Proceedings of the 2012 Winter Simulation Conference
C. Laroque, J. Himmelspach, R. Pasupathy, O. Rose, and A.M. Uhrmacher, eds

A FREE SIMULATOR FOR MODELING PRODUCTION SYSTEMS WITH SYSML

Oliver Schönherr

Universität der Bundeswehr München
Department of Computer Science
85577 Neubiberg, GERMANY

Markus Rehm

Dresden University of Technology
Department of Mechanical Engineering
01062 Dresden, GERMANY

Jan Henrik Moss

Dresden University of Technology
Department of Computer Science
01062 Dresden, GERMANY

Oliver Rose

Universität der Bundeswehr München
Department of Computer Science
8557 Neubiberg, GERMANY

ABSTRACT

In this paper, we present an approach for a free simulator based on our modeling approach for developing a simulation tool independent description of production systems with SysML. Our aim is to develop production models by means of SysML and simulate them. For the graphical development of the SysML models we use our developed free modeling tool TOPCASED Engineer which is a modified TOPCASED environment to improve the usability of SysML especially for systems engineers. Our simulator will be based on the free simulator framework James II from the University of Rostock. The modeling concept for the simulator is based on our general model description for discrete processes in production which permits to create comprehensive production scenarios.

1 INTRODUCTION

In the context of our development of a general description of discrete processes as they can be found for instance in production and logistics systems (cf. Schönherr & Rose 2009), (cf. Schönherr & Rose 2010), (cf. Schönherr & Rose 2011), we decided to develop an open source simulator based on these concepts. To that end, we use our modeling concept based on the modeling language SysML, a simulator based on the open source simulation framework JAMES II, our self-developed SysML modeling tool for Engineers TOPCASED Engineer which is based on TOPCASED and a multilayer architecture to combine these components (see Figure 1).

There are five reasons which motivate such a development:

1. SysML allows realizing projects with the principles of systems engineering. So the modeler obtains comprehensiveness even for large projects and reduces the discrepancy between model and reality.

2. There is a vital need for a modeling concept which shows the user the complete model and provides an opportunity to divide it explicitly into different functional parts. So the modeler has a better overview for modeling and analyzing the model.

3. A non-proprietary framework offers the opportunity to simulate models for free.

978-1-4673-4781-5/12/$31.00 ©2012 IEEE

4. It means the start for an open source project to give different scientists and companies a chance to work together on a free simulation solution for discrete processes as they can be found for instance in production and logistics systems.

5. Companies are given the possibility to adapt the simulator to their specific problems.

This work presents an approach for modeling and simulation of discrete processes. We show how to develop discrete models by means of SysML and how to simulate them. In order to understand the specifics of modeling discrete systems we interviewed experts, studied current literature and conducted a market analysis of simulation modeling tools. Based on this knowledge we intended to develop a general model for discrete processes (firstly for production systems) to create comprehensive scenarios. In addition, we tested whether SysML is appropriate to support our general modeling approach. In our previous work we converted SysML models to a large variety of (commercial) simulation tools (cf. Schönherr & Rose 2009). Now, also the (direct) simulation of these SysML models is intended.

In many areas of science, like computer science or electrical engineering, modeling languages have been established. However, this is not the case in the field of discrete processes (cf. Weilkiens 2008). Also for special domains of discrete processes – like in production or logistics – there is no commonly accepted standard. Usually discrete scenarios are developed with simulation tools. These tools are not standardized, they are proprietary and there is a large variety of tools available on the market (cf. Noche & Wenzel 2000). For modeling discrete processes there are many approaches and methods like Petri Nets, project networks, or different simulation tools, but none of them has been established as a standard. This could be due to the lack of an adequate modeling language. Our criteria for such an adequate modeling language are that it is standardized, powerful, non-proprietary, useful to model very complex scenarios (including a graphical editor) and general but practically useful for domain specific issues. We think SysML fulfills all of these criteria.

There are also many proprietary approaches to model discrete processes for simulation. Especially in the field of production and logistics, e.g. Plant Simulation, FlexSim or Simcron Modeler, Enterprise Dynamics or AutoMod are very popular. Other products such as Arena and AnyLogic try do cover a variety of domain-specific problem areas for discrete simulation (cf. Noche & Wenzel 2000). All of them are proprietary and base the modeling concept on the perspective of a particular tool developer.

