BISS 2013: Simulation for Decision Support

Lecture 18

Discrete Event Modelling and Simulation in AnyLogic (Practice)

Peer-Olaf Siebers (Nottingham University)

Stephan Onggo (Lancaster University)

pos@cs.nott.ac.uk

The University of Nottingham
UNITED KINGDOM · CHINA · MALAYSIA

Lancaster University
MANAGEMENT SCHOOL

Motivation

- Learn more about the principles of DEM/DES in AnyLogic
- Gain experience with building a discrete event simulation model

Principle of DEM/DES in AnyLogic

- Most processes we observe consist of continuous changes
- Often we can abstract these processes from their continuous nature and consider only important moments (events)
- An event is an instance of time in which a significant state change occurs

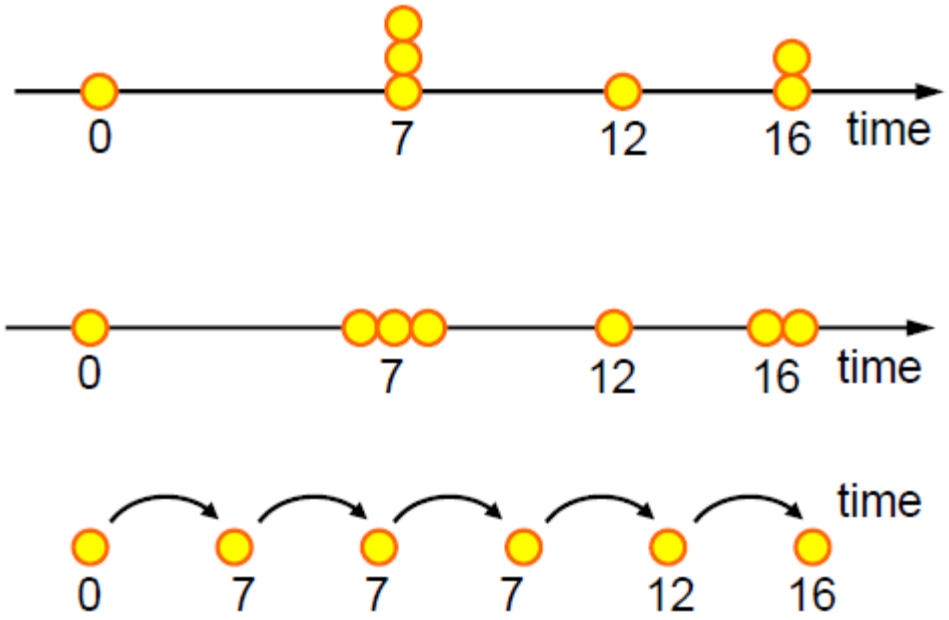
- Events in AnyLogic
  - Take zero time to execute
  - Are atomic (will not interfere with any other event execution)
  - May cause changes in the model
  - May schedule other events in the future

Principle of DEM/DES in AnyLogic

- If AnyLogic engine is executing a purely discrete model, the time is essentially a sequence of events, and the engine just jumps from one event to another.
- If several events are scheduled to occur at the same time (are simultaneous), they are serialized, i.e. executed one after another in some internal (not guaranteed) order.
- If the order is important for you, you should take care of it when developing the model so that the simulation results do not depend on the engine implementation.

Principle of DEM/DES in AnyLogic

- Serialisation

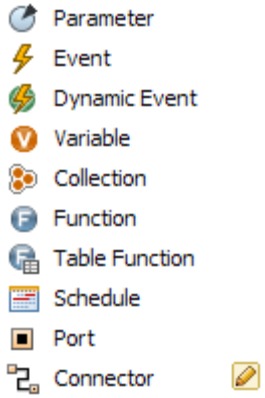


Two Approaches to DES Model Building

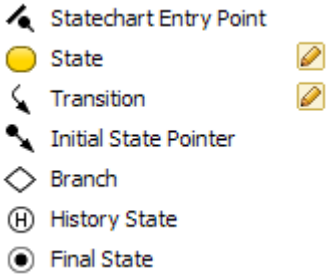
- Two approaches to building DES models in AnyLogic
  - Using low-level primitives
    - At the low level events may be scheduled by two types of objects: Events and state charts.
  - Enterprise Library
    - There is a library of higher-level objects that help you to create discrete event patterns frequently used in process-centric modelling (e.g. queuing, resource usage, entity generation)
    - High-level objects:
      - Building blocks for constructing flow charts
      - Active objects that consist of a collection of low-level primitives and Java code
      - Can interoperate with low-level primitives

