*ARENA*Lib: A Modelica Library for Discrete-Event System Simulation

Victorino S. Prat Alfonso Urquia Sebastian Dormido
Departamento de Informática y Automática, ETS de Ingeniería Informática, UNED
Juan del Rosal 16, 28040 Madrid, Spain
E-mail: {vsanz, aurquia, sdormido}@dia.uned.es

Abstract

The design, implementation and use of ARENA*Lib is discussed in this manuscript.* ARENA*Lib is a new Modelica library for modeling, simulation and analysis of discrete-event systems (DES). This new Modelica library tries to replicate the functionality and capabilities of Arena: a general-purpose simulation environment supporting the process approach to DES modeling [1].* ARENA*Lib library will be released soon under the GNU General Public License (GPL). The library's general architecture, implementation and use are presented. Also the results of some model simulations, validated with Arena's results are discussed, as well as the future work and conclusions.*

Keywords: discrete-event; DES; Arena

1 Introduction

*ARENA*Lib is a new Modelica library for Discrete-Event System (DES) modeling and simulation.
The main objective of this library is to provide a modeling and simulation environment for DES using the process approach, opposite to other contributions in Modelica that use Statecharts[2, 3] or Petri nets[4] approaches.
*ARENA*Lib has also to be considered as a general-purpose tool, instead of application-oriented libraries developed by other authors (for example [5]). DES models built using *ARENA*Lib are completely written in Modelica language and they can be simulated using Dymola, as opposed to other approaches that require the combined use of different software tools (for instance, [6]).
When finished, *ARENA*Lib will be freely distributed under the GNU-GPL license.
*ARENA*Lib has been designed and implemented replicating the functionality and capabilities of Arena [1], which is a simulation environment for DES modeling and simulation using the process approach.
*ARENA*Lib validation is performed by comparison with Arena, when simulating the same systems.
In the next section, the capabilities and functionality of Arena will be discussed. It will be followed by a description of *ARENA*Lib general architecture, detailing the main components of the library. The connection between discrete and continuous systems using *ARENA*Lib will also be presented, as well as the random numbers and variates generation. After that, a simple case study will be explained in detail in order to show the use of the library, followed by the simulation and experiment setup. Finally an extended case study, future work notes and conclusions are provided.

2 Arena

The process approach describes the system from the entity perspective. The entity is the basic component of the simulation model.
To support this approach to DES simulation, Arena components are arranged into panels. The main panel is called Basic Process, and contains the main basic components for system simulation.
Panel components can be classified into two types: *flowchart modules* and *data modules*. Flowchart modules allow to describe the entity flow through the system (dynamic part of the system). They include Create, Process, Dispose, Decide, Batch, Separate, Assign and Record modules. Data modules describe the characteristics of system elements (static part), such as Entity, Queue, Resource, Variable, Schedule and Set modules. The procedure for building a model consists in drawing the flowchart diagram and configuring the required data modules. This procedure is analogous in *ARENA*Lib and will be detailed in Section 10.
At this moment, *ARENA*Lib implements the Create,

Process, Dispose and Decide flowchart modules and the Entity, Resource and Queue data modules. This implementation is described below.

3 *ARENA*Lib Architecture

*ARENA*Lib has been designed around the idea of implementing the Arena's Basic Process panel and being able to model and simulate the same kind of systems than Arena. As a consequence, it's architecture has been divided in two parts (*ARENA*Lib general architecture is shown in Figure 1a)):

- *The user zone*, that contains the Basic Process panel modules (flowchart and data), the Project model draft which is the start package for new systems' models and the tutorial package that includes some examples to help the beginner user to get used with the library. The structure of this zone is shown in Figures 1c),1b) and 1d).

- *The developer zone*, stored in the "src" package, contains all the internal models and functions used by *ARENA*Lib simulations. Details about this package are shown in Figures 1e) and 1f).

The user zone will be detailed in Section 10, while the developer zone will be discussed in the next sections.
The main problem that has been addressed in *ARENA*Lib is the management of the dynamical behavior of the components of the system. It includes the entity flow management, seizing and releasing of resources and the generation of the statistical indicators. This problem has been solved using dynamic-memory structures implemented as an static library (written in C language). The static library is connected to the Modelica code using Modelica's external-function interface. Packages Ext and DynMem contains the functions to access the static library (these packages are shown in Figures 1e) and 1f)).

4 Basic Process Panel

Analogously to Arena's process approach schema, *ARENA*Lib has been divided in two type of modules: flowchart modules and data modules. *Flowchart Modules*, that enables the modeler to describe the flow of entities in the system. It is composed by:

- Create module, represents the creation or arrival of entities to the system.

