Proceedings of the 2011 Winter Simulation Conference
S. Jain, R.R. Creasey, J. Himmelspach, K.P. White, and M. Fu, eds.

A GENERAL MODEL DESCRIPTION FOR DISCRETE PROCESSES

Oliver Schönherr
Oliver Rose

Dresden University of Technology
Institute of Applied Computer Science
Dresden, 01062, GERMANY

ABSTRACT

In this paper, we present an approach for developing a simulation-tool-independent description for discrete processes and for converting such a general model into simulation-tool-specific models. Our aim is to develop models by means of SysML and to build converters from SysML models to models of a large variety of simulation tools. We developed Translator-Plugins for Anylogic, Simcron, Factory Explorer and Flexsim. Based on this architecture, we develop a general model description for discrete processes which permits to create comprehensive scenarios. Modeling can be divided into a structural, a behavioral and a control part. Our main domain is production systems but we show which elements are not domain specific and can be generalized to an approach for a standard to model discrete production planning and control problems. We also test domains like hospitals, logistics and civil engineering.

1 INTRODUCTION

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 (Weilkiens 2006). There are two reasons which motivate such a development:

- Modeling languages allow realizing projects by the principles of systems engineering. So one obtains clearness even for large projects and reduces the discrepancy between model and reality.
- Modeling languages are a central part of automatic code generation.

In this paper, we present an approach for developing a simulation-tool-independent description of production systems and how to convert such a general model into simulation-tool-specific models. In the field of software engineering automatic code generation of UML-Models by CASE-tools is very common and standardized (Fowler 2003). For modeling discrete processes there are many approaches called "*Model Based Software Engineering*" (MBSE) like *Stateflow Coder*, ASCET, or ADAGE, but none of them has been established as a standard. This could be due to the lack of an adequately powerful, non-proprietary or general modeling language. However, in particular for modeling discrete processes in production automatic code generation is useful, because there are many different tools such as simulators, optimizers or schedulers which cannot exchange their non-standardized models so far.

The Object Management Group (OMG) developed the Systems Modeling Language (SysML) to facilitate modeling of complex systems. SysML is a standard based on the standardized general-purpose *Unified Modeling Language* (UML). There have been many disputes 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 their tools.

This paper presents an approach for automatic model generation of discrete processes in organizations such as for hospitals, logistics or civil engineering. Our domain, however, is production systems. Our aim is to develop models by means of SysML and to build converts from SysML models to a large variety of simulation tools. At first we consider whether SysML is suitable for modeling discrete processes in production. In order to understand the requirements of modeling production systems we interviewed experts, studied present literature and conducted a market analysis of simulation modeling tools. Based on this knowledge we intend to create a general meta model for discrete processes in production which permits to create comprehensive production scenarios. In addition, we tested whether SysML is appropriate to build our general model. After presenting theoretical concepts for building production models with SysML, we developed a practical approach for automated model generation for simulators based on SysML models.

2 A PRACTICAL APPROACH FOR AUTOMATIC MODEL GENERATION

We developed a software tool that automatically generates models for commercial simulation packages from given SysML models. To build an effective tool we use a multilayer architecture (Figure 1). At first we build the model with a SysML modeling tool. The modeling tool should provide a suitable data interchange format (XMI), contain all required SysML elements and has to be appropriate for building large models.

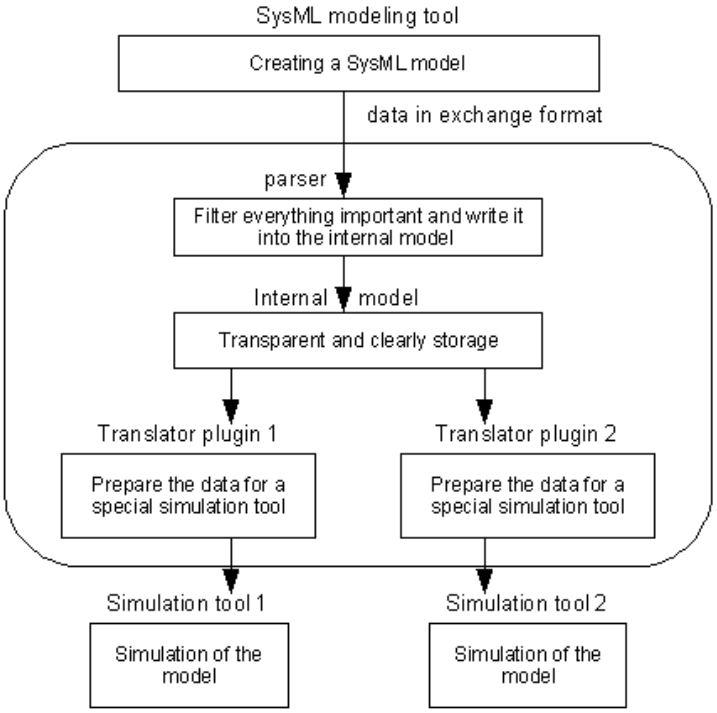

Figure 1: System architecture

If the SysML model is available in a suitable exchange format, it can be transformed into an equivalent model for a given simulation modeling tool. Since it should be possible to transform a SysML model into models of different simulation tools, a separate output must be generated for each package. Each simulation needs a suitable dedicated model in a specific input format. To simplify the software architecture, the model generation is divided into two steps, which involve an additional "internal model" (Figure 1). In the first step we use a parser that reads the SysML model file specified in the exchange format (XMI), filters out all non-relevant information, and writes the remaining significant parts into the internal model. In the second step, the translator plug-in prepares the data from the internal model for a specific simulation tool. More precisely, it takes all the relevant data and translates them into the input data format

of the simulation package, which is defined by a set of rules. Since each simulator has its own format, there has to be a separate translator plug-in for each simulation tool. So far, we developed translator plug-ins for Anylogic, Simcron Modeler, Factory Explorer, and Flexsim.