Using Low-level Primitives

- General elements
  - Parameter + plain variable + collection
  - Event + dynamic event
  - Function + table function + schedule
  - Port + connectors

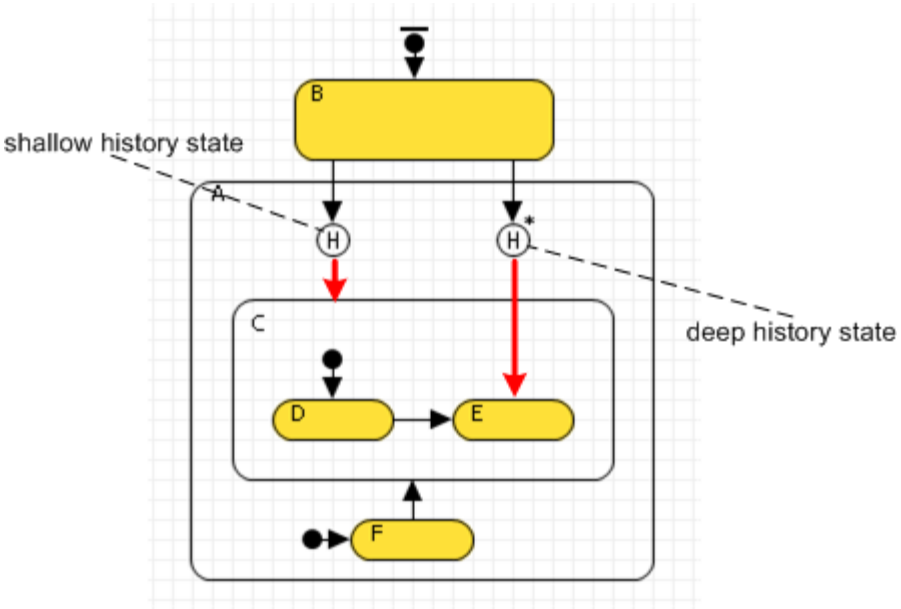


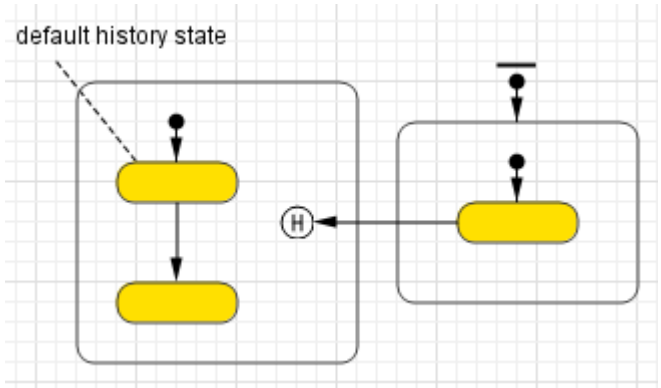
- State chart elements
  - Pointers
  - States
  - Transition
  - Branch



Using Low-level Primitives

- History states
  - Shallow history state is a reference to the most recently visited state on the same hierarchy level within the composite state.
  - Deep history state is a reference to the most recently visited simple state within the composite state.



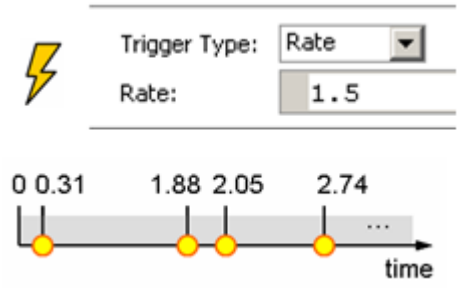


Using Low-level Primitives

- Event triggers
  - Condition Triggered Event
    - Use this if you want to monitor a certain condition and execute an action when this condition becomes true
    - Use event.restart() at the end of your Action list if you want your event to continue monitoring the condition after the event has occurred once
  - Timeout triggered event
    - Use this when you need to perform an action at some particular moment of time; such an event occurs exactly in Timeout time after it is started
    - Different working modes
      - Scheduling periodic events
      - Scheduling some action at the very beginning of simulation
      - Scheduling some action at particular moment from the present situation