- Process module, simulates the point for entity processing.

- Dispose module, where the entities leave the system.

- Decide module, simulates a division in the flow of entities.

Data Modules, representing the components of every process in the system, such as:

- EntityType module, describes the characteristics of the entities.

- Queue module, represents the queue where the entities should wait for processing.

- Resource module, are the components of the defined processes.

All these modules will be discussed in Sections 6 and 7.

5 Interfaces

As previously mentioned the connection between modules is one of the main problems encountered during the development of the library.
One of the problems of module connection is that, for a given point in time, several and different entities could be arriving at the same module. This means that the same connector has to receive different entities from different modules of the system.
At the beginning, the approach taken to solve this problem was to implement an entity receival interface for each module. This interface was basically a text file in which the information of the received entity was written. The module that received the entity could read it from the file and process it. This solution was successful for small systems. However, the use of text files is very time consuming and the performance for large models was poor.
To solve that performance problem a dynamic memory interface was implemented for data exchange among modules. This interface consists in storing in an Integer variable in Modelica a number that corresponds to the pointer to a Dynamic Data Structure (DDS) previously created. These DDS's are stored in memory and correspond to the main data structures used in the system's simulation, such as queues and linked lists of elements. There is a DDS for managing queues of entities, lists of resources, lists of entity types, lists of statistical indicators and lists of attributes of an entity.

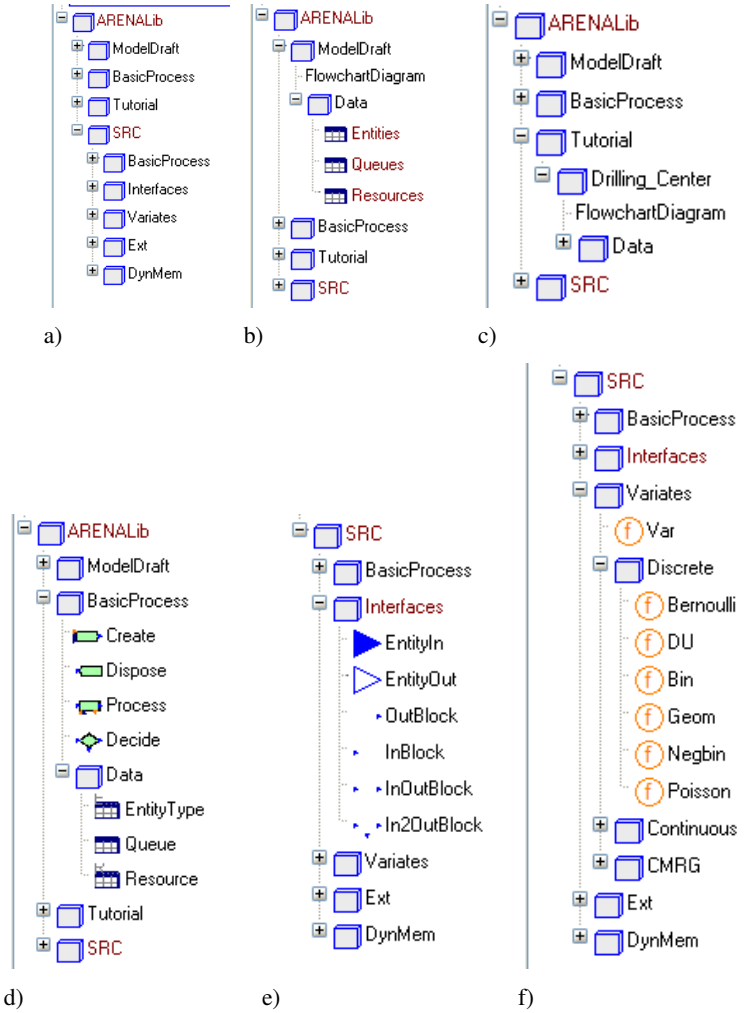


Figure 1: *ARENA*Lib architecture: a) General Architecture; b) Model Draft; c) Tutorial; d) BasicProcess; e) Interfaces; f) Random Variates

The performance was increased about 40 times from the text file's approach.

So the interface between modules is composed by an integer number, which corresponds to the pointer to the entity receival queue of a given module, and indicates the memory address for accessing the queue. Also, every module has a queue of outgoing entities for managing the order in which the entities are transfered to the next module.

For the development of the flowchart modules, some general interfaces have been implemented. The basic ones are the connectors for input and output entities.