The advantage of the proposed architecture compared to the application of a single step conversion from a SysML file into a model for a simulation program is that the first step (the parser) only needs to be executed once. However, the architecture assigns a special role to the internal model because it must be particularly suited to derive models for simulation tools. The internal model has to contain all information for the generation of production system models but has still to remain transparent. It is also possible to convert models from a simulation tool into SysML. The architecture in the reverse direction is very close to the one described above. Actually, we have already developed a translation from Flexsim models into SysML models.

3 A GENERAL MODELING APPROACH

In the following we present an approach for modeling discrete systems. The approach is based on the traditional way to model production systems (Schönherr and Rose 2009). The model can be divided into a structural, a behavioral and a control part. The structural part describes the static structure of a system, like the components and their relationships. The behavioral part describes the dynamic behavior of its components, for example the movement of an entity through the production facility. The control model describes the dispatching, scheduling and routing policies. While the structural and behavior model already belong to the concept of SysML (Figure 2), the control model is a conceptual extension.

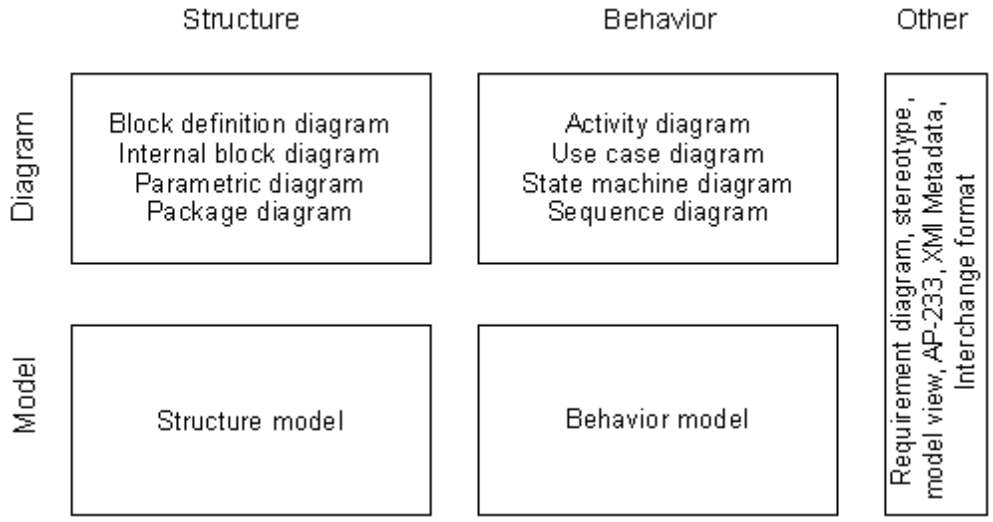

Figure 2: SysML structure

3.1 The Structural Model

In the structural model part, there are real objects, imaginary objects and auxiliary objects (Figure 3). The imaginary objects like Arrival or Departure Process, Queue and Process are necessary for the simulation. The flow object is the central element. In the domain of production systems it represents the job or piece which moves through the facility and which is processed by the elements of the tool set. All events in a model, except interruptions, are triggered by the flow object. The flow object enters the system through the arrival process and leaves it through the departure process. While it travels on specified routes, differ-

ent processes execute actions on them, for which the processes may use resources (but they are not required to). Along their way the flow objects can be stored in queues or buffers.

While every discrete system has the same imaginary objects, the domain is specified in more detail by real and auxiliary objects. Real objects are flow objects which move through the system (entity, patient, etc.) and different kinds of resources (worker, room, machine, etc.). Auxiliary objects are not necessary but they simplify a model. Relationships between the objects can be simply reservations of objects but they can also have very complex attributes or terms. An example is a process which needs a resource only half of its time or two processes which need the same resource in sequence.

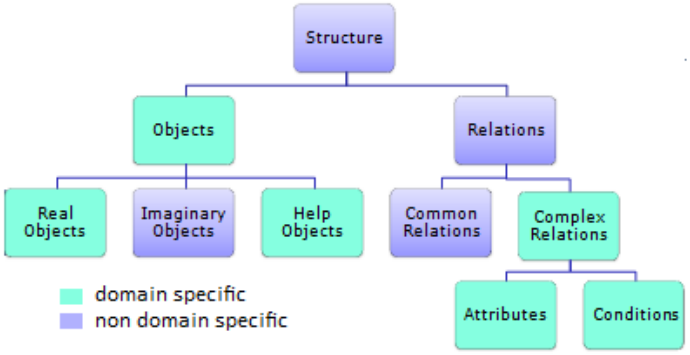

Figure 3: Structural modeling

SysML provides four diagram types for describing the structure of a model (Figure 2). We use the block definition diagram and the internal block diagram. Figure 4 shows the meta model for production systems including the domain specific elements.

