A Process-oriented Approach to ARGESIM Benchmark C2 'Flexible Assembly System' using AnyLogic with Java

Štefan Emrich[1, *], B. Malinowsky[2], Oliver Höftberger[2]

[1] Dept. Real Estate Development, Vienna University of Technology, Gusshausstrasse 28-30, 1040 Vienna, Austria
[2] Inst. f. Analysis and Scientific Computing, Vienna University of Technology; **stefan.emrich@tuwien.ac.at*

Simulator. AnyLogic Advanced (educational version 6.2.0) is used for this comparison. The software allows simulation of discrete, continuous and hybrid systems. From version 6 on, the Integrated Development Environment (IDE) is built upon the Eclipse framework and uses a subset of design- and development features from the standard Eclipse Java IDE. Models are built using a graphical editor and can be accessed by using the Java programming language. Interaction is done through basic action-event notifications of library objects. Modification using Java data types is supported as well. AnyLogic is an object oriented simulator which generates Java code generated.

Modeling. Major parts of the model are built using the Enterprise Library of AnyLogic, its primary tool for discrete event modeling. The model consists of 8 almost identical subsystems which only differ in their respective setup- and timing-parameters. Thus in a first step an AnyLogic *active object class* **AssemblyStation** is created, which can be regarded as a sub model. This object is displayed in Figure 1. The subsystem contains external ports over which pallets are exchanged. In this was these ports are used to interconnect the subsequent assembly stations via conveyor belts (of 0.4 meter length). The assembler within **AssemblyStation** is modeled by an instance of **Delay** from the *Enterprise library*, with parameters for statistical data-collection.

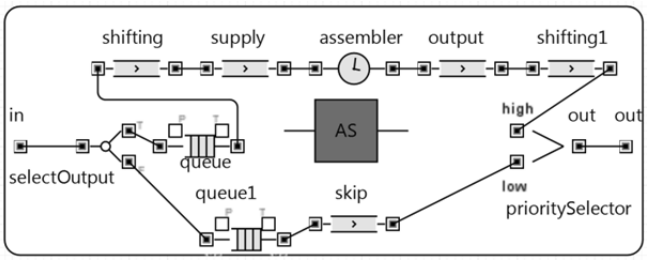

Figure 1. Detailed view of single assembly station, containing belts, shifting, buffers and assembler.

The graphical editor was used to choose, place, arrange and connect the required components from the AnyLogic palette view onto the drawing surface. The object logic is programmed manually in the customization view of the corresponding component using Java language. This logic is stored together with the rest of the model in a XML project file. All of the eight subsystems are linked either directly (A2 and A3 as well as A6 and A1) or by conveyor belts to form the assembly line.

A-Task: Control Strategy - Statistical Evaluation. A local control strategy is used within the components, e.g. for decision whether a pallet is to be shifted and processed. Each pallet object is represented by an instance of the user defined Java class **Pallet**, which extends the predefined class **Entity**. Pallet contains flags representing its processing state and time information, is used for calculations regarding the throughput time.

Collection and analysis of statistical data are dealt with directly in the model. This is supported by the native AnyLogic objects **Dataset** (for internal data storage) and **Chart** (for graphical output) which are instanced from the *Analysis* palette of components.

For the export of the graphical information, i.e. chart presentations the diagrams have been captured from the on-screen simulator window. On the other hand the textual data stored within the **Dataset** objects can be accessed and exported by using a copy-command.

B-Task: Simulation-Throughput. The plot in Figure 2 shows an overview of the simulation results for simulation runs with a varying number of pallets in the system. The red markers indicate the total throughput of pallets processed during the whole simulation period (blue markers: average throughput time of a pallet in sec.).

A total throughput of 1440 pallets is achieved when there are 16 pallets in the system. This result remains constant for any run with a higher number of pallets. Parallel to this, the average throughput time increases steadily, starting from 20 to 60 pallets (Table 1).

