ARGESIM BENCHMARKS

Event-based and State-Automata-based Modelling of FMS

A Comparative Case Study of Flexsim, Dymola and Matlab/Stateflow based on the ARGESIM Benchmark C2

Sebastian Schreiber, Mike Barth, René Nicolaus, Markus Schleburg, Alexander Fay
Helmut-Schmidt-University, Institute for Automation Technology, Hamburg, Germany
{sebastian.schreiber; mike.barth; rene.nicolaus; markus.schleburg; alexander.fay}@hsu-hh.de

Simulation Notes Europe SNE 20(1), 2010, 38-44, doi: 10.11128/sne.20.bn02.09968

By implementing the ARGESIM Benchmark C2 "Flexible Assembly System", the simulation system *Flexsim*, the *Matlab/Simulink*-toolbox *Stateflow*, and the equation-based modelling language *Modelica* are compared to each other. Based on the different modelling techniques, the systems will be described and analysed from an automation point of view. Subsequently, the modelling approaches of state-automata (*Stateflow*) as well as object-orientation (*Flexsim*) and equation-based modelling (*Modelica*) are reflected. The analysis includes (1) the time and efforts that are necessary for the modelling process itself, (2) the complexity of the implementation, (3) the possibility to analyse the simulation results, and (4) the possibility to separate control algorithms and controlled system in the implementation.

Introduction

A wide choice of commercial software tools for the simulation of material flow is available on the market (see e.g. [6]). Within this work, the authors analyse how software tools which are commonly used within the automation community can be used for the modelling and simulation of material flow problems. In this context, modelling and simulation methods are important for testing control algorithms without the need of having access to real systems. Therefore, the simulation environment *Flexsim* [1], the *Matlab/Simulink*-Toolbox *Stateflow* [2], and the modelling language *Modelica* [3] implemented in *Dymola* [4] have been selected and applied to the ARGESIM C2 Benchmark "Flexible Assembly System" [5]. There has not been a publication of C2 Benchmark results concerning these tools before.

The article is structured as follows. First, the Benchmark C2 will be introduced by presenting the structure of the target system and its modelling tasks. Subsequently, the different modelling techniques as well as the respective implementations of the C2 Benchmark are explained for Flexsim (Section 2), Modelica (Section 3) and Matlab/Stateflow (section 4). Finally, all three approaches are compared with respect to the required time for modelling, the complexity of the implementation, the possibility to analyse the simulation results, and the possibility to separate the control algorithms from the controlled system. The latter requirement is essential for the test of control algorithms on the model.

This article is an extended and reviewed version of [15], which was presented at the ASIM ST/GMMS-Workshop 2010.

1 Benchmark ARGESIM C2 "Flexible Assembly System"

The benchmark C2 describes a flexible assembly system and has originally been proposed by the ARGESIM in [5]. Its objective is to test different simulation systems with regard to their ability to define and to combine sub-models, as well as to formulate complex control strategies. The VDI guideline 3633 [7] uses this benchmark as an application example for the efficient handling of simulation studies. The average throughput time and the optimal number of pallets are the comparable target results.

The pallets are used to transport single parts through an assembly system, which is shown Figure 1. The number of pallets to be used is one of the parameters that is kept constant during a single simulation run. The system can be separated into eight sub-models placed along a main conveyor belt. Each of these sub-models contains an assembly station.

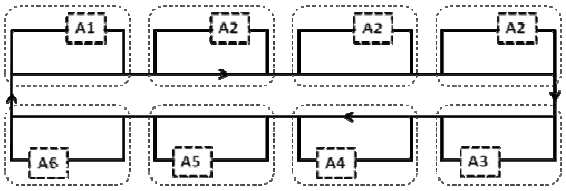

Figure 1. Description of the assembly system.

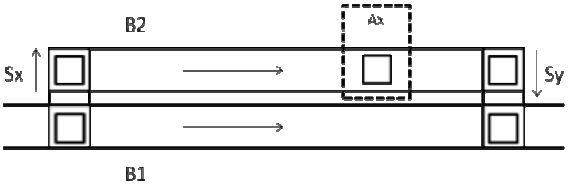

Figure 2. Description of a subsystem.

Station A1 is responsible for unloading finished parts from the pallets and loading new ones. The stations A2 to A6 are each responsible for a predefined processing step. Every part has to pass the processing steps A2 to A5 before it is finished. Station A6 acts as a back-up function and should be used if one of the stations A3 to A5 is busy or down. The processing sequence has to start or end within station A2. The sequence of A3 to A5 is not predefined in this context.