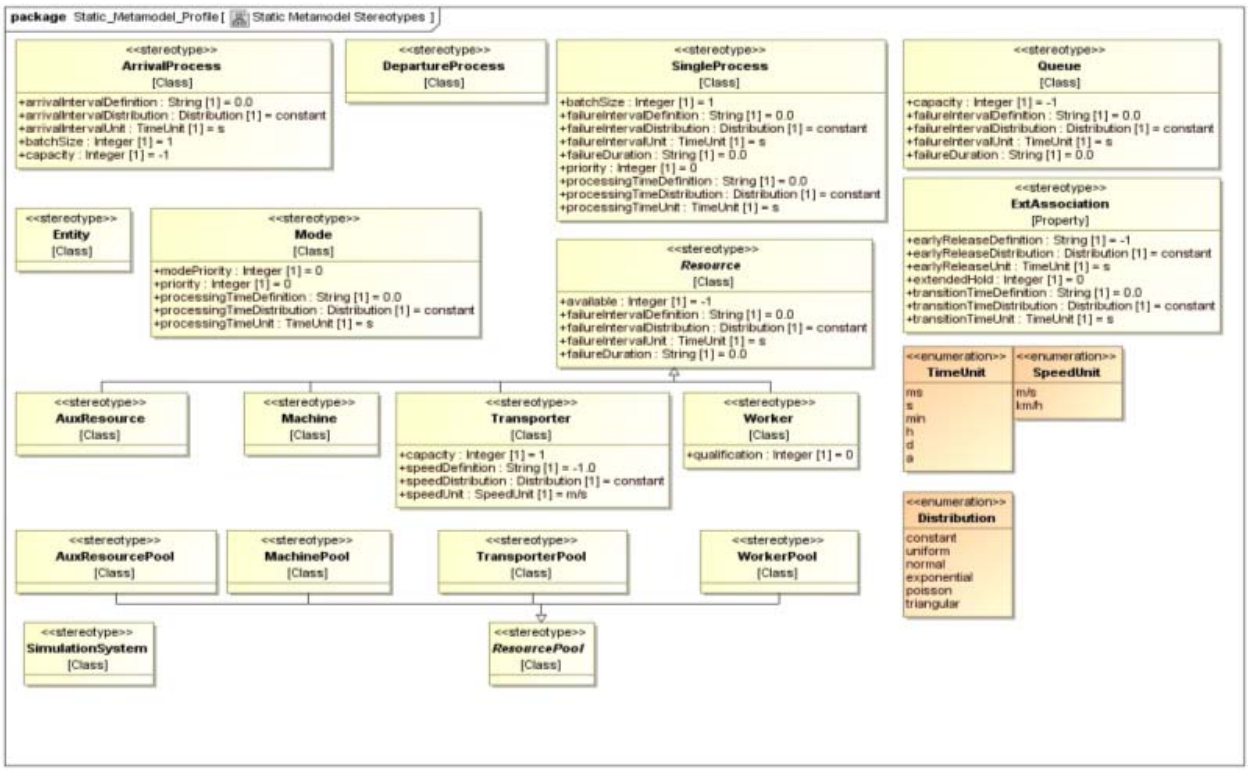

Figure 4: Structural meta model as block definition diagram

3.2 The Behavioral Model

We can describe the behavior in different levels of detail. If we have an open shop problem without product routes, we use just the SysML block definition diagram. To model the behavior of dynamical resources (or agents) we use state machines. For orders with a route or recipe we use activity diagrams.

The behavioral model can be classified into different levels of granularity. Pieper, Röttgers, and Gruhn (2006, p. 46 ff.) describe how to split the behavior of workflows into different levels of behavior. Störrle (2005, p. 194) suggest that activity diagrams can describe different levels of granularity.

Next, we present an example of a small production scenario and describe its behavior in different levels of granularity. The first level, called "process level", shows the sequence of the process steps. The example in Figure 6 shows an entity which arrives in the system through the arrival process, then it will be processed, it cools down, and leaves the system through the departure process. For most simulation tools this level of detail is enough (Anylogic, Flexsim, Simcron, etc.) but some simulation tools need a more detailed description.

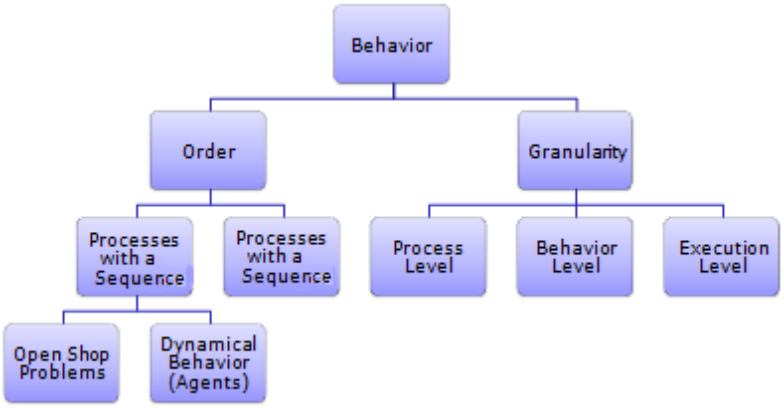

Figure 5: Structure of behavioral modeling

Every process step on the process level is an imaginary object (Arrival or Departure Process, Queue or Process). In the next level of detail, called "behavior level", we define a set of behavioral patterns for every process step of the process level. In the first step the example process acquires the needed resources. In the next step the entity will be processed. At the end the process frees the resources (Figure 7).