2 MULTILAYAER ARCHITECTURE FOR THE SIMULATOR

In previous work we defined a multilayer architecture to translate a simulation model created in SysML into a vendor specific proprietary simulation model for a specific simulation tool. Figure 1 shows how our simulator is integrated into this architecture. To build an effective tool we use a multilayer architecture (see Figure 1). At first we build the model with the SysML modeling tool TOPCASED Engineer. The modeling tool should provide a suitable data interchange format (XMI), contain all SysML elements (identified as mandatory for simulation modeling) and has to be appropriate for building large models.

In the first step we use a program called “parser”. The parser reads the SysML model, which is specified in the exchange format (XMI), filters all non-relevant information, and writes the remaining significant parts into the internal model.

In a second step the internal model is directly executed in our simulator which is based on the simulator Framework JAMES II from the University of Rostock. The simulator uses the internal model as input. This means that no further translation into another format is needed. During a simulation run a lot of data is generated. The data is directly written into the internal model which contains all relevant structures to store this data. To give the user a visual feedback the parser transforms the internal model back into the format used by the modeling tool.

The transformation process is currently performed in an external program. We have to save the model in the modeling tool, open the translator, parse the model and execute the simulation. To simplify this pro-

cedure we intend to integrate it as a new plug-in into the TOPCASED Engineer modeling tool we deverloped.

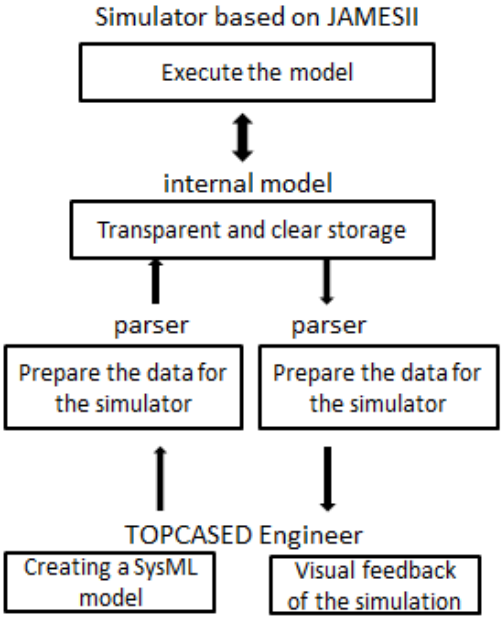

Figure 1: Multilayer architecture for the system.

We use TOPCASED Engineer to model the scenarios for the simulation and to present the results. It is open source, based on TOPCASED and we adapted the usability specifically for systems engineers (see Section 7). As modeling language we use SysML which is a standardized graphical modeling language for supporting specification, analysis, design, verification and validation of systems (see Section 4). The whole modeling is based on our special modeling concept (see Section 5), so we give TOPCASED Engineer a special profile which is based on this concept. The simulation part is executed by a self-developed simulator (see Section 6) which is based on the simulation framework James II (see Section 3). (see Figure 2)

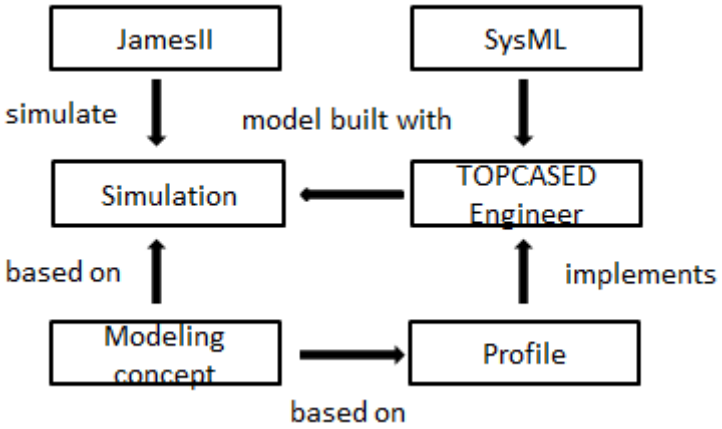

Figure 2: Components of the system.