Each subsystem (Figure 2) consists of two conveyor belts `B1` and `B2`, two shifting units `Sx` and `Sy`, and an assembly station `Ax`. If a pallet arriving on `B1` needs to be processed into `Ax`, the shifting unit `Sx` pushes it onto `B2`. If it does not need to access `Ax`, the pallet remains on `B1`. The transportation area between `Sx` and `Ax` on belt `B2` can be used as a buffer. All finished pallets are pushed back to `B1` through `Sy` and have priority to those on `B1`.

2 Simulation environment Flexsim

Flexsim is an object-oriented system for discrete-event simulation and simulation of continuous-flow processes. The available standard library contains more than 40 objects, of which 26 are discrete ones. In May 2010, the current version 5.0 of Flexsim has been released.

All objects refer to one of the so-called "super-classes" `FixedRessources` (e.g. source, sink, or machine) or `TaskExecuter` (e.g. operator, vehicle, or crane). Elements modelling the material flow are called `Flowitems` and form a special class within the system. The objects are connected to each other via input and output ports, where the connection itself represents information and/or material flow.

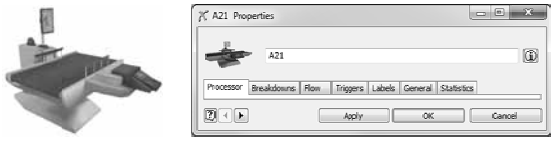

Figure 3. Description (left) and parameterization (right) of a `Processor` object.

Every object allows several possibilities for parameterization. As an example, Figure 3 shows a `Processor` (assembly station A2) and its properties. These properties support all the relevant aspects for modelling a material flow system, such as `Processor`, `Flow`, and `Trigger` (e.g. events `OnEntry` or `OnExit`), and are used to define the element's behaviour. In this context, Flexsim offers pre-defined samples as well as the possibility to define new models within the programming languages C++ or Flexscript. Compared to C++, the use of Flexscript offers the advantage that the simulation model does not have to be compiled before it can be used.

For implementing the benchmark in Flexsim, only standard library elements have been used to allow an efficient traceability. Flexsim itself allows to build own objects and libraries. As an example, Figure 4 shows the implementation of the first assembly station A2. For the sake of visual clarity, the connections between the elements are hidden and the single elements are separated in this Figure. Because material flow is only modelled as an (information) connection in Flexsim, the separation of the elements has no influence on the processing of the simulation model. Figure 4 contains several instances of the objects `Conveyor` and `Processor`. Furthermore, there are some more objects used within the model, e.g. `Combiner`, or `Separator`, which are not displayed here.

For implementing control algorithms, Flexsim supports several possibilities. For example, so-called `Photo Eyes` can be placed on every `Conveyor`, e.g. representing sensor information, or special triggers of an object can be used, e.g. a flow item leaving a con-

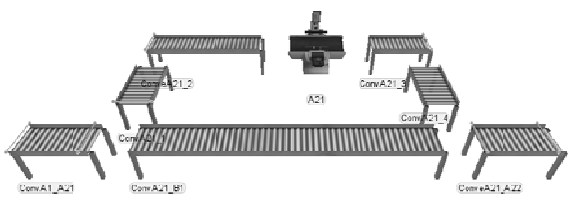

Figure 4. Implementing the first sub-model A2 in Flexsim.

veyor. The listing for the latter possibility is shown in Listing 1. Every `Flowitem` reaching the end of this conveyor triggers the control decision. It represents the decision whether a pallet should be processed in `Ax` (shift to `B2`) or not (stay on `B1`), depending on the pallet's properties, where: (1) is a reference (`treenode`) to the current conveyer `ownerobject(c)` in the global treeview, in which all elements of the simulation model are listed; (2) is a reference to the pallet currently on this conveyor by `parnode(1)`; (3) reads the `target` label on the pallet by `getlablenum(...)`; (4) compares the target information; and (5-6) send the pallet to the defined ports. The example is written in Flexscript, which has a syntax similar to C++.

```
1 treenode current = ownerobject(c);
2 treenode item = parnode(1);
3 int target=getlabelnum(item,"target");
4 if (target==Ax) {
5   return 1;
6 } else return 2;
```

Listing 1. Trigger `Flow-Output-Send to port` of a conveyor in Flexsim.