SNE Simulation Notes Europe – Print ISSN 2305-9974 | Online ISSN 2306-0271
SNE 21(3-4), 2011, 191-192 | doi: 10.11128/sne.21.bn02.10099

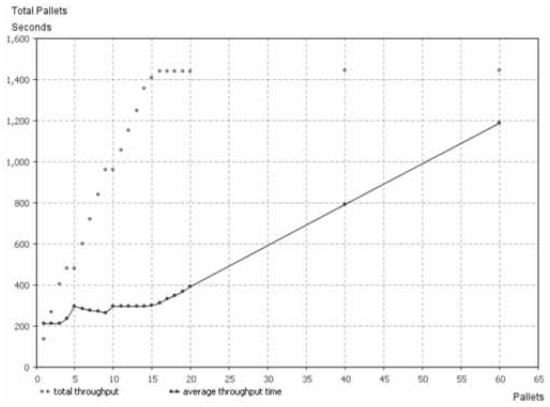

Figure 2. Total pallet throughput (red) and average throughput time of pallet (blue) vs. number of pallets in system.

A validation of the throughput time (for one pallet) can be calculated by hand. It sums up the travelling and processing times. One needs to calculate the travel through internal (2*2*m* + 3*1.6*m* + 1.6*m* + 1.6*m* + 1.6*m*) and connecting (6 * 0.4*m*) belts which sums up to 16*m* and divided this by the travel speed of 18 *m/min* (0.3*m/s*) which leads to 53.33*s*. The processing time of a pallet consist of shifting (2 * 5 * 2*s*) and processing (15*s* + 60*s* + 20*s* + 20*s* + 20*s*) actions which sum up to 155*s*. Thus a pallet needs at least 208.33*s* to travel the system - which correlates with the simulation results (Table 1).

C-Task: Pallet Number Optimisation. A parameter loop (Listing 1) was used in order to find an optimum with regard to maximal throughput and minimal throughput times. The found optimum number for pallets in the system is 16 and leads to a throughput time of 311.76 seconds. This can be taken from Figure 2: The total throughput of pallets remains constant (1440) for 16 and more pallets in the system and a higher number only leads to higher average throughput times. A throughput greater than 1440 pallets cannot be achieved sincethe system is limited by the processing times of the stations (which – as in Tas B - can be calculated by hand).

As suggested the simulation is run without measurement for a warm-up period (method `warmup()`) of two hours followed by a measuring interval of eight hours. The method `getState()` queries the execution state of the simulator. This is helpful during initial tests of the model so the loop can be exited earlier without having to wait for completion.

Pallets	1-3	4	5	15	20	40	60
TTpt	208	234	294	299	398	791	1194

Table 1. Average throughput time **TTp** for number of pallets.

```
1.  for (int i = 0; i < 20; ++i) {
2.  injectPallet(1);
3.  warmup();
4.  getEngine().getPresentation().refresh();
5.  measure();
6.  if (getState() == FINISHED) {
7.  break;
```

Listing 1. JAVA-fraction of the implemented loop.

Figure 3 presents the utilization-ratio (time occupied over total time) of the assemblers within the assembly stations. The ratio is plotted with respect to the number of pallets within the system (loading times neglected).

From Figure 3 the supportive role of A6 for stations A3-A5 becomes clearly visible (e.g. at 17-19 pallets in system). Further the ratio of A1 remains at 75% from 16 pallets on (explained by processing time of 15*s* vs. 20*s* for three A2 stations and each of A3-A5) and shows the same qualitative behavior as the total throughput in fig. 2 (as its load stays constant and every pallet has to pass through it).

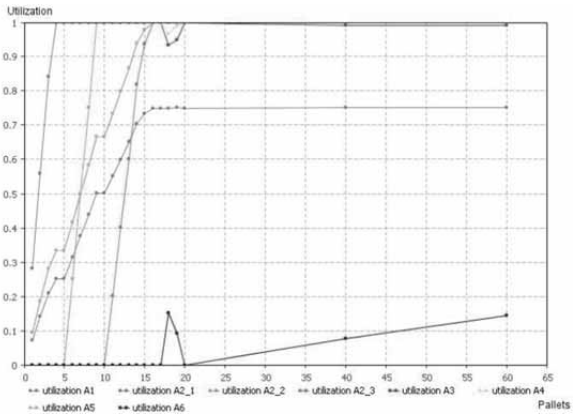

Figure 3. Utilization ratio of assembly stations with respect to pallets in system.

Resume. AnyLogic proved to be a stable and handy simulation environment for the realization of ARGES-IM Benchmark C2. Its main drawbacks lie with the post-processing of simulation results/-data for which workarounds needed to be found. Big advantages of the simulator are definitely the existing libraries in combination with the possibility for the user to add Java code at any given place and thus customize and enhance models, classes or predefined objects, as well as the platform-independence of AnyLogic.

Submitted: March, 2010
Revised: November 5, 2010
Accepted: August 30, 2011