3 JAMES II

JamesII is a plug-in based framework for modeling and simulation (Himmelspach & Uhrmacher 2007). It consists of a stable core which did not change a lot recently. The main parts of the framework according to Himmelspach and Uhrmacher (2007) are:

- User Interface: Although JamesII has a built-in user interface for modeling, experiment setup and the analyses of results and visualization of simulation runs. We do not make use of the JamesII user interface. We use a SysML model as input which contains already all information to run the simulation and present the results. Modeling and analysis is done in a separate SysML modeling tool (see Section 2).//

- Experiment: Experiments are the central elements during simulation. They consist of several simulation runs. JamesII provides flexible methods to design experiments. With the help of experiments you can change parameters between consecutive simulation runs .//

- Database: During simulation runs a lot of observed data is generated. The database components provides an interface but makes no assumptions how the data will be stored.//

- Model: The basic interface and class for models. All new modeling formalisms or concepts which shall be simulated through JamesII have to inherit from these base classes.//

- Simulator: A simulator is used to simulate model. The core of JamesII does not contain any simulation algorithms but provides all capabilities to add new ones.//

- Simulation: Simulation is the execution of a model with a specific simulator. A simulation can be distributed or not. If the simulation is distributed it provides mechanisms to describe partitioning and load balancing.//

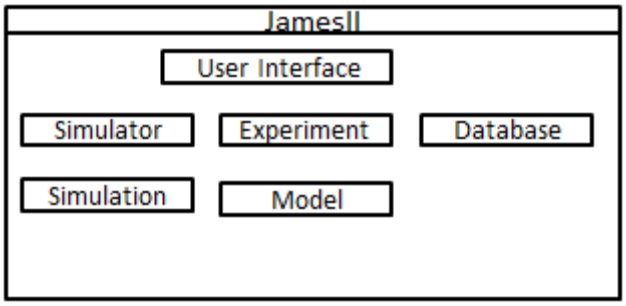

Figure 3: James II.

In addition to the functional components JamesII has a registry in which all available plug-in are loaded dynamically at startup. The registry provides methods to list, get and add plugins during the runtime. For adding functionality JamesII provides a plug in concept. Every plug-in belongs to a certain plug-in type. The declaration of new plugins is done in XML files. The implementation is done in Java. Another important concept is the separation of the model from the simulation algorithm. The simulation algorithm can access the model but the model has no access to the simulation algorithm. This allows exchange and comparability of different simulation algorithms for the same model. To add the capability to simulate SysML models which are conforming to our general model description for discrete processes in production inside the JamesII framework we have to execute two steps.

1. Adapt the internal model to the model interface of JamesII and enhance the model with semantic details to make it executable.

2. Create the simulation algorithm. All simulation algorithms have to implement the IProcessor interface. The most important method to implement is the nextStep() method. It is called every time the next step of the simulation should take place. Every simulation algorithm has to manage the progress of time on its own. Therefore the getTime() method has to be overwritten to get the current simulation time in return.

Although JamesII can be run as a stand-alone application we intend to use the framework as part of the Sys4Sim application. Another alternative is to integrate it directly into the SysML modeling tool. We use TOPCASED as our main tool for developing Simulation models. As TOPCASED is based on the Eclipse framework JamesII capability can easily be integrated as a new plugin.

4 MODELING WITH SYSML

In July 2006 the OMG and the International Council on Systems Engineering (INCOSE) published the Systems Modeling Language (SysML) 1.0, a standard that is based on the Unified Modeling Language (UML) 2 and which is especially adapted to the needs of systems engineers. The goal of SysML is to become the standardized modeling language in the field of Systems Engineering. The goal of SysML is defined by the OMG as: "standard modeling language for systems engineering to analyze, specify, design and verify complex systems, intended to enhance systems engineering information amongst tools, and help bridge the semantic gap between systems, software and other engineering disciplines" (cf. OMG 2010).

SysML is a graphical modeling language for supporting specification, analysis, design, verification and validation of systems. In June 2010 the OMG released SysML 1.2. There have been many controversies about SysML during the short period of time since its publication. SysML is spreading very fast. Today many of the most prominent developers of modeling tools like ARTiSAN, Telelogic, I-Logix and Sparx Systems make use of SysML. In systems engineering a great amount of projects like the engineering of planes, helicopters or ships are performed by using SysML. Pörnbacher describes how to model a software control of an automated manufacturing systems with SysML and Modelica (cf. Pörnbacher 2010). Rosenberg and Mancerella develop embedded systems with SysML (cf. Rosenberg & Mancerella 2010). We try to model a large variety of discrete problems from production, logistics or civil engineering with SysML.

In accordance with UML, SysML distinguishes between model and diagram and divides the model into a structural and a behavior part. The model contains the complete description of the system and consists of diagrams that visualize particular aspects. The structural part describes the static structure, like the elements and their relationships, in a system. In the behavior part SysML describes the dynamic behavior at and between its elements (see Figure 4). SysML has been extended by some concepts, however, in contrast to UML also important concepts such as time and the interaction diagram have been removed. To model the behavior and the structure of a system four diagrams are used, which we consider in detail. SysML uses also a request and an assertion concept, which are extensions to basic UML (see Figure 4).