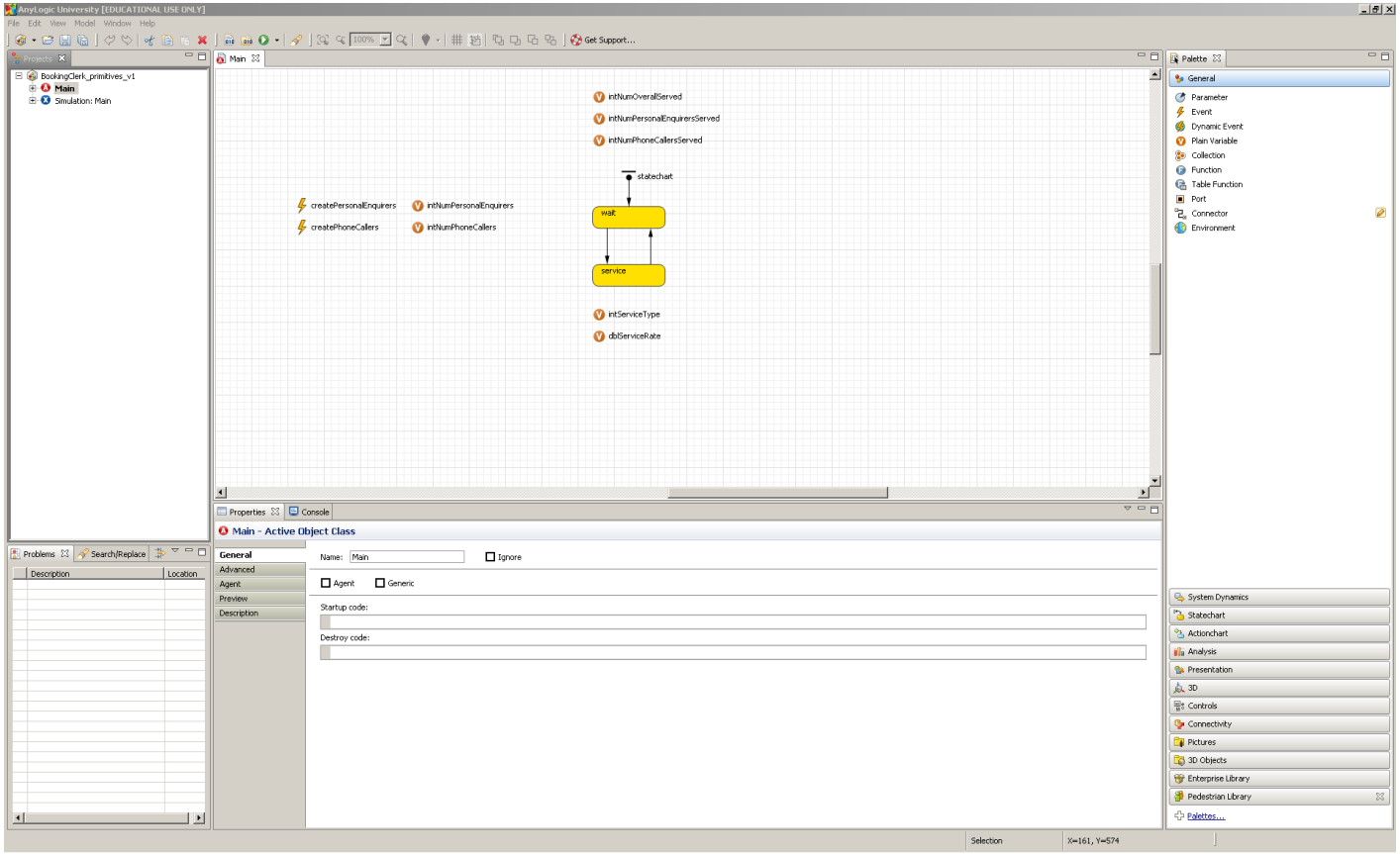
Using Low-level Primitives

- Event triggers (cont.)
  - Rate triggered event
    - Use it to model a stream of independent events (Poisson stream)
    - Event is executed periodically with time intervals distributed exponentially with the parameter rate (e.g. if rate is 5, the event will occur on average 5 times per time unit).
    - Frequently used to model independent arrivals (e.g. customer arrivals in queuing systems)
    - Example:
      - Arrival rate = 1.5
      - Inter arrival time = 1/1.5 = 0.67