Based on these basic connectors several models have been created to cover all the possible interfaces of the flowchart modules. Depending on the number of inputs and outputs of each module they can be classified into: InBlock (one input), OutBlock (one output), InOutBlock (one input and output) and In2OutBlock (one input and two outputs).

6 Flowchart Modules

Flowchart modules are used to model the flow of entities through the system. This entity flow begins in Create modules, and finish in Dispose modules. In this section, each of the currently implemented modules is detailed and it's use and configuration parameters are shown.

6.1 Create

As mentioned before, it allows to model the arrival of entities to the system. This arrival is expressed in form of an inter-arrival time between entities. The module extends the OutBlock interface module. The parameters of the module are the following:

- *EntityType*, is the type of the entities created in the module. This entity type has to be previously instantiated from an entity type data module.
- *Entity Generation Function*, represents the inter-arrival time. This time can be calculated from: (1) a probability distribution; (2) a constant expression and (3) a Modelica variable from any other model.
- *g1,g2 and g3*, are the parameters of the previous function. Depending on the distribution selected the meaning of each one changes.
- *TimeUnit*, is the local time unit. The inter-arrival time will be based on it.
- *Entities Per Arrival*, defines the number of entities generated in each arrival time.
- *Max Arrivals*, establishes the maximum number of entities generated in the module. If the number of entities reach this value, no more entities will be created.
- *First Creation*, sets the simulated time when the first entity is created.

6.2 Process

It defines a process. It extends the InOutBlock interface module. The parameters of the module are the following:

- *Name*, the name of the module.
- *Type*, can be standard and submodel. The standard defines a simple process configured by the module parameters. On the other hand, the submodel indicates that this module is just a mask for a more complicated process, composed by a DES itself. The submodel type is not fully implemented, but it can be easily done by using the Modelica's object oriented capabilities.
- *Action*, the possible values are Delay, Seize-Delay, Seize-Delay-Release and Delay-Release. Depending on the option chosen, the entity will seize a resource, be processed (delayed), and at the end release the resource.
- *Priority*, establishes the priority of entity selection from the waiting queue. This option is not implemented yet. The entity selected is the first in the queue (FIFO).
- *Resources*, is the list of the resources associated to the process. Each resource has to be previously declared using a resource data module.
- *Resource Quantities*, defines the quantity of each resource that has to be seized by the entity. If any of the resources does not have the specified quantity available then the seize operation fails and the entity must wait in the queue. This parameter must have the same length than the previous one.
- *Queue*, is the queue associated to the process. It has to be previously defined with a queue data module.
- *Delay type*, represents the processing time for an entity. It can be a continuous value or a probability distribution.
- *g1, g2 and, g3*, are the parameters of the *Delay type* function.
- *TimeUnit*, also represents the local time unit. It is the base time for the delay value.
- *Allocation*, determines the kind of process the module is simulating. Possible values are "Value Added", "Non Value Added", "Wait" , "Transfer" and "Other", and will influence the statistical results of the entities processed.

6.3 Dispose

It is the final point for the entities in the system: they are removed from the system and their statistical information is stored. For that reason, this module has no parameters.

6.4 Decide

It permits the modeler to simulate a division in the flow of the entities. Given a chance or a condition, the module decides the output connector the entity will leave the module through. It extends the In2OutBlock interface model. The parameters of the module are the following:

- *Type*, establishes the type of flow division, either by chance (percentage) or by condition. At this moment only the "by chance" option is implemented.
- *Percent True*, is the percentage of entities that will leave the module through it's True output connector.

7 Data Modules

Data modules represent the static components of the system. These are the entities themselves and the rest of the components that will interact with them along it's flow, such as queues and resources. At the present, three modules have been implemented and are detailed below.

7.1 EntityType

This data module defines the attributes of a type of entity. These attributes are the name, the picture of the entity (currently not implemented), and the costs associated to the entity.

7.2 Queue

It is used to describe process queues. The only type of queue currently implemented is the FIFO.

7.3 Resource

This module represents the resources used by the entities in the processes. An entity must seize (when needed) the resource in order to be processed. After processing, the entity can release the resource or not, depending on the kind of process performed.