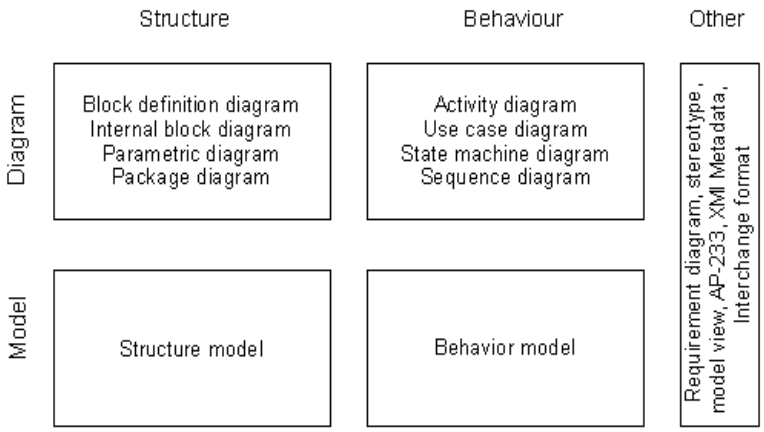

Figure 4. SysML structure.

5 MODELING CONCEPT

Figure 5 shows the common standard modeling concepts compared to our approach. A vast majority of the commercial simulators does not distinguish between the different parts of the model – there is almost no functional separation. Often some parts of the model are even not accessible to the user (e.g. the Communication Model). In addition, in many simulation tools there is no separation between model and simulator (see Figure 5).

Our modeling concept differs clearly between the model and the simulator which works on it. The model is split into the System Model which represents the modeled system and into the Experimental and Analysis Model where the user determines how the simulator has to work on the single part of the System Model. Some parts of the system model are modifiable by the engineer and some parts are only informative (restricted).

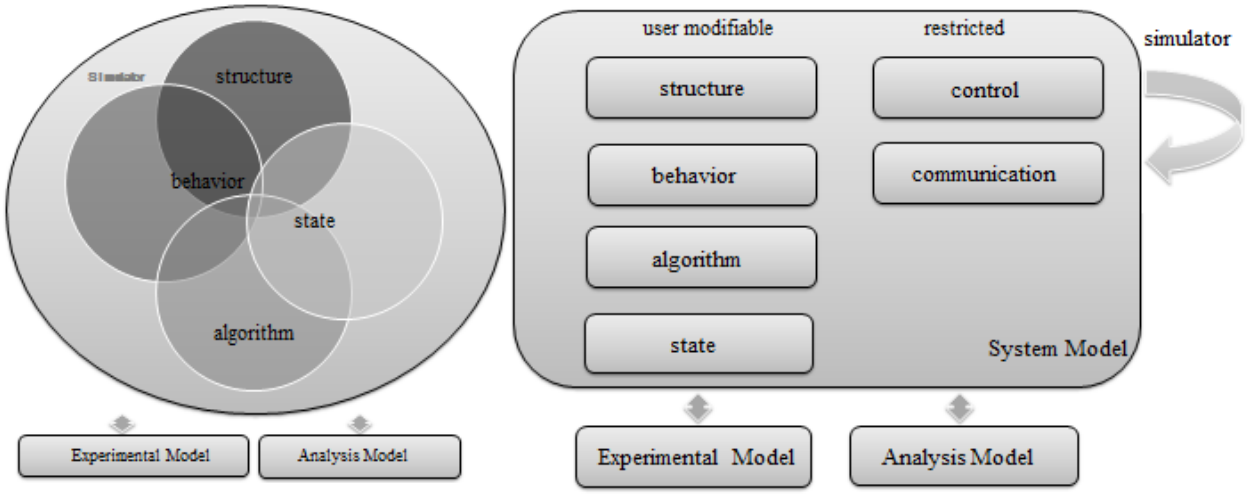

Figure 5: Modeling concept.

The structural part describes the static structure, such as elements and their relationships, in a system. Whereas in other areas workflows are determined by information flows, in production the entity flow controls the behavior of the model: The entity is the central element as it represents the job / piece which moves through the system and is processed by the elements of the machinery. All events in a model, ex-

cept for interruptions, are triggered by the entity. The entities enter the system through the arrival process and leave it through the departure process. While they move on specific routes, different processes execute actions on them; these may or may not require resources. Along their way, the entities can be stored in queues. SysML provides four diagrams for describing the structure of a model. We use the block definition diagram (cf. Schönherr & Rose 2011).