Using Low-level Primitives

- Example: Booking Clerk (using primitives)



Using the Enterprise Library

- Enterprise Library has two base classes
  - Entity class (for representing anything that is an object in the process)
    - The base class for all entities defined within Enterprise Library
    - Regular Java class
    - Objects can contain other entities
    - Objects can possess resource units
    - Objects can have user-defined data fields and methods (class extension)
    - Objects have default and can have user-defined animation
  - Resource Unit class
    - The base class for all resource units defined within Enterprise Library
    - Regular Java class
    - Objects can have user-defined data fields and methods (class extension)
    - Objects have default and can have user-defined animation

Entity flow

 **Source** Generates entities.

 **Sink** Disposes incoming entities.

 **Enter** Inserts entities created elsewhere into the flowchart.

 **Exit** Accepts incoming entities.

 **Hold** Blocks/unblocks the entity flow.

 **Split** Creates a new entity (copy) of the incoming entity.

 **Combine** Waits for two entities, then produces a new entity from them.

 **SelectOutput** Forwards the entity to one of the output ports depending on the condition.

 **SelectOutput5** Routes the incoming entities to one of the five output ports depending on (probabilistic or deterministic) conditions.

 **Queue** Stores entities in the specified order.

 **Match** Finds a match between two entities from different inputs, then outputs them.

 **RestrictedAreaStart** Limits number of entities in a part of flowchart between corresponding area start and area end blocks.

 **RestrictedAreaEnd** Ends an area started with RestrictedAreaStart block.

Working with entity contents

 **Batch** Accumulates entities, then outputs them contained in a new entity.

 **Unbatch** Extracts all entities contained in the incoming entity and outputs them.

 **Pickup** Adds the selected entities to the contents of the incoming entity.

 **Dropoff** Extracts the selected entities from the contents of the incoming entity.

 **Assembler** Assembles a certain number of entities from several sources (5 or less) into a single entity.

Processing

 **Delay** Delays entities by the specified delay time.

Working with resources

 **ResourcePool** Provides resource units that are seized and released by entities.

 **Seize** Seizes the number of units of the specified resource required by the entity.

 **Release** Releases resource units previously seized by the entity.

 **Service** Seizes resource units for the entity, delays it, and releases the seized units.

Transportation

 **Conveyor** Moves entities at a certain speed, preserving order and space between them.

Time measurement

TimeMeasureStart

TimeMeasureStart as well as **TimeMeasureEnd** compose a pair of objects measuring the time the entities spend between them, such as "time in system", "length of stay", etc.

This object remembers the time when an entity goes through.

TimeMeasureEnd

TimeMeasureEnd as well as **TimeMeasureStart** compose a pair of objects measuring the time the entities spend between them.

For each incoming entity this object measures the time it spent since it has been through one of the corresponding **TimeMeasureStart** objects.

Clock

This object displays the current time and date during simulation as a clock.

Network based modeling

Network

Object that defines network.

NetworkEnter

Adds an entity to the specified location in the network.

NetworkExit

Removes an entity from the network.

NetworkMoveTo

Moves an entity from its current location to new location.

NetworkResourcePool

Describes resources located in the network.

NetworkRelease

Releases previously seized resources.

NetworkSeize

Seizes a set of resources.

NetworkSendTo

Sends a set of portable and/or moving resources resources to specified location.

NetworkAttach

Attachs a set of portable and/or moving resources to the entity.

NetworkDetach

Detachs previously attached resources from the entity.

Modeling storages in network

NetworkStorage

Models two opposite racks and an aisle between them.

NetworkStorageZone

Models a storage zone containing a set of racks (defined by **NetworkStorage** objects), providing centralized access and managing of racks.

NetworkStoragePick

Picks an entity from a cell of a rack (**NetworkStorage**) or a storage zone (**NetworkStorageZone**) and moves it into the specified network node.

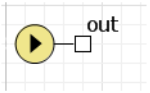
NetworkStoragePut

Places an entity into a cell of the specified rack (**NetworkStorage**) or storage zone (**NetworkStorageZone**).

