

Events in AnyLogic

Nathaniel Osgood

October 29, 2013

Events & Scheduling in AnyLogic

- Reminder: In simulating stock & flow models, time advances in steps
 - Euler integration: Fixed-sized Steps
 - Runga-Kutta: Fixed or variable sized steps
 - For each timestep, we compute the flows & update the stocks
- AnyLogic jumps from “event” to “event”
 - The data structure that keeps track of such events is called the “schedule”
 - The associated process is called the “scheduler”

Implicit Events we've Seen

- Transitions
 - Fixed rate (Poisson arrival)
 - Timeout
 - Condition
 - Message transmission (schedules event for the receiver)
- Starting a model
- Stopping a model
- In this course, we term these *implicit events* because they are not reified as objects in the model
- To handle these events, code is inserted into certain handler areas for each of different sorts of classes

Example: Built-In Events (Agent 1)

The screenshot displays the AnyLogic Advanced interface. On the left is a project tree with a 'Person' agent selected. The main workspace shows a statechart for the 'Person' agent with states 'FreeWandering' and 'GoToWater'. A red text box with a red arrow pointing to the 'On Arrival' event handler in the properties panel contains the following text:

“Handler”: Code is executed when the specified event (e.g., arrival at a destination, message arrival) occurs.

The 'Person - Active Object Class' properties panel is open, showing the following settings:

- General: Space type: Continuous Discrete GIS
- Advanced: Environment defines initial location
- Agent: Initial coordinates: X: [] Y: []
- Parameters: Movement parameters: Velocity: [] Rotation: []
- Description: On Arrival: []
- On Message Received: `infectionStatechart.receiveMessage(msg);`
- On Before Step: []
- On Step: []

Example: Built-In Events (Agent 2)

The screenshot displays the AnyLogic Advanced software interface. The main workspace shows a state transition diagram for a 'Person' agent. The diagram includes a state 'FreeWandering' (represented by a yellow oval) and a state 'GoToWater' (represented by a yellow rectangle). Transitions are labeled 'GotThirsty' and 'NewDir'. The 'FreeWandering' state has a self-loop labeled 'NewDir'. The 'GoToWater' state has a transition labeled 'GotThirsty' leading back to 'FreeWandering'. The diagram is set against a grid background.

The 'Person - Active Object Class' properties window is open, showing the following settings:

- Name: Person
- Ignore:
- Agent: Agent Generic
- Startup Code:
- Destroy Code:

The left sidebar shows the project hierarchy, including 'Main', 'Person', 'Simulation', 'Person', 'Person.java', 'Male.java', 'Male', 'Main', and 'Elephant'. The right sidebar shows the tool palette, including 'Model', 'Action', 'Analysis', 'Presenta...', 'Line', 'Polyline', 'Curve', 'Rectangle', 'Round Rect...', 'Oval', 'Arc', 'Pixel', 'Text', 'Image', 'Group', 'Button', 'Check Box', 'Edit Box', 'Radio Buttons', 'Slider', 'Combo Box', 'List Box', 'File Chooser', 'Progress Bar', 'CAD Drawing', 'GIS Map', 'Connectivity', 'Enterprise Li...', and 'More Libraries...'.

Example: Built-In Events (Main)

The screenshot displays the AnyLogic Advanced software interface. The main workspace shows a statechart for a patient's behavior. The statechart includes a state named "FreeWandering" (represented by a yellow oval) and a state named "GoToWater" (represented by a yellow rectangle). The "FreeWandering" state has a self-loop labeled "NewDir" and a transition labeled "GotThirsty" leading to the "GoToWater" state. The "GoToWater" state has a transition labeled "behavior" leading back to the "FreeWandering" state. Parameters listed include "drinkingPeriod", "thirsty", "headingRandom", and "headingToWater".

The "Main - Active Object Class" properties window is open, showing the "Startup Code" field with the following code:



```
environment.deliverToRandom("Infection");
```

The "Destroy Code" field is empty. The "General" tab is selected, and the "Name" field is set to "Main".

The Schedule

- At a given time, the schedule keeps track of a number of queued events
- Events may get added to the schedule (e.g. when we enter a new state)
- Events get deleted from the schedule
 - When they fire off and are complete
 - When another mutually exclusive event preempts them (e.g. a person dies before they recover from an infection)

Explicit Events

- Explicit Events can also be declared from the palette  Event  Dynamic Event
 - Dynamic events can have multiple instances
 - Each instance can be scheduled at different times
 - The instances disappear after event firing
 - Regular (static) events can be rescheduled, enabled/disabled, but can only have one scheduled firing at a time
- There are some subtleties with explicit events

