jESlet, java Enterprise Simulation light experimental Tool: an Introduction to a Simplified Version of the Model

Pietro Terna

(August 2003)

Dipartimento di Scienze economiche e finanziarie G.Prato, Università di Torino, Italia
pietro.terna@unito.it

INTRODUCTION

jES (the Java Enterprise Simulator) is a frame useful to develop enterprise simulation models based on the Java version of Swarm both (i) to simulate the actions - with the consistent emerging results - of an actual enterprise and (ii) to build virtual or hypothetical enterprises.

In the first case we can use the simulator to test the behavior of an emulated enterprise, as is or modified (also simulating never seen situations), with highly practical goals. In the second case, we are interested in theoretical analysis of enterprise creation, behavior and network interaction.

With *jES* we introduce the existence of two independent sides in our world description and representation and, in a consistent way, in our program (i.e., in our model).

Our simulated enterprise has both orders to accomplish – each described by a "recipe" that contains the WD (What to Do) side of the world - and production units that perform the different steps of the production process; the production units represent the DW (which is Doing What) side of the same world.

A third formalism - the WDW formalism[1]: When Doing What - is related to the time sequence of the events (the orders to be executed) that occur in the world that we are reproducing.

jES has a simplified version named *jESlet*[2] (*jES* light experimental tool), developed using JavaSwarm, JAS, Ascape, RePast and, with some differences, StarLogo) both for comparative reasons and to help scholars of agent based simulation techniques to introduce themselves to these different instruments, mainly in the social science perspective.

---

[1] WDW is not used in the simplified version of the simulator.
[2] jESlet has been created downsizing jES (jesframe-0.9.7.60) from 36 classes with globally 10.612 lines of code to 11 classes with globally 1.670 lines of code.
To follow this explanation in a useful way you need to have the code in your hands; you can download jeslet-0.1.tar.gz (or successive versions) from web.econ.unito.it/terna/jes.

The key feature of *jES* - as identified with the *[KF]* in the paragraph *How the model works* within the *How to use the jES program* - are reproduced in *jESlet* and reported here.

HOW THE LIGHT MODEL WORKS

From a technical point of view it is important to note that almost all the intelligence of our simulation process is placed into the order (WD) side. We can describe the behavior of the code in the following way (suppose that we are not at the beginning of the simulation, so the process is already running to elaborate orders):

1. production units[3] act operating on the orders existing in their waiting lists, if any, one order per each tick of the simulation clock;

    o   the orders whose step is done are placed in a "made production" list, to be successively diffused to other production units;

2. new orders[4] (each containing its recipe) are launched in production:

    o   each order contains a recipe built as sequence of steps to be done,

    o   new orders enter into the simulation:

        ▪   while testing the program or for a theoretical use, randomly generating the orders, via the **orderGenerator** object; this is also the way used in the light version of the program (jESlet);

    o   the new orders are assigned to the production units in the same way described at point 3;

3. each order[5] kept into the made production lists (point 1 above) of the production units makes a search - via the unit code using the assigning tool code - into the world to discover if one or more production units can perform its following (undone) step;

    o   assignments:

        ▪   if (only) one production unit makes a positive reply the order is assigned to the waiting list of that production unit;

    o   orders stay, in a FIFO (First In First Out) way, in the waiting list of the chosen production unit up to their specific step is done;

    o   an order is dropped when a step of its recipe is done and its recipe does not contain other undone steps;

---

[3] ModelActrions2 in ESFrameModelSwarm.java and unitStep1 in Unit.java.
[4] modelActrions2generator in ESFrameModelSwarm.java and createRandomOrderWithNSteps in OrderGenerator.
[5] modelActrions2b in ESFrameModelSwarm.java and unitStep2 in Unit.java.

4. the sequence continuously goes back to the phase described at point 1 for the next click of the clock. (Other steps are devoted to initializing and to accounting operation, but are not reported here, to simplify this presentation).

Time synchronization and parallelism are obtained via a usual trick in simulation: at each tick of the simulation clock, all the production units make the actions described at point 1 above, independently one to each other (the actual time sequence does not matter); then, always in the same clock tick, the program executes point 2; when all these actions are concluded, orders make the operations described at point 3, newly independently and always in the same tick.

THE PROBE PARAMETERS AND THE RECIPE STRUCTURE

When *jESlet* starts we can see its two probe windows (Figure 1). In the first pane we have the ESFrameObserverSwarm parameters[6]:

- displayFrequency, which states the frequency of the display updating while the simulation is running: 1 for updating the display in each simulation clock tick; 2 for updating it every two ticks etc.

- verboseChoice, which states the production, if *true*, of a lot on printed lines related to the internal activities of the program;

- timeToFinish, if not equal zero, is the time, expressed in number of ticks, at which the simulation is stopped.

Figure 1. The parameters of the simulation.

