Supply Chain Demo 1

Updated: 2021-12-05

This demo is based on the model provided by Abhisekh, from Sean Luke's team at GMU.

For the Rutgers model, see pharma2.html.

The code in this project represesnts a simple supply chain simulation implemented in Java using the MASON API and the MASON DES library

Prepare

You need a Java compile, which can be obtained from Oracle or OpenJDK. For convenience, I also use a Git client and Apache Ant (a `make`-like tool for Java).

- Check out the source code from GitHub. Let's say you've put it to ~/mason/work on your machine. We'll call this location your *work directory*.
- Create also `~/mason/lib`, and put into it the MASON jar file, `mason.20.jar`, which you should download from the MASON site above.
- Obtain the DES librsry source code from its own GitHub repo (see the link above), compile it and package it into `des.jar`, and place the JAR file into the same `~/mason/lib` directory. Sample commands for doing this (assuming you place the DES library source code under `~/mason/git`):

```
#!/bin/csh

set m=~/mason

cd $m
mkdir git
cd git
git init
git remote add origin https://github.com/eclab/mason.git
git pull origin distributed-3.0

cd $m/git/code/contrib/des
javac -d $m/out -cp $m/lib/mason.20.jar sim/des/*.java
# javadoc  -d $m/javadoc -cp $m/lib/mason.20.jar:$m/out -sourcepath . sim.des
cd $m/out
jar cf des.jar sim/des/*.class
cp des.jar ../lib
```

Compile

I compile with Apache Ant:

```
cd ~/mason/work
ant
```

Apache Ant is the tool that takes its commands from build.xml. If you don't want to use Ant, you can just compile the Java code from `src` using the `javac` command, and making use of the JAR files in the ../lib directory.

Run

You can run the simulation with a command like this:

```
./run.sh -until 300 > tmp.log
```

The `-until 300` is a standard MASON option, which will cause the application to run for 300 units of simulation time. Without this option, the application will likely never stop.

Command-line options

There are a few additional command-line options you can use with `run.sh`:

- `-verbose` --- causes various elements of the supply chain model to print various progress messages
- `-config config/pharma.csv` -- specifies the name of the configuration file (see below)

You can also use any of the options that all MASON applications take, such as the already-mentioned `-until`. See the MASON manual (found at the same web site) for details.

Configuration file

The simulation tool takes the parameters of each element of the model from the configuration file, which, by default, is `conf/pharma.csv`

The configuration file is in CSV format. All lines beginning with a hash mark (#) are ignored, as well as all empty lines.

The data lines have the following format:

```
elementName,propertyName,val1[,val2,...]
```

The first column contains the name of the element of the network to which the property applies. The second column, the name of the specific property. The third column (and any subsequent) columns contain the value of the property.

Normally, there is just one value column in each row. However, some types of properties need several columns to be described. For example, the following line

```
Production,faulty,Uniform,0.1,0.2
```

means the that the parameter `faulty` for the element called `Production` describes a uniform distribution with the range [0.1,0.2]. (This is the distribution from which the model of the post-production QA step draws the number that determines the portion of faulty items in each produced lot. This particular distribution means that 10% to 20% of items in each lot are found faulty.)

Sample output

The output file, for a run without the `-verbose` option, may look like this:

```
MASON Version 20.  For further options, try adding ' -help' at end.
Job: 0 Seed: -1860969305
===== Start ===================================
[IngStor.IngStore0(cap=1000.0), has 0.0 units of Ing0. EverReceived=0.0]
[IngStor.IngStore1(cap=1000.0), has 0.0 units of Ing1. EverReceived=0.0]
[IngStor.PackMatStore(cap=1000.0), has 0.0 units of Pack.Mat.. EverReceived=0.0]
[PrePStor.PreprocStore0(cap=1000.0), has 0.0 units of Ing0, plus 0.0+0.0 still in QA; discarded 0.0 faulty units, accepted 0.0 good ones]
[PrePStor.PreprocStore1(cap=1000.0), has 0.0 units of Ing1, plus 0.0+0.0 still in QA; discarded 0.0 faulty units, accepted 0.0 good ones]
[PrePStor.TestedPackMatStore(cap=1000.0), has 0.0 units of Pack.Mat., plus 0.0+0.0 still in QA; discarded 0.0 faulty units, accepted 0.0 good
[Production.Production, has 0.0+0.0 in the work, 0.0+0.0 in QA, and 0.0 in ready storage. Ever started: 0.0. Produced: faulty=0.0, good=0.0]
[PostProcStore has 0.0 units]
[Packaging.Packaging, has 0.0 in the work, 0.0 in QA, 0.0 in ready storage. Packaged: faulty=0.0, good=0.0]
[DispatchStorage.DispatchStore, has 0.0 units. Ever shipped: 0.0]
================================================
===== Finish ===================================
[IngStor.IngStore0(cap=1000.0), has 0.0 units of Ing0. EverReceived=1300.0]
[IngStor.IngStore1(cap=1000.0), has 0.0 units of Ing1. EverReceived=1300.0]
[IngStor.PackMatStore(cap=1000.0), has 0.0 units of Pack.Mat.. EverReceived=1300.0]
[PrePStor.PreprocStore0(cap=1000.0), has 100.0 units of Ing0, plus 0.0+0.0 still in QA; discarded 130.0 faulty units, accepted 1170.0 good one
[PrePStor.PreprocStore1(cap=1000.0), has 7.0 units of Ing1, plus 0.0+0.0 still in QA; discarded 223.0 faulty units, accepted 1077.0 good ones]
[PrePStor.TestedPackMatStore(cap=1000.0), has 251.0 units of Pack.Mat., plus 0.0+0.0 still in QA; discarded 139.0 faulty units, accepted 1161.
[Production.Production, has 0.0+0.0 in the work, 0.0+0.0 in QA, and 0.0 in ready storage. Ever started: 1070.0. Produced: faulty=160.0, good=9
[PostProcStore has 0.0 units]
[Packaging.Packaging, has 0.0 in the work, 0.0 in QA, 0.0 in ready storage. Packaged: faulty=135.0, good=775.0]
[DispatchStorage.DispatchStore, has 9.0 units. Ever shipped: 766.0]
================================================
Job 0: Quit 0
```