The parts and pallets are modelled as different `Flowitems`. Each `Flowitem` can carry a number of so-called `Labels`, e.g. information of a RFID tag, which can be analysed and manipulated through every object. The part objects are generated within station A1 and are combined with a pallet for transportation. After being processed on every necessary machine the pallet returns to station A1, in which the part object is separated from the pallet object.

For analyzing the simulation results, Flexsim offers an included module as well as an interface to MS Excel, which was used within this work.

3 Modelica and Dymola

Due to its object-oriented equation-based (OOE) architecture, Modelica is well suited for the modelling of continuous physical systems. By using the module *State-Graph* ([10], [11]) of the Modelica Standard Library (MSL) [9], it is also capable of discrete-event simulation. The Benchmark C2 can be classified as a hybrid model, which represents a combination of both discrete and continuous modelling (e.g. [12], [13]).

Within Modelica, it is possible to define so-called real-world models through known mathematical relationships. By defining interfaces of different variable types (e.g. real, boolean, integer), the combination of

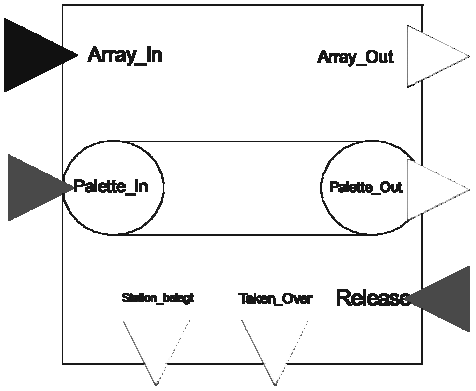

Figure 5. Screenshot of the basic object `Conveyor` in Modelica.

time-based equations and decision routines can be implemented. While applying tests on event-driven models it is often necessary to allow user interaction in parallel to a running simulation. These can be initiated together with the *User-Interaction* (UI) module [9] and a built-in real-time option. The UI module is part of the *MSL*. As an example, a boolean variable (e.g. start of a conveyor) can be changed manually and the user is able to visualize additional feedback variables at the same time (e.g. photo eye on this conveyor).

Following the object-oriented approach of Modelica, the objects of the benchmark can be separated into physical objects (e.g. conveyor, pallets) and decision (control) objects. A combination of physical objects represents the controlled system, while the distributed control decisions are modelled as separated objects. Figure 5 shows the object "conveyor" with its different types of interfaces. There is one array input and one output (SISO) for the exchange of status information of the current pallet. In this context, a capacity of six real variables has been implemented. Furthermore, two interfaces represent boolean type information whether there is a pallet waiting for takeover (input) or is ready for takeover (output). Another input informs the conveyor about the release of a waiting pallet. The output interfaces on the bottom of the conveyor are also of boolean type and represent the status of the object (idle/busy) as well as a trigger signal to the previous object sending a release flag.

The whole modelling of the benchmark is based on the object `Conveyor`. To each object, a range of physical parameters is assigned, e.g. length and conveyor belt speed, processing time, as well as the respective process number in case of representing an assembly station. For modelling the elements `Sx` and `Sy` (see

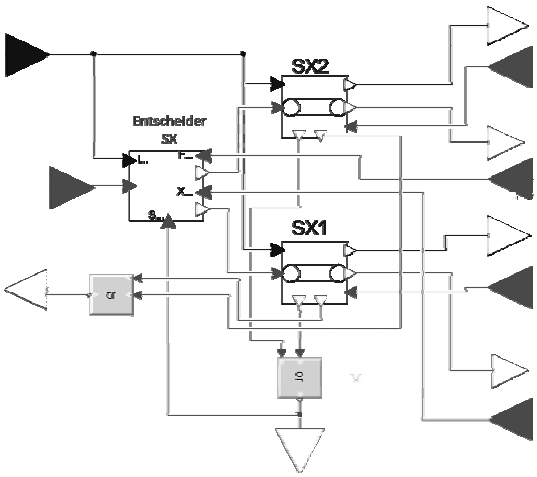

Figure 6. Description of an element `Sx` in Modelica.

Section 1), two `Conveyor` objects and a decision block have been combined to a single object (see Figure 6). For the initialisation phase of each simulation run, a separate module was modelled, which releases a defined number of pallets and later deactivates them.