8 Connection with other Modelica models

*ARENA*Lib modules can be connected to other Modelica models, so hybrid continuous-discrete systems can be easily modeled.
Arena provides the possibility of including continuous modules in the models, however this possibility is very limitated. Analogously, *ARENA*Lib provides interfaces in each module to perform a connection with any Modelica model, which is much more powerful than Arena's capabilities.
This connection interfaces give the modeler the possibility of configuring *ARENA*Lib module parameters from external Modelica models.
We can separate the connection interfaces in two:

- Integer ones, that represent the transfer of entity related events between discrete and continuous systems. The input Integer connector is used to tell the system that an entity has arrived or that an entity has finished the processing (in the Create and Process modules). And the output connector (in the Process module) is used to tell the continuous system which entity (with it's serial number) has seized the resource and is ready to be processed.

- Real ones, are just values for the rest of the available parameters of the modules, such the time for the first arrival, the maximum arrivals, the priority, etc.

On the other hand, each discrete module provides information that can be accessed using the Modelica dot notation. The list of variables that can be accessed for each module is displayed in the information icon of the module's model.

9 Random Variables

A key point in DES simulation is the random number generation. This section has been divided into two parts to separate the random number generation (observations of the $U(0,1)$ distribution) from the variate generation.

9.1 Random Number Generation

Due to the DES dependence on stochastical distributions and to ensure good simulation results, it is very important to have a good source for pseudo-random numbers, to build statistically good random variates[7].
Also, in order to be able to analyze and validate the results from *ARENA*Lib in comparison with the ones obtained from Arena, the same pseudo-random number generator has been implemented.
This random number generator is called CMRG (Combined Multiple Recursive Generator) [8] and has a period length of 2^{191}. This period length can be divided into disjoint streams, each of them of length 2^{127}.
*ARENA*Lib associates one of those streams with each Random Variable, so the random number stream independency can be ensured for variate generation.
The implementation of the CMRG has been done translating the implementation in C, done by Pierre L. Ecuyer (available on the web at [9]), into Modelica code. In this way, the generator is available for anyone outside the *ARENA*Lib environment. The only remarkable thing for it's usage is the management of the seed, which is read from a text file and updated every time a new stream is generated.

9.2 Random Variates Generation

To provide the modeler enough functionality for modeling and simulating many kind of systems, some of the most commonly-used probability distributions have been implemented and included in the library.
These distributions are functions that use the stream created by the CMRG to generate random variates. The output variables of all the distribution functions are the variate's value and the updated random stream, that will be used to obtain more new variates. The input variables of the functions are the following:

9.2.1 Continuous Probability Distributions

- Uniform (min,max)
- Exponential (mean)
- Normal (mean,variance)
- LogNormal (mean,variance)
- Triangular (min,mode,max)

9.2.2 Discrete Probability Distributions

- Bernoulli (p)
- Discrete Uniform (min,max)
- Binomial (n,p)
- Geometric (p)
- Negative Binomial (n,p)
- Poisson (alpha)

10 *ARENA*Lib Use

In this section the use of *ARENA*Lib will be introduced by means of a simple example. The development of the model, the experiment setup, the analysis of the results an the validation with Arena are explained in detail.

10.1 Model Description

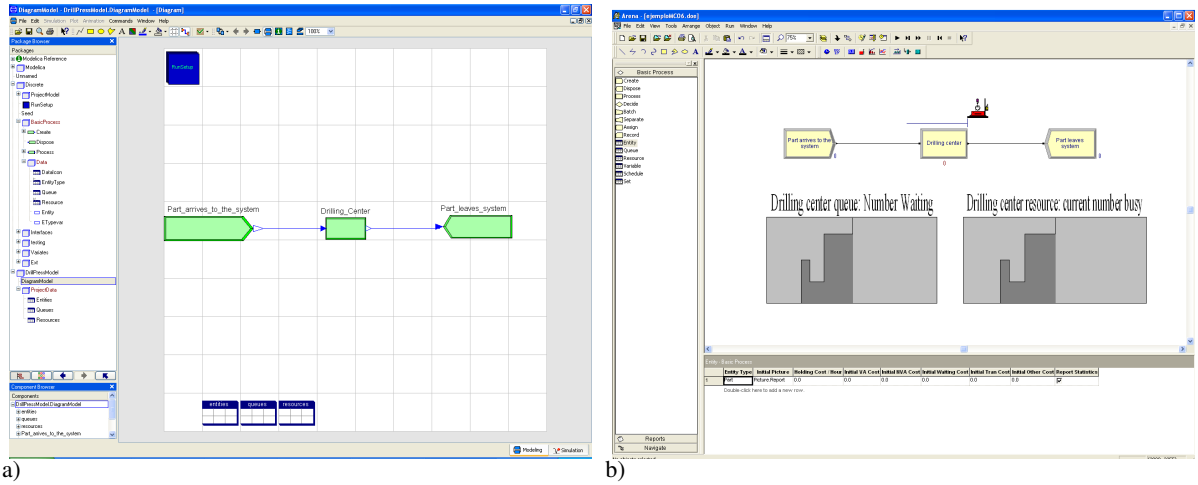
It is a very simple case of a processing system. The modeled system consists on a Drilling Center where the parts arrive, are drilled (processed) and leave.
The flowchart components of the system are a create module, that represents the parts arriving to the center, a process module, which is the drilling center itself and finally the dispose module where the parts leave the system. The flowchart diagram is presented in Figure 2.
The data modules are: (1) the entity type that represents the processed parts; (2) the drilling press viewed as a resource; and (3) the queue associated to the press.
For building this model in *ARENA*Lib we should use the package ModelDraft, duplicate and rename it to match our model requirements. This package includes all the basic components necessary to simulate the system.
Inside the ModelDraft there are two structures, the flowchart diagram that will be used to compose the system's structure and the data package which contains models for structuring all the needed data modules of our system.
For building the Drilling Center in *ARENA*Lib, an entity type, a resource and a queue modules have to be inserted in their corresponding data models, and then the flowchart diagram has to be drawn by drag a drop of a create, a process and a dispose module.
The last step is to configure the parameters of each data and flowchart module, as desired, to match the requirements of the system.