Using the Enterprise Library

- Enterprise library reference guide

Source



Generates entities. Is usually a starting point of a process model.

The entities may be of generic class **Entity** or of any user-defined subclass. You can customize the generated entities by specifying which constructor should be called, what action should be performed before the entity exits the Source object, and what should be the animation shape associated with the entity.

There are a number of ways to define when and how many entities should be generated.

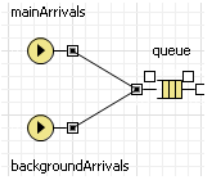
The following demo model illustrates entity arrival modes:

Demonstration model: Source Arrival Modes

You can use arrival rate (and change it dynamically by calling `set_rate()`), interarrival time, rate defined by a table function of time, table of exact arrival times and quantities, and you also can programmatically call the `inject()` method of this object. For example, a Poisson steam of arrivals can be implemented by choosing arrivals with a certain rate, or by specifying the exponentially distributed interarrival time. You can also set the number of entities in each arrival and limit the total number of arrivals.

If interarrival time is used and it occasionally evaluates to `Double.POSITIVE_INFINITY`, the **Source** stops generating entities and will never resume. If rate is used and it becomes 0, no next arrival will be scheduled until the rate changes to a positive value.

In some cases it makes sense to use two or more **Source** obejcts working in parallel to implement complex arrival patterns, see the picture below.



Please note that **Source** would not allow the entities to wait at its output until they can be consumed by the next object. Therefore you may need a buffering object, say, a **Queue** between the **Source** and, e.g. a **Conveyor**, which is not always able to accept an incoming entity.

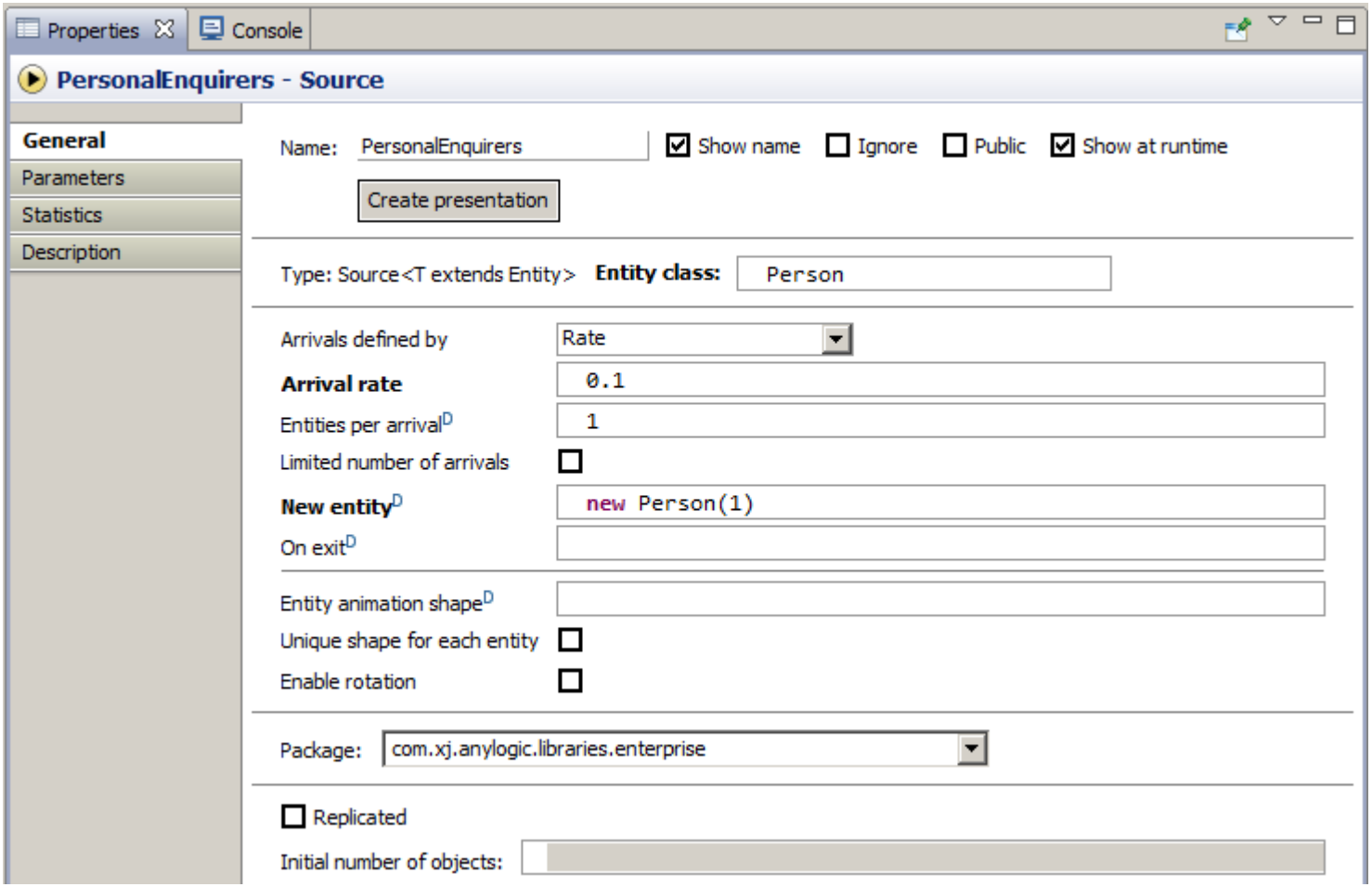
Apart from **Source**, there are other ways to create entities in the Enterprise Library models. For example, you can use **Enter** object as a flow starting point and call its `take()` method to inject entities. The latter method makes sense when the entities are generated elsewhere (e.g by a statechart or an event) and just need to be injected into the process.

