depth with type Depth!

Initialize variables
<%
measureData = new Depth();
%>!

DEVS



Sensor2.dnl

– State machine

to start passivate in s0!
when in s0 and receive Depth go to s1!
hold in s1 for time 1!
after s1 output Depth!
from s1 go to s2!
passivate in s2!

DEVS



Sensor2.dnl

- Events (Input and Output)
 - It should send the same type of data received

external event for s0 with Depth

<%

measureData = (Depth) messageList.get(0).getData();

%>!

//On the transition going out from the state, perform the
actions

output event for s1

<%

output.add(outDepth, measureData);

%>!

Sensor2.dnl code

DEVS

A Depth has a value!
the range of Depth's value is Integer!

use measureData with type Depth!

accepts input on Depth with type Depth! //environment //sense
generates output on Depth with type Depth!

Initialize variables

```
<%  
measureData = new Depth();  
%>!
```

to start passivate in s0!
when in s0 and receive Depth go to s1!
hold in s1 for time 1!
after s1 output Depth!
from s1 go to s2!
passivate in s2!

external event for s0 with Depth

```
<%  
measureData = (Depth) messageList.get(0).getData();
```

```
%>!
```

//On the transition going out from the state, perform the actions
output event for s1

```
<%  
output.add(outDepth, measureData);  
%>!
```

DEVS

Gateway.dnl

- It receives and verifies the data

A Depth has a value!
the range of Depth's value is Integer!

use measureData with type Depth!

accepts input on Depth with type Depth! //environment //sense

to start passivate in s0!
when in s0 and receive Depth go to s1!
hold in s1 for time 1!
from s1 go to s0!

external event for s0 with Depth

<%

for(int i = 0; i < messageList.size(); i++){

 Depth valueReceived = (Depth)messageList.get(i).getData();
 int value = (int) valueReceived.getValue();
 System.out.println("Value: " + value);

}

%>!

DEVS



Remarks

- There are 3 types of events: internal, external and output;
 - Internal: you can associate with a simple/internal transition (to perform something, with no input nor output);
 - Output: the event you associate with states that output data;
 - External (could be named “Input”): the event you use to receive data;

DEVS



Remarks

- The state machines of an Atomic Model are deterministic.
- It means that:
 - A same passivate state can have multiple external transitions (one for each input) in different ports;
 - A hold state should have only one transition (internal or output)

DEVS - From Class 00

- General Rule for Specification of an Atomic Model State Machine:
 - Communication between two systems is only possible if one system is in the sending state and the other is in the receiving state.
 - “Pattern” for States specification:

DEVS Input:

passivate in << fromState > >!

when in << fromState > > and receive <<dataReceived > > go to << toState > >!

//external transition!

DEVS Output:

hold in << fromState > > for time 1!

after << fromState > > output << dataType > >!

from << fromState > > go to << toState > >! //internal transition!

<<hold or passivate again to>> <<toState>>!

Exercises



- 1) Change the Flood Monitoring specification so that the data sent continuously reaches the Gateway;
- 2) Modify Sensor1 to send random numbers!
- 3) Modify Sensor2 to not only forward data, but also to send sensed data through it.
- 4) Add a new element in the Simulation called StimuliGenerator. It should create random values and send them to sensors. Create one for each sensor.
- 5) Add a clause to the Gateway: if the value is greater than 2 (meters), trigger an alarm.

Next class



- Working with files;
- Composing models and changing perspectives.