4 Matlab-toolbox Stateflow

Stateflow is part of the simulation environment *Matlab/Simulink* and supports the modelling and simulation of state automata. A detailed description can be found in [8]. In this context, state-charts are used for modelling state automata whose basic elements are shown in Figure 8. The main modelling elements are states and transitions, which can be grouped as charts or superstates. States can be modelled as exclusive (OR) or parallel (AND) with respect ot their activation within a chart. State transitions are implemented as `event[condition]{condition_action}/transition_action}` and can therefore be triggered by an event and/or a fulfilled condition. In addition, `Truth Tables` can be used for pre-defined control decisions and corresponding actions.

The system behaviour of the benchmark needs to be modelled as enclosed states. For this purpose, the

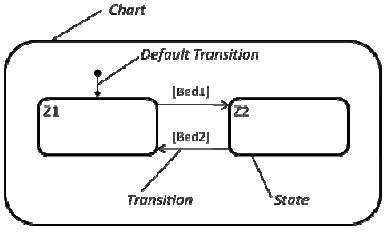

Figure 8: Basic elements in Matlab/Stateflow.

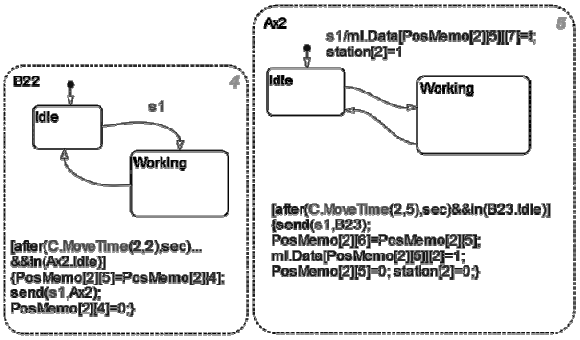

Figure 7. Implementation in *MMatlab/Stateflow.*

conveyors are separated into segments (charts) having the length of a single pallet. Each segment can be empty (`Idle`) or can carry a pallet (`Working`). A pallet receives a unique identifier (ID) which is transported through all segments and is assigned by a global table. Because there is no "physical" pallet object, it is called "virtual" in contrast to the `Flowitems` within Flexsim.

Figure 7 shows the state transition from `Idle` to `Working` within segment `B22` that is triggered trough the event `s1`, which will be activated through the previous segment. The opposite state transition (see Listing 2) consists of: (1) two conditions, where `MoveTime` is a function that generates the processing time for the segment; (2) virtual transport of the pallet, where `PosMemo` is a global 3D-array in Matlab containing the information of all segments in the system (here `[2][4]` describes the segment `B22`), and the status of the current pallet in this segment; (3) release of the event for the next segment `Ax2`; (4) clearing the stored information in `PosMemo` for the segment `B22`.

```
1 [after(MoveTime(2,2),sec)&& in(Ax2.Idle)]
2 {PosMemo[2][5]=PosMemo[2][4];
3 send(s1,Ax2);
4 [PosMemo[2][4]]=0;}
```

Listing 2. Listing of transition `Working` to `Idle` of segment `B22` (see Figure 7).

The structure of a two state segment within one chart can also be used for the assembly station (see Figure 7, right). A segment of an assembly station differs by two aspects from conveyor segments (see Listing 3). First, (4) manipulates the information stored on the current pallet at `PosMemo`, where `[2]=1` indicates that the necessary processing in A2 is done. Second, `station[2]` (6) generates a status information for the assembly station, which is an output variable to *Simulink* and is used for user interaction.

```
1 [after(MoveTime(2,5),sec)&&in(B23.Idle)]
2 {send(s1,B23);
3 PosMemo[2][6]=PosMemo[2][5];
4 ml.Data[PosMemo[2][5]][2]=1;
5 PosMemo[2][5]=0;
6 station[2]=0;}
```

Listing 3. Listing of transition `Working` to `Idle` of segment `Ax2` (see Figure 7).

As already mentioned, Stateflow is embedded into the environment of Matlab/Simulink (see Figure 9). All necessary parameters, e.g. processing times, or the numbers of pallets, are implemented in the form of Simulink inputs. The user interaction in form of displays is shown on the right side. The array `PosMemo` itself is stored on the level of Matlab for later analysis, e.g. by use of Matlab functions.

5 Comparison of modelling approaches

As shown before, the modelling and simulation of the benchmark aspects can be implemented within all three modelling techniques analysed here. The main differences are: (1) the time which was necessary for modelling the system, (2) the complexity of the implementation (3), the possibility to analyse the simulation results, and (4) the possibility to implement the control algorithms and the controlled system separately within the tools. The comparison of these four aspects is done from an automation technology point of view. A brief summary is shown in Table 1 at the end of this section.