10.2 Simulation and Experiment Setup

	Arena		*ARENA*Lib
Indicator	Average	Half Width	Average
Part.NumberIn	20010		19887
Part.NumberOut	20006		19887
Part.VATime	3.3254	0.01574	3.3385
Part.WaitTime	3.4822	0.24270	3.4571
Part.WIP	1.3622	0.06354	1.3527
VATimePerEntity	3.3254	0.1574	3.3385
WaitTimePerEntity	3.4822	0.24270	3.4571
TotalTimePerEntity	6.8077	0.24771	6.7956
Queue.NumberInQueue	0.69695	0.05623	0.68754
Queue.WaitingTime	3.4827	0.2427	3.4571

Table 1: Results of the drilling center simulations

When having the system's model, and after having configured all the parameters of the modules, a simulation experiment can be run.
There are two basic parameters for each simulation. The simulation time and the global time unit. The first one establishes the duration of the simulation experiment, and the second one is the time unit (seconds, minutes, hours,etc.) in which the simulation time is established. These two parameters will influence the time unit parameter of the modules in the system, translating the values in each module to match the simulation time.



a) b)

Figure 2: Drilling center model composed using: a) *ARENA*Lib; and b) Arena

When the simulation time is finished, the statistical results of the run are written to a file named "SimResults.dsc". After that, the only remaining variable in the system is the CMRG seed. This ensures that several experiments of the same model will, each of them, use different pseudo-random numbers.
Results from the Drilling Center simulation are shown in Table 1. These results have been obtained using Dymola.
Several experiments can be easily configured using Modelica's scripting facilities.
Other result analysis capabilities are the plots. Every module has several variables that can be plotted at the end of the simulation and permits the analysis of the evolution in time of the experiment. This can be very useful to detect peak loads or compare several parameters in the system.

Indicator	Arena		*ARENA*Lib
	Average	Half Width	Average
Bottle.NumberIn	637		650
Bottle.NumberOut	623		632
Bottle.VATime	6.1618	0.35771	6.405
Bottle.WaitTime	35.894	5.2451	36.237
Bottle.WIP	5.3305	(Corr)	5.5132
Labeling.NumberIn	659		677
Labeling.NumberOut	645		660
Labeling.VATimePerEntity	5.3348	0.13555	5.3026
Labeling.WaitTimePerEntity	34.697	(Corr)	34.796
Labeling.TotalTimePerEntity	40.031	5.6018	40.098
Labeler_q.NumberInQueue	4.5534	(Corr)	4.7017
Labeler_q.WaitingTime	34.723	(Corr)	34.839
Cleaning.NumberIn	22		28
Cleaning.NumberOut	22		27
Cleaning.VATimePerEntity	19.202	(Insuf)	20.44
Cleaning.WaitTimePerEntity	1.0079	(Insuf)	0
Cleaning.TotalTimePerEntity	20.210	(Insuf)	20.44
Cleaner_q.NumberInQueue	0.0044	(Insuf)	0
Cleaner_q.WaitingTime	1.0079	(Insuf)	0

Table 2: Results of the Bottle Filling simulations

11 Case Study

As a case study, a Bottle Filling process is discussed. This process consists in a tank which fills bottles with liquid. Once a bottle is full, it is labeled and controlled in two quality control processes. The bottles that pass the first control are considered as first class bottles and the ones that pass the second control are considered as second class bottles. Any bottle that doesn't pass the second quality control process is cleaned and relabeled again.
The tank has been modeled as a continuous system which fills the bottles at a constant rate. Every 400 time units the tank is refilled to it's maximum level.
The flowchart diagram of the process is displayed in Figure 3, modeled either in Modelica and Arena.