Generic parameters

Using the Enterprise Library

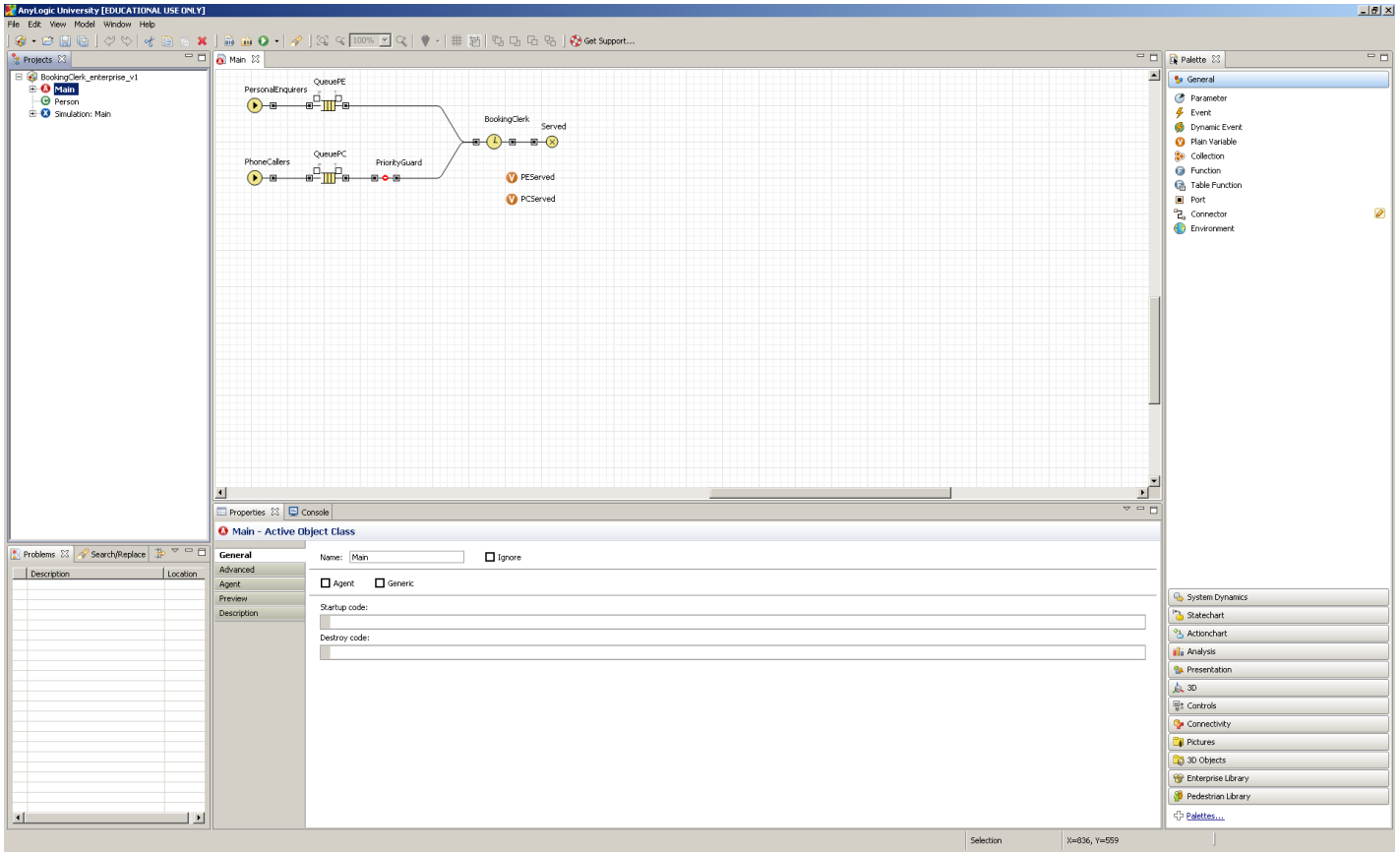
- Parameters of Enterprise Library Objects
  - Simple static parameters
    - Evaluated once, but may be changed during the model execution
    - Example: Capacity: 100
  - Dynamically evaluated expressions
    - Evaluated each time they are needed, e.g. each time the delay time, the speed or other property of an entity needs to be obtained
    - The corresponding entity is accessible as "entity. ..."
    - Example: DelayTime: exponential(1); SelectCondition: entity.type==VIP
  - Dynamically executed code pieces
    - Evaluated each time a certain event occurs at the object: the entity enters/exits it, conveyor stops, etc.
    - Example: OnExit: entity.setColor(Color.red);