The second pane of Figure 1 reports the ESFrameModelSwarm parameters:

- totalUnitNumber is the number of production units populating our simulation; in the example of Figure 1 we have three units; automatically the units will receive the identifying number 1, 2 and 3, following the file unitData/unitBasicData.txt of the distribution jeslet-0.1.tar.gz. The file contains in each line the identifying number of a unit and the code describing its production capability. In our case the numbers of the unit and those of the related activities are the same in each case, but this is not

---

[6] When changing parameters remember to press Enter key to effectively modify the value into the system. Logic values are *true* and *false*; the shortened forms *f* and *t* do not work here.

a mandatory condition; also the sequence of the number has not to be ordered nor continuous;

- **maxStepNumber** is the maximum number of steps contained in a recipe describing an order;

- **maxStepLength** is the maximum number of units of time (e.g. seconds) attribute to the execution of a step.

In our case, valid cases of recipe structures are:

1 s 1   3 s 1   2 s 1 (the first step requires the execution of the activity 1, in our case made by unit 1, and lasts 1 second; the second step requires the execution of the activity 3, in our case made by unit 3, and lasts 1 second; the third step requires the execution of the activity 2, in our case made by unit 2, and lasts 1 second;

3 s 1   2 s 1 (the first step requires the execution of the activity 3, in our case made by unit 3, and lasts 1 second; the second step requires the execution of the activity 2, in our case made by unit 2, and lasts 1 second.

With **maxStepLength** = 4, a valid case would be:

3 s 4   2 s 2 (the first step requires the execution of the activity 3, in our case made by unit 3, and lasts 4 seconds; the second step requires the execution of the activity 2, in our case made by unit 2, and lasts 2 seconds.

Recipes in orders, in jESlet, are randomly generated following this kind of rules and a dictionary build using the information contained into the units.

EXPLAINING THE CLASSES AND THE MODEL SCHEDULE

The overview, of jESlet classes and of their methods, is made following a UML[7] diagram automatically (Figure 3) generated employing the program[8] Poseidon CE (community edition), version 1.6.1.

In Figure 3 we have the simplified presentation of the classes of jESlet. The starting point is **StartESFrame** containing the **main** method and creating an instance of **ESFrameObserverSwarm**. The creation is followed by the execution of the methods **buildObjects**, **buildActions** and **activateIn**[9] of the new instance.

---

[7] Within the OMG web site (Object Management Group, with the goal of "setting vendor-neutral software standards), at www.omg.org/gettingstarted/what_is_uml.htm, we read that the Unified Modeling Language "helps you specify, visualize, and document models of software systems, including their structure and design."

[8] Poseidon for UML (www.gentleware.com) is directly based on ArgoUML, which is an open source project (www.argouml.org). The Community Edition of Poseidon for UML is the base version and it is offered for free.

[9] **buildObjects**; into the Observer, is responsible of the creation both of the objects used to monitor the model outcomes and of the model itself; the same method, into the Model, has the task of generating all the objects used to run the simulation. **buildActions**, into the Observer creates the events to be executed with the simulation clock to supervise the Model; the same method, into the Model, creates the events to

Following the Swarm protocol, a container class, the Observer, is used to create both the model and the tools necessary to observe its outcomes. Exactly as StartESFrame creates ESFrameObserverSwarm, this one creates an instance of ESFrameModelSwarm, running its methods buildObjects, buildActions and activateIn.

Executing the buildObjects method, ESFrameModelSwarm creates an instance of OrderGenerator and totalUnitNumber instances of Unit. It also creates an instance of AssigningTool and an instance of UnitParameter. UnitParameter is used only in the initial phase of the generation of the instances of Unit, to deal – in the full version of jES – with the problem of complex production units, able to perform more than one activity.

The instances of Order are generated by OrderGenerator while the simulation is running.

SwarmUtils, MyReader and MyExit are static classes used to perform technical tasks.

```
/* producing */
modelActions2.createActionForEach$message(unitList,
             SwarmUtils.getSelector("Unit","unitStep1"));
/* a new order; this step is placed here, after the production step,
 * to align the diffusion of the order forms (orders under execution
 * - next step - or new ones - this step -) */
modelActions2generator.createActionTo$message(orderGenerator,
                       SwarmUtils.getSelector(orderGenerator,
                            "createRandomOrderWithNSteps"));
modelActions2b.createActionForEach$message(unitList,
                 SwarmUtils.getSelector("Unit","unitStep2"));
// Then we create a schedule that executes the
// modelActions.
modelSchedule = new ScheduleImpl (getZone (), 1);
modelSchedule.at$createAction (0, modelActions2);
modelSchedule.at$createAction (0, modelActions2generator);
modelSchedule.at$createAction (0, modelActions2b);
```

Figure 2. The modelActions and the schedule in ESFrameModelSwarm.

When the user presses the *Next* button or *Start* one in the Swarm control pane, the simulated time makes a step or starts running. The sequence of events of the paragraph "How the light model works" is so executed. Now we look contemporary to the points of that description and to the schedule[10] contained in ESFrameModeSwarm, That schedule is reported here in Figure 2. In Figure 2 we have also the use of selectors: for readers unfamiliar with JavaSwarm, a selector is a structure useful to transfer a method name via a list of parameters.

---

be executed with the simulation clock to activate in the due moment the various simulation steps. activateIn is a mandatory technical Swarm task both of the Observer and of the Model.