The results of the simulation and the validation data, from the comparison with the Arena's results, are presented in Table 2. Also some plots from Dymola's simulation results are shown in Figure 4.

12 Future Work

The main task for the future work will be the Basic Process panel development completion, including the modules still not implemented and the rest functionalities for the existing ones.
Other problems, that actually delay the development of the whole library, are the management of different data types for attributes and variables and the implementation of variable size matrices, for example, when introducing entity attributes that can be modified dynami-

cally during the simulation by any module, or the use of system variables whose value and size can change similarly to the attribute ones.
The solution for these problems is still in progress but will be solved soon, enabling the further development of the whole library.
The tutorial package will also be completed with more examples as well as the information pages for every module, either in the user zone and the development zone.
Visual representation of the simulations will also be studied.

13 Conclusions

A new Modelica library has been designed and implemented, offering the possibility of modeling DES and hybrid continuous-discrete systems in a simple way. This new library is based on Arena, a simulation environment for DES.
This new library is completely compatible with the rest of the Modelica's components.
Several experiments have been performed obtaining successful results. Validation has been done using Arena for comparing results.

References

[1] Kelton W.D, Sadowski R.P, Sturrock D.T. Simulation with Arena (Third Edition). McGraw-Hill, 2004.

[2] Otter M, Årzén K.-E, Dressler I. StateGraph - A Modelica Library for Hierarchical State Machines. In: Proc. of the 4^{th} Int. Modelica Conference, 2005, pp. 569–578.

[3] Ferreira J. A, Estima de Oliveira J.P. Modelling Hybrid Systems using Statecharts and Modelica. In: Proc. of the 7^{th} IEEE Int. Conference on Emerging Technologies and Factory Automation, 1999.

[4] Mosterman P.J, Otter M, Elmqvist H. Modelling Petri Nets as Local Constraint Equations for Hybrid Systems using Modelica. In: Proc. of the Summer Computer Simulation Conference, 1998, pp. 314–319.

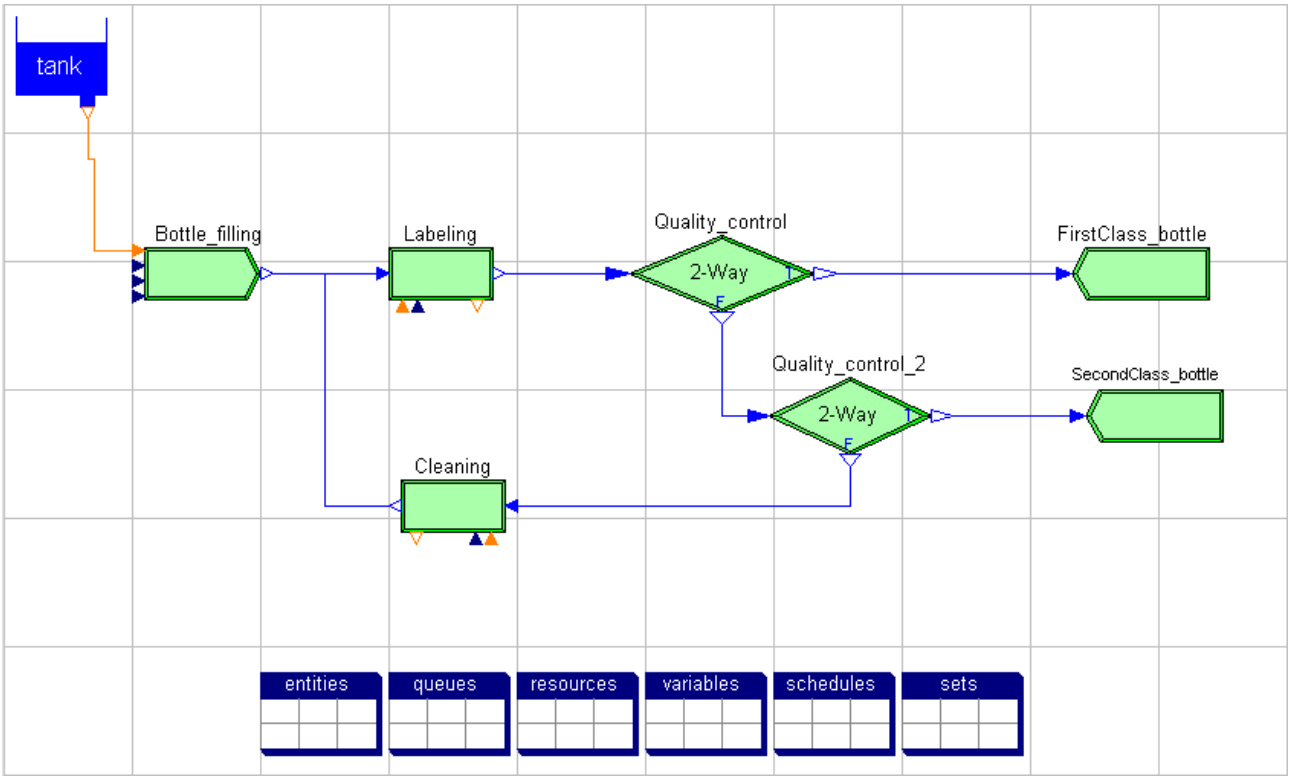
[5] Färnqvist D, Strandemar K, Johansson K. H, Hespanha J.P. Hybrid Modeling of Communication Networks Using Modelica. In: Proc. of the 2^{nd} Int. Modelica Conference, 2002, pp. 209–213.