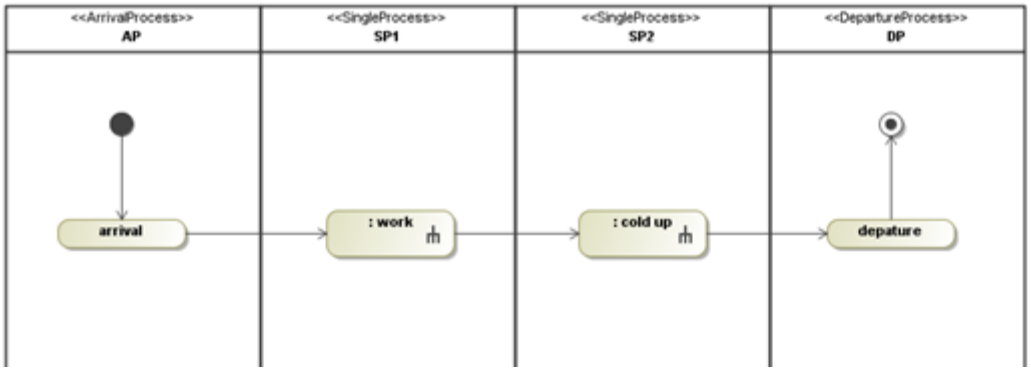

Figure 6: Example on process level

The detailed execution of the behavior patterns will be described on the "execution level". For this very detailed description the OMG has defined about 40 actions which serve as a basis for a detailed description of behavior (Pieper, Röttgers, and Gruhn 2006, p. 47; Weilkiens and Oestereich 2006, p. 161).

In our example we model the acquisition of resources. At first we compare the number of needed workers with the number of existing workers. Then the workers will be reserved (Figure 8).

Figure 7: Example on behavior level

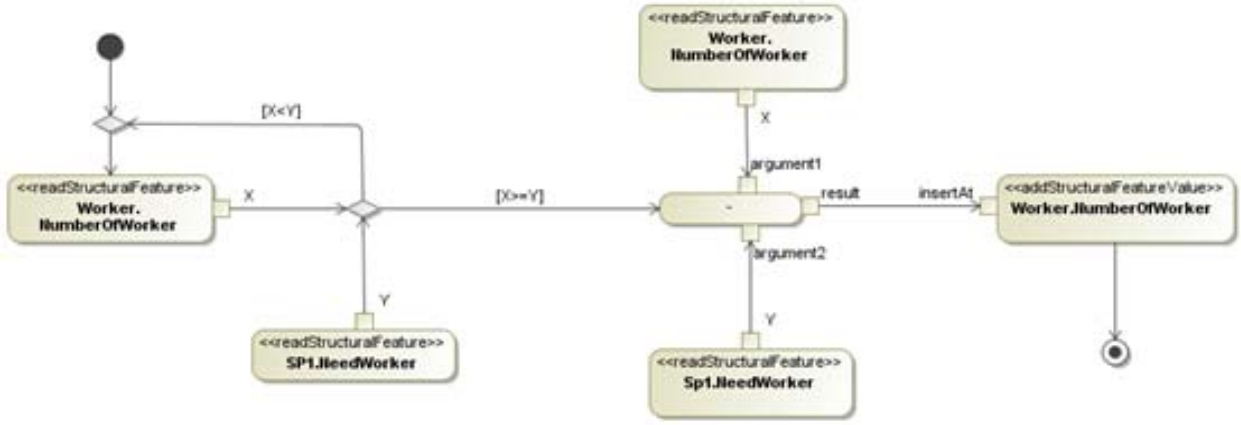

Figure 8: Example on execution level

3.3 The Control Model

We were not able to find a general approach for modeling the control of a production system in the literature. As a first step, we tried to classify the different controls by means of their complexity. Afterwards we analyzed the interfaces and elements of the control part of the complete model and which information flows are necessary and important. To this end we analyzed the control approaches of seven simulation tools. Furthermore, we classified different types of control. Figure 9 gives an overview of our findings, additional details can be found in Rehm (2010). The control model consists also of a structural and behavioural part. As a consequence, we can describe it with SysML.

Figure 10 shows the control model and its functions. The local decisions describe algorithms which do not need information from other elements. These elements only need the attribute algorithm and attributes for the local control. They are executed statically just by means of their values. The algorithmic decisions may include the complete system state. All elements have an interface to the element monitor, from which they gain information and therefore know the complete state of the system. If a global decision in a queue, a router, a batch machine or a 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.

Furthermore, we work on a description of algorithmic models to map a high variety of control algorithms to SysML. To achieve this goal we look at fundamental elements of different algorithms in order to use them like a construction kit to build other algorithms from these modules. We intend to use the modules as stereotyped actions in SysML activity diagrams (Figure 11).