The behavioral part describes the dynamic behavior between the model elements, for example the movement of an entity through the production facility. We can describe the behavior in different levels of detail. In order to model the behavior of dynamical resources (agents) we can use state machines. For behavior related to an order or schedule we use activity diagrams (cf. Schönherr & Rose 2011).

The algorithmic decisions can include the complete system state. All elements have an interface to the monitor element, from which they gain information and therefore know the complete state of the system. If a global decision in the queue, the router, the batch-machine or the resource pool needs to be made, the controller obtains the information from the monitor, executes the algorithm and returns the final decision to the corresponding element. The controller can execute every algorithm which we define in the Algorithm Model. So the Control Model describes the Interfaces between the monitor and controller and the other elements. And it describes the behavior of the monitor and controller. This information is only given as information for the model developer. To describe the Control Model we use the internal block diagram and the behavior diagram.

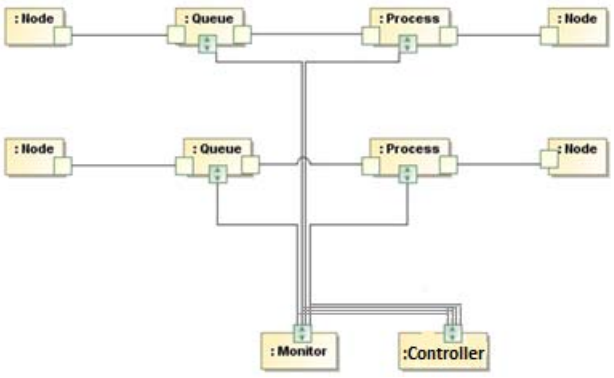

Figure 6: Control Model.

The Algorithm Model provides many algorithms to the user. These algorithms are modeled as stereotypic blocks in a separate block definition diagram. Every algorithm owns different start values which can be defined by the user, e.g. which priority rule is necessary to construct a CONWIP control system. Furthermore, the user defines elements which are run by the algorithm (by setting part properties). Finally, the algorithms are executed automatically via the selected elements by the controller.

The state model is another model which describes the behavior of the elements to be simulated. Every element of the metamodel has one state model. This model is usually the same for every instance of the metamodel type, e.g. all machines have the same state model as they are applied on the metamodel level. A further extension can be that the state model can be altered on model level which results in different behavior for the same metamodel without changing the metamodel itself. For describing the state model the state machine diagram will be used. A state machine consists of a set of states and the relation between states. The relation is described as transitions between a source and a target state. As a first approach we use the standard SEMI E10 from the semiconductor industry to describe our set of states.

The Communication Model is only informative for the engineer. We describe how the elements communicate by using sequence diagrams.

The Experimental Model is not a part of the System Model. There the user defines the experiment, e.g. the simulation time, the length of the warm-up phase or the number of replications. We model the Experimental Model with SysML block definition diagrams.

The Analysis Model is also not a part of the System Model. There the user defines which factors need to be analyzed. As a first approach we use the standard SEMI E124 from the semiconductor industry to describe our set of factors.