[6] Remelhe M.A.P. Combining Discrete Event Models and Modelica - General Thoughts and a Special Modeling Environment. In: Proc. of the 2^{nd} Int. Modelica Conference, 2002, pp. 203–207.

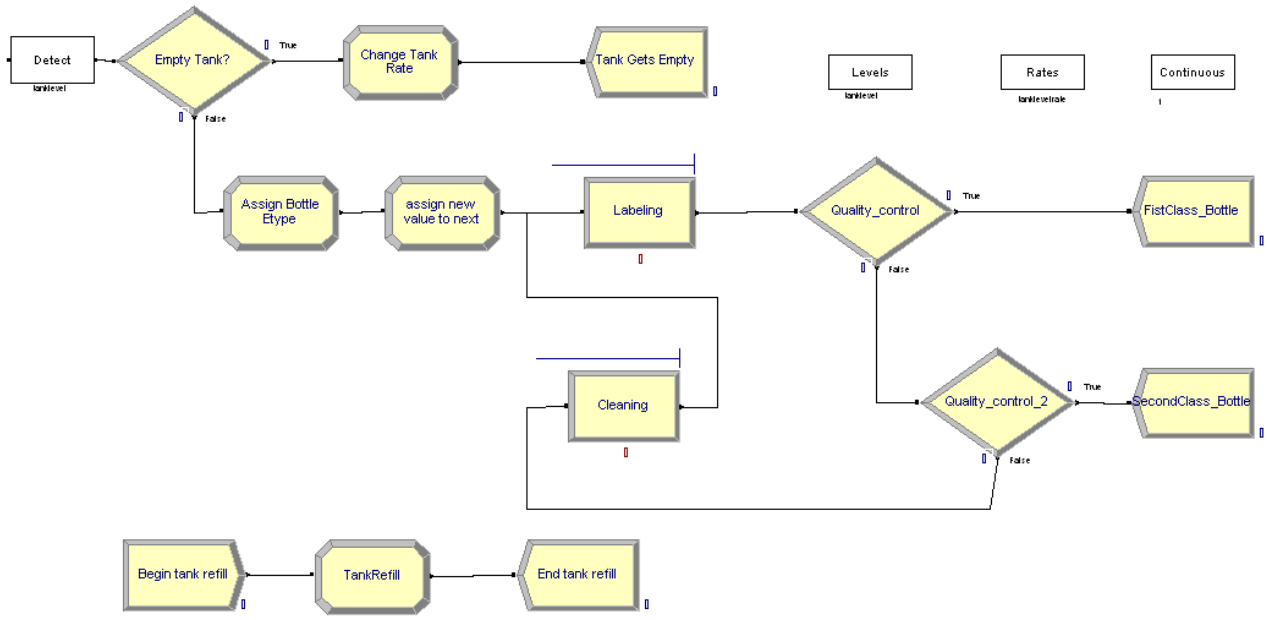
[7] L'Ecuyer P. Software for uniform random number generation: distinguishing the good and the bad. In: Proc of the 33^{rd} conference on Winter simulation, 2001, pp. 95–105.