(Explicit) Event Subtleties

- Be very careful of what you count on for recomputation of rate – may think was recomputed, but hasn't been
- Event rates (and likely event timeout times) are only computed occasionally, not continuously
 - These are computed when
 - Explicitly call event methods
 - `start()`
 - `restart()`
 - `onChange()`
 - » e.g. if wish to update rates associated with transitions, *Main* can periodically call *onChange()* on each agent
 - An event in Main can take care of this task
 - When event fires and requires restarting
 - (For outgoing transitions) when enter a state in a statechart
- Calling “reset” will disable a rate until re-enable (e.g. with call to *restart()*)

Event Times: Common Options for Event Scheduling

- At a specified rate (Poisson arrivals)
 - Interarrival time is exponentially distributed!
 - Mean time between events is reciprocal of rate (i.e. $1/\text{rate}$)
- One-time
 - Can go off at a particular time (specified as a calendar time or as a double-precision value)
- At some initial time and then cyclically beyond with set “timeout” period
 - The timeout period is set according to the time unit
 - This goes off after *exactly* the timeout time
- When boolean condition changes (depends on *onChange* being called)
- Manually (via `restart()`) – see following slides)

Event Subtleties

- Be very careful of what you count on for recomputation of rates – may think was recomputed, but hasn't been
- Event rates (and likely event timeout times) are only computed occasionally, not continuously
 - These are computed when
 - Explicitly call event methods
 - `start()`
 - `restart()`
 - `onChange()`
 - When event fires and requires restarting
 - (For outgoing transitions) when enter a state in a statechart
- Calling “reset” will disable a rate until re-enable (e.g. with call to *restart()*)

Dynamic Events (Closure-Like)

- Like a static event, a dynamic event is associated with an *action* to invoke when it occurs
- A *static* event has a single associated schedule
- Just as a class can be associated with multiple instances, Dynamic events can have *multiple instances*
 - Each instance can be scheduled at different times
 - The schedule for each different instance proceed in parallel
 - The instances disappear after event firing
 - We can think of each dynamic event instance as its own one-time (“one-shot”) event
- Schedule a dynamic event with `create_event(timeout, parameters...)`
 - The event will be “awoken” time *timeout* from now!

Parameterization of Dynamic Events

- With a dynamic event, we create the event during simulation, but at a different time than it occurs
- Frequently the action we want to performed in a dynamic event depends on specific context known at the time that it was created
 - For example, we want to create or delete a particular person, or a person with particular characteristics
- Specification of dynamic events at design time defines custom ‘parameters’ (‘arguments’)
 - Parameters values can be used to communicate context from time of creation of the dynamic event until when it fires
 - Particular values for these parameters are then given at time when dynamic event instance is created

Specifying a Dynamic Event Step 1

The screenshot displays the AnyLogic Advanced software interface. The main canvas shows a grid with a text label "add Dynamic event" and a green oval containing a lightning bolt icon labeled "ScheduleDeerBirth". A blue arrow points from the text "1) Click here," to the "Dynamic Event" icon in the Palette window. Another blue arrow points from the text "2) use mouse to click in Canvas to" to the "ScheduleDeerBirth" icon. A red arrow points from the text "Click on the 'Model' label in the 'Palette' window" to the "Model" label in the Palette window.

1) Click here,
2) use mouse to click in Canvas to

add Dynamic event

ScheduleDeerBirth

Model

Click on the "Model" label in the "Palette" window

General

Name: Show Name Ignore Public Show At Runtime

Description

Name	Type	Default Value

Action:

Specifying a Dynamic Event Step 2

The screenshot displays the AnyLogic Advanced interface. The main workspace shows a grid with several elements: a 'population [...]' icon, an 'environment' icon, and a 'ScheduleDeerBirth' icon. A mouse cursor is positioned over the 'ScheduleDeerBirth' icon, and a line points from it to the 'Properties' window.

The 'Properties' window is open to the 'ScheduleDeerBirth - Dynamic Event' configuration. It includes the following settings:

- General:** Name: ; Show Name; Ignore; Public; Show At Runtime
- Description:** Parameters table:

Name	Type	Default Value
isFemale	boolean	true
genotype	int	0
latitude	double	52.9
longitude	double	106.05

Action:

Attractive Use of Dynamic Events 1

Scheduling Future Birth at time of Conception

- Mating of deer during rut occurs long before births of fawns
- Contacts between deer during rut could be simulated in the model
 - At time of contact, create single dynamic event to schedule associated future birth
 - Could save away information of history relevance e.g.
 - Characteristics of parents
 - Infection status
 - Genotype
 - Stress level
 - Location of where conception occurred

Attractive Use of Dynamic Events 2

Adding in Individuals to Population over a Time Interval

- Dynamic events can be very handy if have a known number of actions that need to take place spread out over some period of time
- Example: Given: Known count of Immigrants with particular characteristics to be added to model population over course of each month
 - Suppose we don't know when these individuals arrive during the month
 - We can simply create the same count of dynamic events, whether each dynamic event takes care of
 - Creating a person with known characteristics
 - Adding that person to the model population

This approach will be discussed in an upcoming guest lecture