6 HOW THE SIMULATOR WORKS ON OUR MODEL

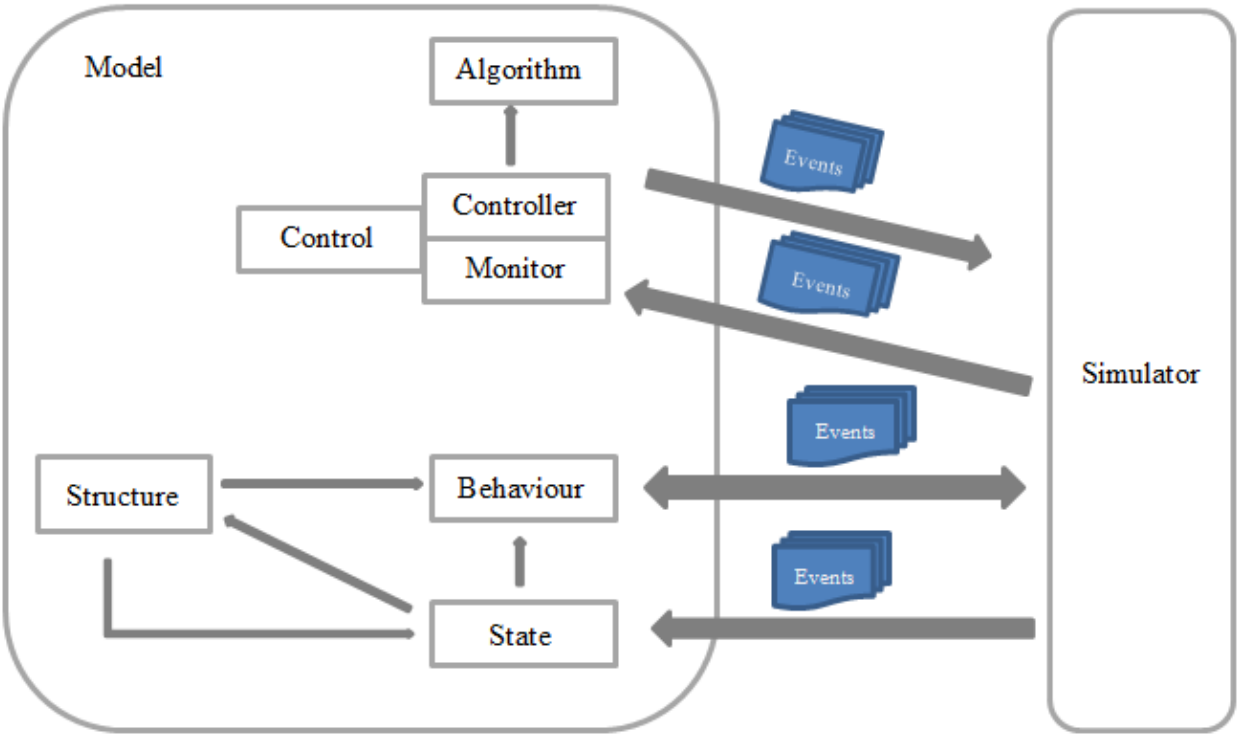

Figure 7: Shows the components of the simulation model and their relationships.

The simulation starts with setting up the experiment defined in the Experimental Model. The Experimental Model itself is divided into a mandatory structural and an optional behavioral part. The definition of a simple experiment does not require any behavior. More complicated experiment setups like parameter variation and optimization algorithm require a more complex definition within a behavioral part. According to the experiment definition, a set of simulations are created.

The Analysis Model describes which attributes of the model elements we want to observe and which and how the observations are transformed into meaningful statistical measures, i.e. if we want to know the average utilization of a queue we have to observe the current capacity and compute the average after the simulation. According to the Analysis Model we can instrument the model as well as the simulation and attach observers to all entities which we like to observe during simulation. The simulator itself is based on a discrete event simulation algorithm. Figure 8 shows a very basic approach to discrete event simulation.

In the first step the model is initialized with the parameters defined in the experimental setup. The simulation clock is set to start and the first events are added to the event list. Then we select the event or events with the minimal time progress. After that we advance the clock to the new time according to the current time and the time progress. The events will be sent and routed to all elements which can manage the events. The activation of the behavior, state or control will possibly generate new events. The events

are added to the event list. If the queue contains additional elements the algorithm repeats with getting the events with the minimal time progress. If there are no additional events or the user pauses or stops the simulation the simulation run ends.
Events are used to induce a communication between the various elements of the model and to provide a synchronization of time between the model elements. In addition to events, model elements can communicate through a request response messaging mechanism, e.g. a process can ask for resources.

The control part of the model is used to control complex decision which occur during the simulation run. It is divided into a controller and a monitor. The monitor is used to provide a global view on the whole system. The monitor can trace events coming from other elements and specify triggers. If the condition of the trigger holds the monitor invokes the controller to make a decision, i.e. rescheduling of resources between several processes if some resources are not available anymore. The controller uses the Algorithm Model to make decisions. The controller itself uses events to communicate with the simulator and its environment.

Every element of the metamodel has one state model. This model is usually the same for every instance of the metamodel type, e.g. all machines have the same state model since it is applied on the metamodel level. It is described by a state machine diagram. The state model is initialized at the initialization of the specific element. The state changes as a reaction to received events. Furthermore, a behavior can be activated during the transition from one element to another or when entering or leaving a state. Another behavior may be executed when the entering behavior is completed or the state changes.