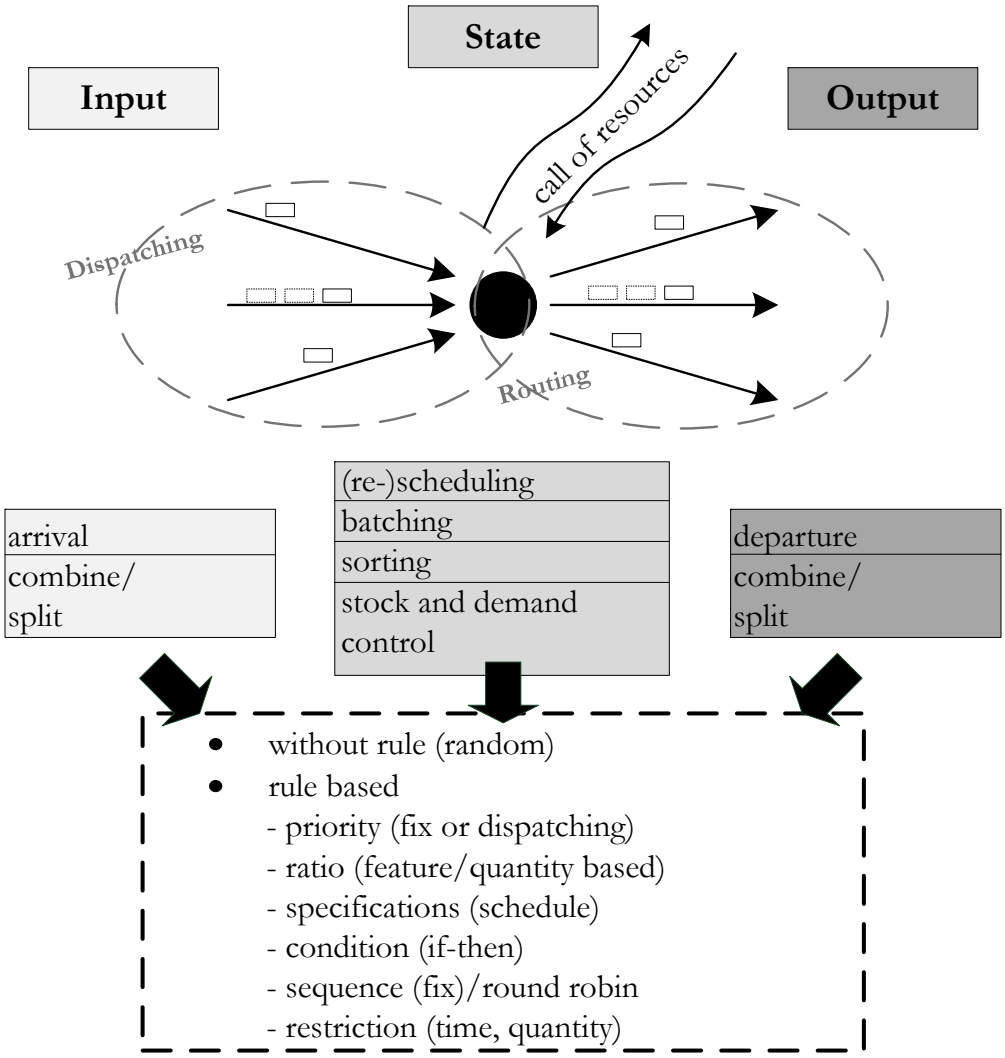

Figure 9: Structure of control

Control Function	Elements	Effect
Scheduling [(Re-) Scheduling, Sorting]	Queue	local/global
Routing [Arrival, Leaving]	Router	local/global
Dispatching [Pooling, Allocation]	Batch Machine	local/global
Management of Resources	Resource Pool	local/global
Global decisions	Controller	global
Loging Unit	Monitor	global

Figure 10: Elements of the control model

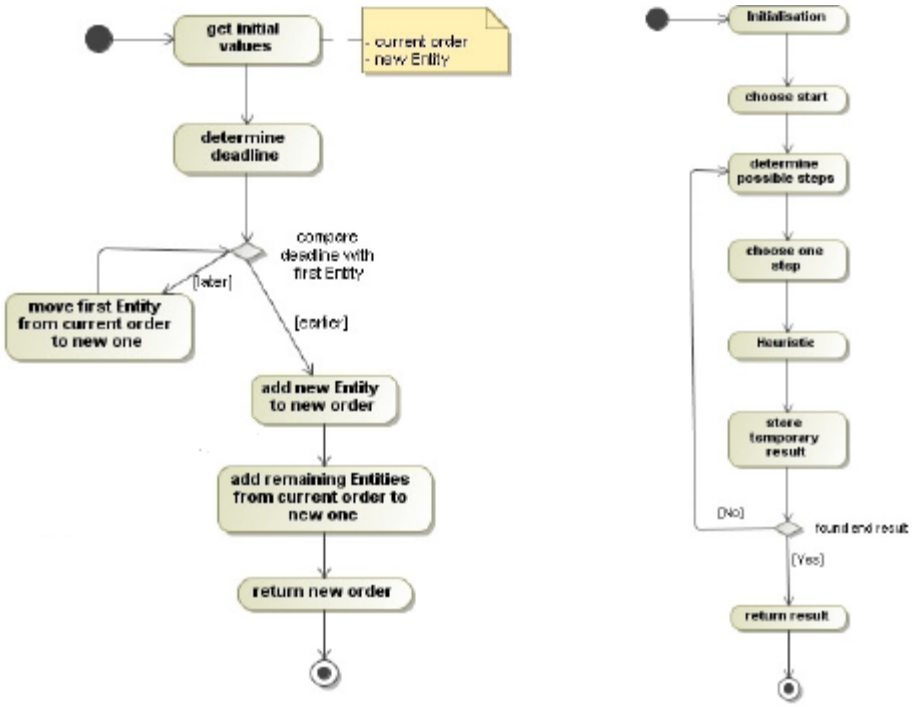

Figure 11: A first approach to describe control algorithms

As a first step we determined algorithms that we want to consider by performing a broad literature search in field of production control. The literature divides production control into the following task areas: order generation, order release, sequence planning and capacity control (Figure 12). In most approaches the order generation represents no task of production control. It can in fact be allocated in the production planning layer and will therefore not be considered here (Figure 12). The capacity control is part of the production control layer but will not be examined in this work because it deals with short-term deviation control in the context of capacity management and we focus on control methods for storage and utilization of orders in work systems. Therefore we only investigate columns 2 and 3 closer.