[10] The numbers used to identify the modelActions items are the same of the full version, where a ModelActions1 exists.

a) The point 1 is related to the execution of ModelActions2 in ESFrameModelSwarm schedule, which activates, at the beginning of each tick of the simulated clock, the unitStep1 method in all the instances of Unit. The current step of the recipe of the first order in the waiting list of each Unit is executed.

b) The point 2 is related to the execution of ModelActions2generator in ESFrameModelSwarm, which activates, as the second event of each tick of the simulated clock, the createRandomOrderWithNSteps method in OrderGenerator. OrderGenerator, via the assign method of AssigningTool, sends the new generated order (one per tick) the instance of Unit able to perform its first step.

c) The point 3 is related to the execution of ModelActions2b in ESFrameModelSwarm, which activates, as the third event of each tick of the simulated clock, unitStep2 method in all the instances of Unit. UnitStep2, via the assign method of AssigningTool, sends the Order instances contained in the made production list of each instance of Unit to the instance of Unit able to perform their next step. Order instances without successive steps to be execute ad dropped out the simulation.

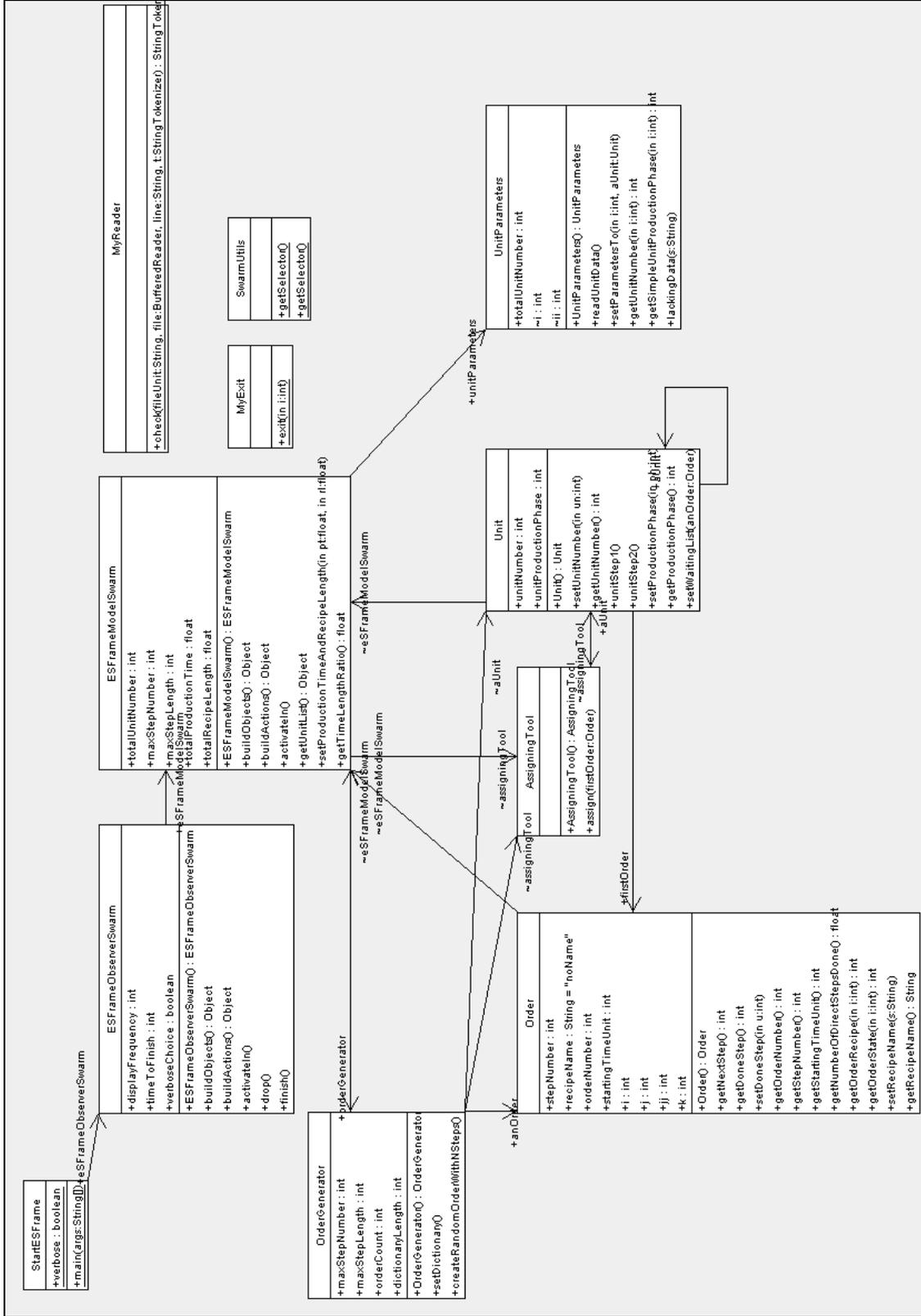With the simulation time running, the schedule of ESFrameModelSwarm continuously repeats the *a*, *b* and *c* tasks.

Figure 3. An UML view of jESlet