The behavior part of the system is modeled through activity diagrams. The simulation of activities is based on a token flow. Tokens can travel over arcs from one node to another. Nodes are either actions or special control nodes. The simulation of the token flow is performed in four steps (similar to Petri nets):

1. Determine the nodes which can be executed;

   A node can be executed if all conditions of the input arcs are satisfied, i.e. all incoming arcs of that node have a token.

2. Choose node;

   If more than one node can be executed choose the next node to be executed.

3. Consume tokens;

   Some or all tokens from the input arcs are removed and one token is placed on the node to execute.

4. Forward tokens;

   When the execution of the node has terminated the token is removed from the node and tokens are offered to some or all of the outgoing arcs.

After the simulation is done all statistical measures will be calculated from the observed data. In case the measurement is just a single value we can directly write the value into the Analysis Model. Furthermore, complex experiment setups can use the results as feedback for further simulation runs.

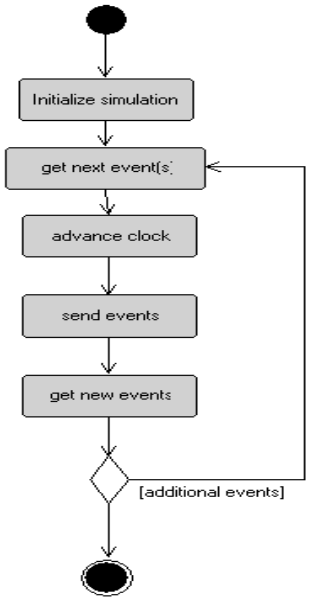

Figure 8: Basic discrete event simulation.

7 TOPCASD ENGINEER

TOPCASED is an open source modeling tool mainly for the development of critical embedded systems. For TOPCASED there are many extensions, including TOPCASED-SysML. TOPCASED-SysML supports SysML 1.1 and the models are stored in XMI 2.1 format. The editor can be used in the TOPCASED development environment but also be integrated into the Eclipse development environment. TOPCASED is an open source program. It can be used freely and can be arbitrarily changed and further developed.

We chose to develop our own modeling tool concept which is based on the open source project TOPCASED, a modeling tool, which receives large industrial support, e.g. Airbus, Continental or EADS. During the development of our tool, we adapted the usability specifically for systems engineers. The default setup is for the domain of production systems, but it is easily adaptable to other domains. Currently we evaluate some extensions we have implemented, which are necessary for system engineers to use SysML comfortably.

We analyzed the usability of SysML with some commercial modeling tools. In comparison to other modeling languages, e.g. MoogoNG or Jane, SysML provides the capability to model the complete scenario without restrictions or problems. Trained users can build complex models comfortably and rapidly. But only skilled and trained users can use SysML adequately because modeling with SysML is very abstract. Although there are good SysML modeling tools, they are not intuitively usable for discrete models. One reason for this is that these modeling tools were created for computer scientists and not for systems engineers. Usually systems engineers are not familiar with SysML. Modeling with SysML requires additional skills due to the following reasons.

- SysML is very powerful and has a huge amount of modeling elements (see figure 9).
- SysML is very abstract (which makes it powerful).

Therefore, we decided to develop our own modeling tool which is customized to the requirements of systems engineers of production systems. SysML provides much more modeling possibilities than a production engineer would ever need. This is why a customized tool has to support the modeler during the selection process.

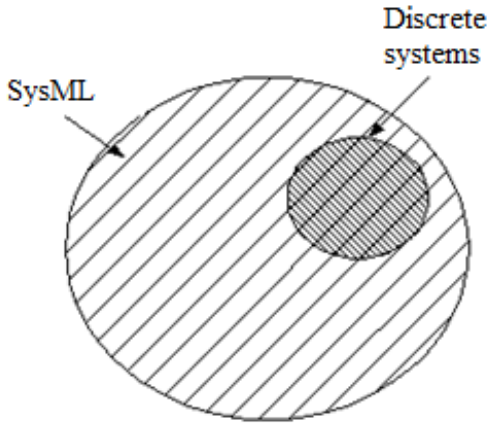

Figure 9: Complexity of SysML.

The basic element of SysML is the block. A block is able to represent everything, for example a machine, a worker or a transporter. This is a very abstract way of thinking which can be confusing. Even engineers whose tools generally have a fixed amount of defined elements can be confused. In SysML one can work with stereotypes which describe a defined element with attributes. So a suitable modeling tool is able to give a fixed and not too large number of modeling elements. To cover the modeling of all possible scenarios in the production, a significant number of stereotypes and stereotype attributes is necessary. In order to keep the overview, the engineer has to obtain only a preselection of stereotypes and stereotype attributes. To make this possible, the modeling tool needs to provide integrated, dynamically definable stereotypes. We use TOPCASED Engineer to model the scenarios for the simulation and to present the results.