Order release can be classified into three procedures: immediate order release, order release according to a fixed schedule and stock-controlled order release. Immediate order release frees an order immediately after its creation. Inventory, lead time and utilization of production cannot be directly influenced by this type of order release. In scheduled order releases the plan inflow affects the to-be inflow. Therefore the order will only be released when the planned start date was reached. Stock-controlled order releases can be divided into specific sub-classes (Figure 13). Other authors also use the stock-controlled order release as an independent procedure class and mean the mentioned procedures (Nyhuis 2008, p. 229; Dickmann 2008, p. 178).

Sequence planning can be divided into exact methods (process-optimizing) and heuristic methods (non-optimizing procedures) which are mainly global methods. Linear optimization, complete enumeration and decision trees are exact methods. Heuristic methods can be subdivided into initial algorithms, search and improvement algorithms, priority rules and methods of artificial intelligence (Zäpfel 2001, p. 212; Majohr 2008, p. 34; Rehm 2010, p. 48; see Figure 14).

4 VALIDATION AND VERIFICATION

Model transformations are an active research field, in particular since the model-driven software development gained importance. Part of the research is (apart from the development of approaches for describ-

ing such transformations) the development of appropriate techniques for their validation and verification. These are necessary to validate the developed model transformations with respect to given criteria for correctness.

- not applicable	X applicable	O not exact definable

	Tasks of Production Control				
Author	Order Generation	Order Release	Sequence Planning	Capacity Control	Approach
Kiener et al.	-	-	X	X	MRP II
Mathar / Scheuring	-	X	X	X	MRP II
Hackstein	-	-	X	X	MRP II
Adam	-	-	-	X	MRP II
Scheer	O	O	O	O	Aachener MRP II
Yang / Kiener et al.	O	O	O	O	APS
Scheer / Abramovici / Schulte	-	-	O	O	CIM
Lödding /Nyhuis	O	X	X	X	Task-oriented Approach
Conclusion	-	X	X	X	Conclusion

Figure 12: Tasks of production control

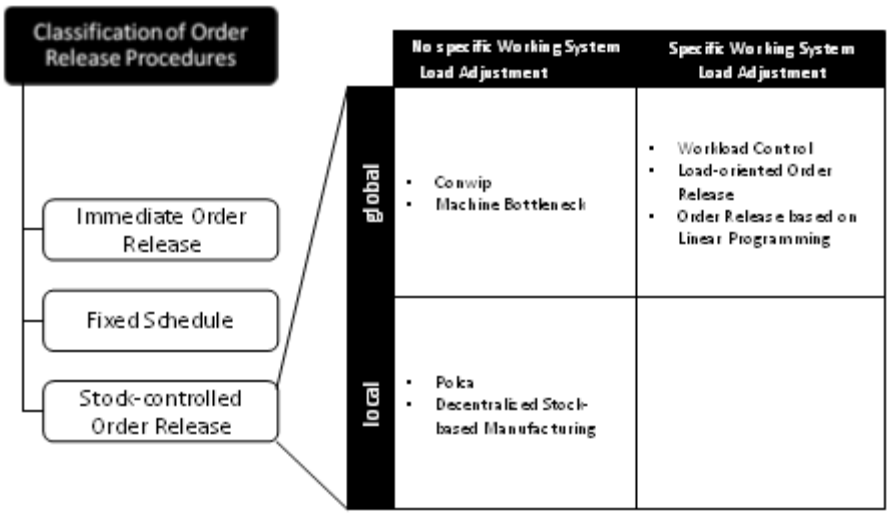

Figure 13: Classification of order release procedures (adapted from Lödding (2008, p. 311))

Verification and validation are two common terms in literature, which are often used synonymously, although they pursue generally different goals (Schatten et al. 2010, p. 5). In the literature the terms are defined differently. Without going into detail of the many different definitions, we use the following definition in this paper: the validation of model transformations is a process, which evaluates whether the specification of the transformation is sufficient for the requirements of the transformation. The verification of model transformations is a process, which evaluates whether the implementation of transformation satisfies the specification of transformation.