There is a considerable difference between the several approaches with respect to the time necessary for modelling (1). Based on the well-suited library for material flow processes, the modelling in Flexsim just comprises the identification of suitable objects, as well as the creation of the necessary connections between them. The library also contains the necessary objects for generating elements (e.g. queue, sink), as well as for combining and separating parts and pallets. The benchmark constraints, e.g. a fraction of pallet capacities for the conveyors, can be implemented correctly. This is different concerning the other approaches. For modelling a conveyor in Modelica or Stateflow, it needs to be split up into an integral number of segments. An object (Modelica) or a state (Stateflow) represents each of these segments, with each of them having the capacity of one pallet. It would be possible to model these segments with a lower capacity, to reach a "continuous-like" behaviour as it is done in Flexsim. Nevertheless, this would

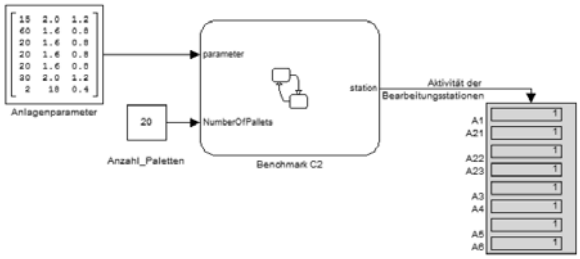

Figure 9. Embedding of *Stateflow* in *Matlab/Stateflow*.

lead to two main problems: First, it would increase the number of objects/states, including more effort for implementation. Second, the model of a pallet would have to be changed in a way that it can be split up and "cover" more than one object/state. This would result in a higher complexity, especially for keeping the inner-coherence of a pallet. Hence, this alternative was not further considered The modelling in Stateflow compromises of using similarly structured charts, which only contain two states and two transitions. When events are sent between the objects, there is no necessity for connecting the charts, because only the recipient needs to be manipulated. This allows a high degree of reuse. For modelling in Modelica, the reuse of objects is achieved by building a basic conveyor object. The effort for modelling this object was higher, compared to the other approaches. Due to the inherited possibility of building up the whole system based on this object, this once-only-effort could be justified.

This modelling has been time consuming for each of the three modelling techniques, but the effort has been well spent: the carefully created objects result in low complexitiy of the C2 Benchmark implementation. The instantiation and parameterization of the objects is the main aspect to be considered. Flexsim, as a commercial simulation software, covers this in an intuitive dialog-based manner. In addition, the already mentioned possibility of describing the control decisions is helpful. Connections between the objects can be implemented through interfaces (ports) on which the control decisions are based on. The implementation in Modelica is mainly based on the combination of previous modelled classes. It is possible to build and reuse a module for representing a whole sub-system, as required in the benchmark description. This can be done by parameterization, e.g. processing and transportation times, or the defined stage in the process. The modules are combined through connecting the pre-defined type-safe inter-

	Flexsim	Modelica/ Dymola	Matlab/ Stateflow
1. necessary time for modelling	object library (+)	basic element (–) combination (+)	state automata (+)
2. implementation complexity	instantiation, connection (+)	instantiation, connection (+)	marking (–) connection (+)
3. ability of analysing	Labels, interface for MS Excel (+)	"virtual" objects fix data structure in early phase (+/–)	„virtual" objects global array (+/–)
4. separation of system and control	combined behaviour-description logic (–)	capsulation of decision modules (+)	separation between Simulink and Stateflow (+)

Table 1: Comparison of the modelling approaches: (+) positive, (+/–) neutral, (–) negative influence.

faces. Before building the connections in Stateflow, all charts have to be placed and clearly marked (ID). Due to the lack of direct connections between the charts and the necessity of a global assignment array, this results in being inflexible, e.g. concerning later revision. The manipulation of the transitions is, in comparison to this, straightforward.

The necessary effort for providing and collecting the relevant information is a very important aspect. Especially the evaluation of the ability to analyze simulation results (3) needs to be considered. Here Flexsim performs well. It is possible to store data within every object (`Label`), esp. on the `Flowitems`. This supports an easy verification of the process variables, as well as a local reasoning and manipulation. In addition, a further development of the models, e.g. extended data collection, can be implemented with few effort. This approach can be realized in Modelica too. However, with respect to information transmission, analyzing, and manipulation, the effort is considerably higher. Furthermore, it is not possible to verify the simulation model without knowing the internal structure of the "virtual" pallet objects. The same problem occurs within Stateflow, where all the relevant information is stored within a global data table. It is important for both approaches, Modelica as well as Stateflow, to define the necessary information before starting the modelling phase. For example, a later extension of the array size in Modelica affects each interface. As already mentioned this is not the case in Flexsim. For analyzing simulations results, each of the three approaches is well suited. A clear preference only depends on the personal aspects and cannot be testified at this point.