8 CONCLUSION

An approach for a free simulator based on our modeling approach for developing a simulation tool independent description of production systems with SysML is presented. Our aim is to develop an open source simulator which can simulate all kinds of discrete processes as they can be found for instance in production and logistics systems. For these we use our modeling concept based on the modeling language SysML, a simulator based on the open source simulation framework JAMES II, our self-developed SysML modeling tool for engineers TOPCASED Engineer which is based on TOPCASED and a multilayer architecture to combine the components.

The general concept consisting of SysML, our modeling approach, TOPCASED Engineer, JAMES II and the multilayer architecture (including the internal model and the parser) is finished so far. Recently we work on the simulator itself and we are already able to simulate small models.

We hope that further users join our work on an open source simulation solution. We see advantages to have the possibilities to realize projects with the principles of systems engineering, to model scenarios with a modeling concept which shows the user the complete model and divide it explicitly in different parts and to simulate completely for free.

REFERENCES

Himmelspach, J., and A. M. Uhrmacher. 2007. “Plug'n Simulate.” In Proceedings of the 40th Annual Simulation Symposium. 137-143. Washington, DC: IEEE Computer Society.

Noche, B., & Wenzel, S. 2000. “The new simulation in production and logistics. Prospects, views and attitudes. Paper presented” In *9. ASIM-Fachtagung Simulation in Produktion und Logistik*, Berlin, Germany.

OMG. 2010. "SysML Modelling Language explaines." Accessed March 20, 2012, http://www.omgsysml.org/SysML_Modelling_Language_explained-finance.pdf.

Pörnbacher, C. 2010. „Modellgetriebene Entwicklung der Steuerungssoftware automatisierter Fertigungssysteme." Munich, Germany: Herbert Utz Verlag.

Rosenberg, D., & Mancarella, S. 2009. "Embedded Systems Development using SysML." Sparx Systtems. Accessed March 20, 2012, http://www.sparxsystems.com/downloads/ebooks/Embedded_Systems_Development_using_SysML.pdf.

Schönherr, O.,& Rose, O. 2009. "A General SysML Model for Discrete Processes in Production Systems." In *Proceedings of the 2009 INFORMS Simulation Society Research Workshop*. Coventry, U.K.

Schönherr, O.,& Rose, O. 2010. "Important Components for Modeling Production Systems with SysML." In *Proceedings of the 2010 IIE Annual Conference and Expo*. Cancun, Mexico.

Schönherr, O.,& Rose, O. 2011. "A General Model Description for Discrete Processes." In *Proceedings of the 2011 Winter Simulation Conference*. Phoenix, Arizona. Accessed March 20, 2012, from http://www.informs-sim.org/wsc11papers/198.pdf.

Weilkiens, T. 2008. "Systems Engineering with SysML/UML." Heidelberg, Germany: Dpunkt Verlag.

AUTHOR BIOGRAPHIES

OLIVER SCHÖNHERR is a PhD student at the Universität der Bundeswehr München, Germany. He is a member of the scientific staff at the Chair for Modeling and Simulation. He received his M.S. degree in computer science from the Dresden University of Technology, Germany. His e-mail address is oliver.schoenherr@unibw.de

MARKUS REHM is a PhD student at the Dresden University of Technology. He is a member of the scientific staff at the Chair of Logistics Engineering. He received his M.S. degree in Industrial Engineering & Management from the Dresden University of Technology, Germany. His e-mail address is markus.rehm@tu-dresden.de

JAN HENRIK MOSS is a student at the Dresden University of Technology. He is a student of the scientific staff at the Chair for Modeling and Simulation. His e-mail address is Jan-Henrik.Moss@mailbox.tu-dresden.de

OLIVER ROSE holds the Chair for Modeling and Simulation at the Institute of Applied Computer Science of the Universität der Bundeswehr, Germany. He received an M.S. degree in applied mathematics and a Ph.D. degree in computer science from Würzburg University, Germany. His research focuses on the operational modeling, analysis and material flow control of complex manufacturing facilities, in particular, semiconductor factories. He is a member of IEEE, INFORMS Simulation Society, ASIM, and GI. Web address: www.simulation-dresden.com.