[8] L'Ecuyer P, Simard R, Chen E. J, Kelton W. D. An Object-Oriented Random-Number Package with Many Long Streams and Substreams. Operations research, vol. 50, 2002, pp. 1073–1075.

[9] L'Ecuyer P. CMRG source code web page. http://www.iro.umontreal.ca/ lecuyer/myftp/streams00/ July, 2006.



a)



b)

Figure 3: Bottle Filling model composed using: a) *ARENA*Lib; and b) Arena

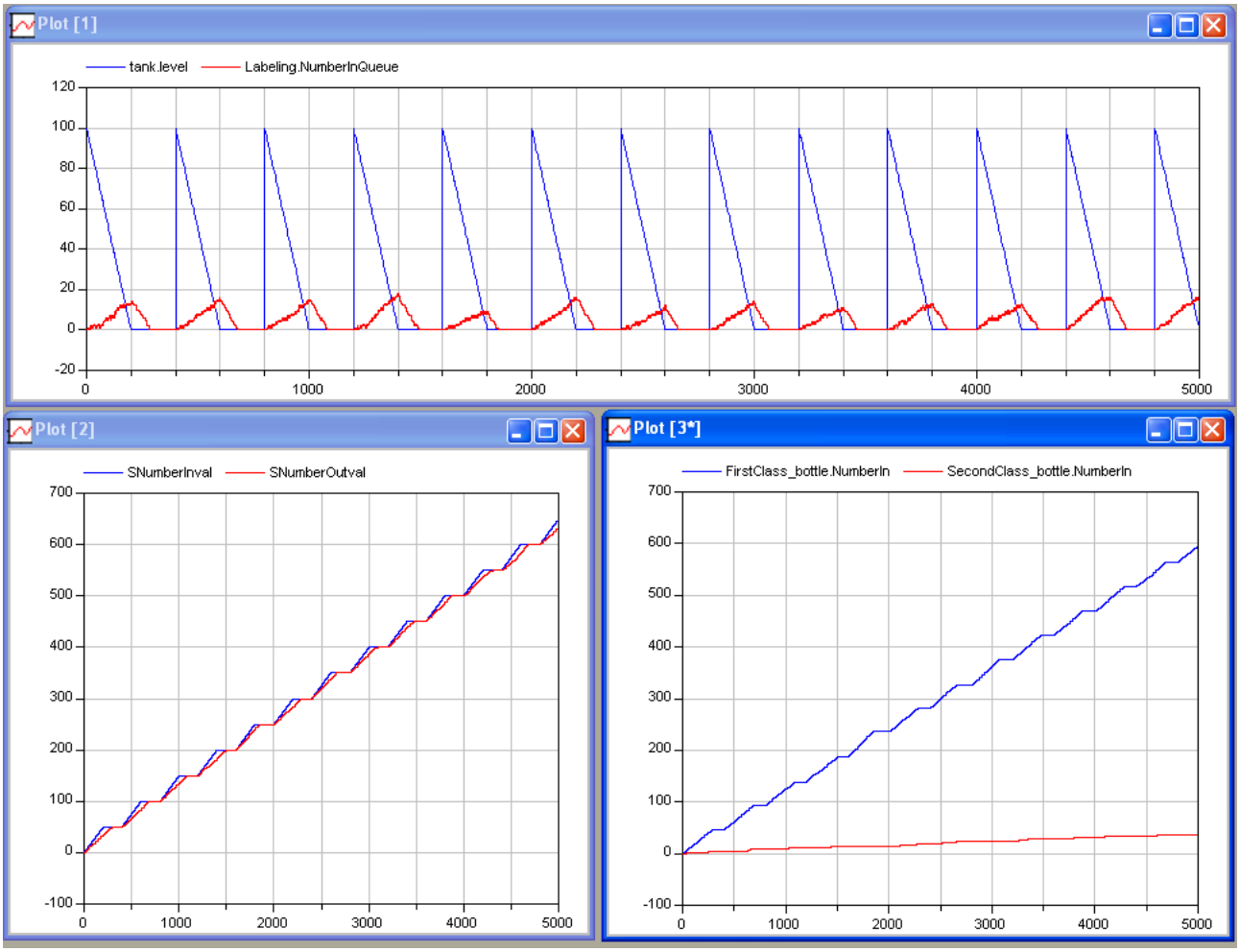


Figure 4: Bottle Filling model result plots using Dymola