Another relevant point is the separation of control algorithms from the controlled system (4), which is especially important in the field of automation. This separation can be implemented in both Modelica and Stateflow. The implementation in Modelica already encapsules the control decisions within the shifting modules. For Stateflow, this could be reached by communicating the relevant information to Matlab/Simulink where the reasoning can be implemented, e.g. using function block diagrams. This is not possible within Flexsim, which is based on the concept of combined behaviour-description logic. This works quite well for acting within this simulation environment but not for testing or rather verifying new control algorithms. In the field of automation technology, these algorithms are usually implemented in different programming languages, e.g. using IEC 61131-3 [14], which are not supported in Flexsim.

6 Summary and Outlook

The modelling approaches as well as the implementation of the benchmark have been described for the three selected simulation environments. The comparison has shown a clear distinction especially in the effort necessary for modelling and for the analysis of the simulation results.

In summary, the three modelling approaches are capable for modelling and simulating the benchmark. The results of the several simulation runs are comparable with those that have already been published.

In further steps, the described benchmark will be extended by a dynamic behaviour with specified stochastic attributes, e.g. breakdowns and changing types of products, which would allow to test new control algorithms regarding to their roubustness. The authors currently develop a new version of the C2 Benchmark and would be thankful for remarks. In addition, the separation of the modelled process and its control should be focused on.

References

[1] Flexsim Software Products: http://www.flexsim.com /products/flexsim/ [last visited: 2010-05-11]

[2] The MathWorks™: http://www.mathworks.de /products/stateflow/ [last visited: 2010-05-11]

[3] Modelica Association: http://www.modelica.org/ [last visited: 2010-05-11]

[4] Dassault Systèmes: http://www.3ds.com/products /catia/portfolio/dymola/ [last visited: 2010-05-11]

[5] ARGESIM: Benchmarks → List of Benchmarks → Flexible Assembly System, http://www.argesim.org/ [last visited: 2010-05-11]

[6] M. Lindemann, S. Schmid. *Simulationswerkzeuge in Produktion und Logistik: Marktübersicht*, PPS Management, Vol. 12 (2), 2007, pp. 48–55. (*in German*)

[7] VDI 3633-1:2000-03: *Simulation von Logistik-, Materialfluß- und Produktionssystemen: Grundlagen*, Beuth Verlag, 2003.

[8] A. Angermann: *Matlab – Simulink – Stateflow. Grundlagen, Toolboxen, Beispiele*, Oldenburg Wissenschaftsverlag GmbH, 2009. (*in German*)

[9] The Modelica Association – Chairman Martin Otter: *Modelica Standard Library 3.1_build5* (released on 2009/12/18). http://www.modelica.org/libraries /Modelica [last visited: 2010-05-11].

[10] Otter M; Arzén K.-E; Dressler, I.: StateGraph - A Modelica Library for Hierarchical State Machines, Proceedings of the 4th Int. Modelica Conference, 2005, pp. 569–578.

[11] J.A. Ferreira, J.P. Estima de Oliveira. *Modelling Hybrid Systems using Statecharts and Modelica*, Proc. 7th IEEE Int. Conference on Emerging Technologies and Factory Automation (ETFA), 1999.

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

[13] V.S. Prat, A. Urquia, S. Dormido. *ARENALib: A Modelica Library for Discrete-Event System*, Proceedings of the 5th Modelica Conference; Vienna, Austria; September 2006; pp. 539-548.

[14] EN 61131-3:2003: Programmable controllers – Part 3: programming languages (IEC 61131-3:2003). (*or derived national versions*)

[15] S. Schreiber, M. Barth, A. Fay. *Modellierungs- und Simulationsmethoden: Vergleich und Bewertung anhand des Benchmarks ARGESIM C2*. In: Proceedings of "ASIM-Treffen 2010 – STS/GMMS". Ulm, 2010, pp. 290-297. *(in German)*

Corresponding author: Sebastian Schreiber
Helmut-Schmidt-University,
Institute for Automation Technology,
Hamburg, Germany
sebastian.schreiber@hsu-hh.de

Received: March 25, 2010

Accepted: April 5, 2010