Using the Enterprise Library



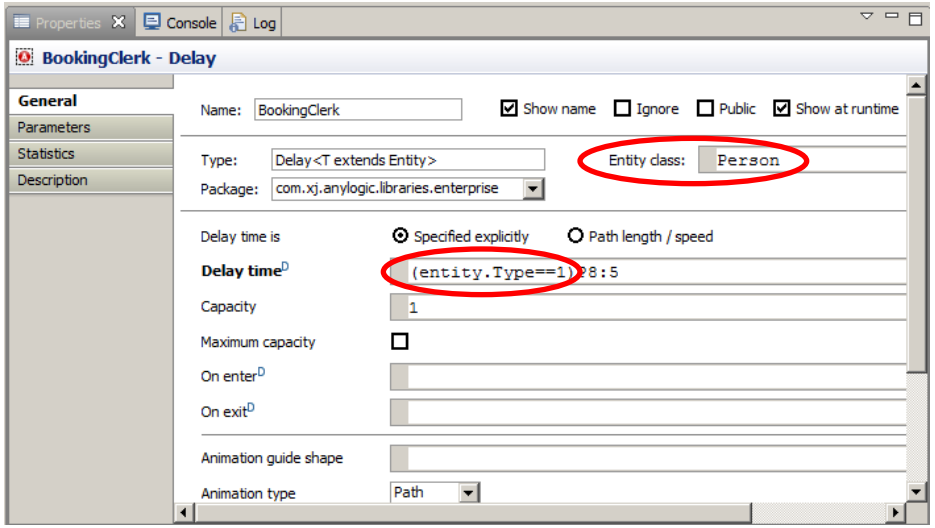
Using the Enterprise Library

- Example: Booking clerk (using Enterprise Library)



Some Tips

- Remember that all names have to be Java compliant; they are case sensitive and no spaces are allowed
- Remember to use the correct Entity class when you want to access class specific fields or methods
- Use code completion (CTRL+SPACE) to avoid errors
- Beware: Entity vs. entity
  - Entity = base class
  - entity = standard name for enterprise library entities
- Read AnyLogic Help FAQ section



Exercise

- Supermarket
  - Develop a simple UML diagram to capture the operations in the till area of a supermarket (without looking at the tutorial sheet)
  - Implement the UML diagram provided in the tutorial

Questions / Comments