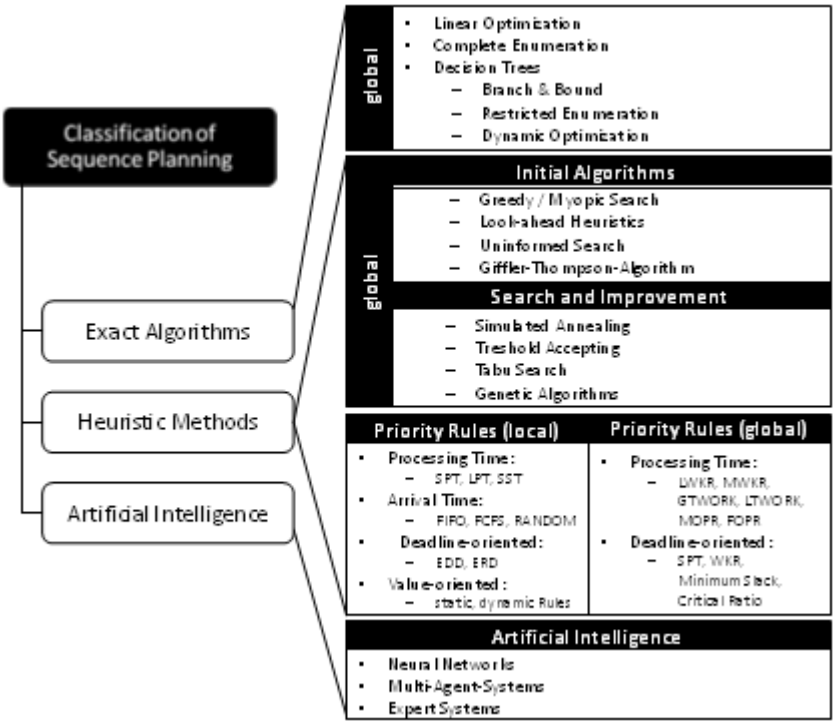

Figure 14: Classification of sequence planning

We defined two requirements for our development of transformations. The transformation should map SysML models into simulation models and be syntactically, functionally and semantically correct. During validation a test will be performed whether the rules and algorithms satisfy the requirements. During verification, however, it is evaluated whether the implementation is correct according to the rules and algorithms.

The validation and verification of a model transformation is used to test their correctness. For this it is important which criteria will be assessed. The following correctness criteria can be found in the literature (Varro and Patariza 2003, Narayanan and Karsai 2008).

- Syntactical correctness. The transformation produces a syntactically correct model or the transformation is syntactically correct with respect to their language.
- Syntactical completeness. The transformation includes all elements of the original model.
- Termination. The transformation is completed after a finite time.
- Confluency, uniqueness. The transformation produces always the same result/output independently from the order the transformation rules are applied.
- Semantic correctness (dynamic consistency, preservation of behavior). In the transformation, the semantics of the original model or important semantic properties are preserved.

Model transformations can be described imperatively (as in our approach) or with graph-grammars (Huang, Ramamurthy, and McGinnis 2007, p.799). In a recent work, we intended to compare both approaches in our field of research. While V&V methods for model transformation using imperative methodology are quite rare, there are several approaches for the model transformation with graph grammars. There are two fundamentally different approaches to formally verify model transformations (Leitner 2006). On the one hand for certain specified criteria it can formally be proven that the source and target models are semantically equivalent. This equivalence is checked automatically at each transformation by a "checker." This approach is called checker-approach (Leitner 2006) for formal model transformation

verification (Varro and Patariza 2003). In contrast, the rule-based approach (Leitner 2006) or meta model level verification (Varro and Patariza 2003) proves the general semantic correctness of a transformation for each input. In a recent work we considered the various approaches in more detail (Scharfe 2011).

4.1 Verification by Structural Correspondences

Now we show the method for verification of model transformations by structural correspondences (Narayanan and Karsai 2008), which is comparable with the approach of correspondence graphs by Triple Graph Grammars (Ehrig et. al 2007). This approach was introduced for graph grammars, but can also be adapted in a way that it can be used for the imperative transformation approach.

The method of verification of the structural correspondences does not verify the translator in general, but only certifies the model instances as transformed correctly with respect to the transformation specification. During the transformation, certain structures of the original model are assigned to related structures of the target model according to corresponding transformation rules. To test these structures after the transformation, we define for each pair of structures a set of correspondence rules. A structural correspondence rule describes the specific relation of elements of the corresponding structures. So it defines how a description of a corresponding target element out of a source element needs to be done in detail.

In order not to make a complete traversal through both model instances, the elements of the meta model of the input and target description language are connected by cross links. For each pair of structures of the source model and the corresponding target model all rules of the particular amount of correspondence rules are evaluated. Even if only one rule is not met, the target model will be considered as a defective transformed model.

The cross-links are defined at the meta model level and instantiated during the transformation. After the transformation the cross-links are available together with the source and target model. It is then sufficient to go through the initial model instance and to identify its structures. If there is no cross-link existing for the structure, an error occurred during the transformation. However, if there is a cross-link, the structure of the target model and the correspondence rules are available for verification.

4.2 Approaches for Validation and Verification

The syntactical correctness as well as the syntactical completeness of the transformation from SysML models to simulation models can be proven by an informal descriptive analysis of the implementation (Scharfe 2011). Furthermore the syntactical correctness and completeness of a concrete model transformation can be checked by the described rules.

With respect to confluence, the termination of a plug-in can be proven by a descriptive analysis of the implementation. It must be demonstrated that the transformation is a monotonous decreasing function (Küster, Heckel, and Engels 2003). Again, the existing implementation of the translation can be analyzed to show that in the transformation only deterministic decisions are made. Therefore we were able to show in our V&V that non-deterministic data structures (Java Sets) create non-confluent translations. It is also possible to prove unity formally by proving that each pair of transformation rules is independent with respect to their execution order (Küster, Heckel, and Engels 2003; Heckel, Kuster, and Taentzer 2002).

Unlike the other correctness criteria, the semantics is barely testable automatically. Since we defined the semantics in the initial SysML model ourselves, it is easy to validate the semantics of the target model by testing. If the correct semantics of each element is proven, the correct semantics of the whole translated model is shown.

5 SUMMARY AND OUTLOOK

We tried to identify and structure the significant properties of discrete processes especially for production systems to give them a theoretical basis (Section 3). Additionally, we evaluated whether SysML is suitable to build models according to our concept. We split our models into a structural, a behavioral and a control part. It turned out that SysML is comprehensively usable for all three parts. Moreover, we devel-

oped a practical approach for automated model generation of a few simulators based on SysML models. With this approach we are able to transform models from SysML into models of a few commercial simulators and from one simulator into SysML. At first, we developed our approach for the domain of production systems. Recently, we also modeled scenarios from the domains of logistics and civil engineering (Rehm, Schönherr, and Schmidt 2011).

Another part of the research is the development of appropriate techniques for the validation and verification of the model transformation. These are necessary to validate the developed model transformations with respect to established criteria for correctness. While V&V methods for model transformation using imperative methodology are quite rare there are several approaches for the model transformation with graph grammars. In this paper we gave an approach to validate the imperative model transformation.

A practical problem of using SysML for discrete systems is that no suitable modeling tools are available. Therefore we develop our own tool for modeling discrete systems with SysML especially suited for engineers (Schönherr and Rose 2010).

REFERENCES

Dickmann, P. 2008. *Schlanker Materialfluss mit Lean Production, Kanban und Innovationen*. 2nd ed. Berlin, Heidelberg: Springer.

Ehrig, H., K . Ehrig, C. Ermel, F. Hermann and G. Taentzer. 2007. "Information Preserving Bidirectional Model Transformations." In: *Fundamental Approaches to Software Engineering*, edited by M. Dwyer and A. Lopes, 72–86.

Fowler, M. 2003. *UML Distilled: A Brief Guide to the Standard Object Modeling Language*. Reading, Mass.: Pearson Education.

Heckel, R., J. M. Kuster, and G. Taentzer. 2002. "Confluence of Typed Attributed Graph Transformation." In *Proceedings of the First International Conference of Graph Transformation*, LNCS 2505:161–176.

Huang E., R. Ramamurthy, and L. McGinnis. 2007, "System and simulation modeling using SysML." In *Proceedings of the 2007 Winter Simulation Conference,* edited by S. G. Henderson, B. Biller, M.-H Hsieh, J. Shortle, J. D. Tew, and R. R. Barton, 796-803. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Küster, J., R. Heckel, and G. Engels. 2003. "Defining and validating transformations of UML Models." In *Proceedings of the HCC 2003*, edited by IEEE Computer Society, 145-152. Washington, DC.

Leitner, J. 2006. "Verifikation von Modelltransformationen basierend auf Triple Graph Grammatiken." Unpublished master thesis, Department of Computer Science, Berlin University of Technology.

Lödding, H. 2008. *Verfahren der Fertigungssteuerung*. Berlin, Heidelberg, New York: Springer.

Majohr, M.-F. 2008. "*Heuristik zur personalorientierten Steuerung von komplexen Montagesystemen*." Dresdner Fügetechnische Berichte, Band 16/2008. Dresden: TUDPress.

Narayanan, A., and G. Karasai. 2008. "Verifying Model Transformations by Structural Correspondence." In *Proceedings of the Seventh International Workshop on Graph Transformation and Visual Modeling Techniques 2008*.

Nyhuis, P. 2008. *Beiträge zu einer Theorie der Logistik*." Berlin, Heidelberg: Springer.

Piepper, D., C. Röttgers, and V. Gruhn. 2006. *MDA: effektives Software-Engineering mit UML 2 und Eclipse*. Berlin: Springer Verlag.

Rehm, M. 2010. "Konzipierung eines Metamodells zur Beschreibung von Produktionssystemen." Unpublished internship thesis, Department of Computer Science, Dresden University of Technology.

Rehm, M., O. Schönherr, and T. Schmidt. 2011. "Ein Metamodell von Produktionssystemen als Grundlage für die automatische Simulationsmodellgenerierung." *Logistics Journal* 1:121-132 .

Schönherr, O., and O. Rose. 2009. "First Steps towards a General SysML Model for Discrete Processes in Production Systems." In *Proceedings of the 2009 Winter Simulation Conference*, edited by M. D.

Rossetti, R. R. Hill, B. Johansson, A. Dunkin and R. G. Ingalls, 1711-1718. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

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

Scharfe, D. 2011. "Development, Verification and Validation of the Translation from Factory Explorer to SysML." Unpublished internship thesis, Department of Computer Science, Dresden University of Technology.

Schatten, A., S. Biffl, M. Demolsky, E. Gostischa-Franta, T. Streicher, and D. Winkler. 2010. *Best Practice Software-Engineering: Eine praxiserprobte Zusammenstellung von komponentenorientierten Konzepten, Methoden und Werkzeugen. Spektrum*. Heidelberg: Akademischer Verlag.

Störrle, H. 2005. *UML 2 für Studenten*. München: Pearson Verlag.

Weilkiens, T., and B. Östereich. 2006. *UML-2-Zertifizierung: Fundamental, Intermediate und Advanced. Test-Vorbereitung zum OMG Certified UML Professional*. Heidelberg: Dpunkt Verlag.

Varro, D., and A. Patariza. 2003. "Automated Formal Verification of Model Transformations." In: *Proceedings of the UML'03 Workshop*, edited by J. Jürjens, B. Rumpe, R. France, and B. Fernandez, 63-78. Technische Universität München.

Weilkiens, T. 2006. *Systems Engineering mit SysML/UML*. Heidelberg: Dpunkt Verlag.

Zäpfel, G. 2001. *Grundzüge des Produktions- und Logistikmanagement*. 2nd ed. München: Oldenbourg.

AUTHOR BIOGRAPHIES

OLIVER SCHÖNHERR is a PhD student at the Dresden University of Technology. 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@tu-dresden.de.

OLIVER ROSE holds the Chair for Modeling and Simulation at the Institute of Applied Computer Science of the Dresden University of Technology, 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.