Platform-independent Modeling for Simulation-based Energy Optimization in Industrial Production

Bernhard Heinzl, Wolfgang Kastner
Institute of Computer Engineering, Research Group of Automation Systems, TU Wien, Vienna, Austria.
bernhard.heinzl@tuwien.ac.at

Abstract - Dynamic simulation models can help to assess and evaluate different planning scenarios in complex industrial production facilities in order to optimize energy efficiency. However, different planning strategies impose different requirements on these simulation models regarding their runtime efficiency and accuracy, thereby raising the need to combine different levels of modeling detail and consider multiple modeling paradigms (hybrid, discrete-event, etc.). In an effort to support such multi-paradigm modeling in practical applications, we present an approach for higher-level component-based specification of industrial simulation models. This domain-specific model abstraction is independent of any concrete modeling paradigm and allows to specify application models in a platform-independent manner by separating the model specification from the concrete implementation. We employ a model-driven engineering methodology by providing metamodels and model transformations that automatically generate different implementations from a unified abstract model specification. The approach is demonstrated on a real-world application example of an industrial bakery.

Keywords - model-driven development, metamodel, multi-stage optimization, hybrid simulation, DEVS, SimEvents, Cube.

I. INTRODUCTION

Many of today's methods for improving energy efficiency in planning and operation of industrial production facilities rely on simulation to evaluate different scenarios regarding their economic and ecological impact. Heuristic and metaheuristic optimization algorithms can employ simulation-based strategies to find e.g. a short-term optimal production plan that covers production goals while also making use of additional leeway to reduce energy costs by considering fluctuating energy prices throughout the day. Simulation-based optimization strategies rely on runtime-efficient simulation models of production equipment that accurately model material as well as energy flow with sufficient level of detail. This typically requires combining discrete and continuous simulation methods as part of a hybrid simulation approach. Hybrid modeling and simulation is still challenging in practical application as conventional approaches (e.g. co-simulation) come with significant drawbacks regarding modularity and component reusability.

In addition, different production planning strategies often focus on different goals and thereby impose different requirements on the simulation. For example, production smoothing strategies try to streamline resource demand over a longer planning horizon, for which a purely discrete-event material flow simulation is better suited.

In this paper, we test a model-driven engineering approach to facilitate the development of domain-specific simulation models of industrial facilities. This approach specifies component-based simulation models independently of their concrete implementation formalism and therefore allows to derive different implementation variants (discrete, hybrid, etc.) that serve different requirements.

II. BACKGROUND

A. Production Optimization

Production optimization deals with finding an optimal combination of production resources, such as equipment, utilities or energy to achieve a given production target in the best possible way. Operative Production Planning and Control (PPC) strategies consider energy together with other production resources and cost factors to optimize production schedules.

Highly fluctuating production volumes with long lead times profit from a two-stage hierarchical planning process [1] with different planning horizons (see Figure 1) where the PPC for short-term planning is complemented by a long-term production smoothing strategy [2]. The idea behind production smoothing follows the Japanese heijunka concept that is employed e.g. as part of the Toyota Production System [3]. Heijunka tries to achieve a ‘balanced’ production with constant flow of material through the whole value stream in order to avoid friction between production steps and avoid product batching [4]. This ultimately results in lean manufacturing with minimum inventories, waste, manpower, and production lead time.

Together, short-term and long-term planning exploit synergies between daily operative planning optimization and periodic long-term alignment of production volumes, as part of a continuous planning strategy with integrated consideration of material, resource and energy flow.
Modern APS (Advanced Planning and Scheduling) software systems offer integrated resource planning and control for production optimization using mathematical optimization methods [5].

Some heuristic and metaheuristic optimization methods rely on simulation models for evaluating each planning scenario in an iterative manner and quantifying their fitness in order to find an optimal solution. These simulation-based approaches [6] allow to consider more complex systems than conventional analytical models, thereby offering more accurate predictions and overall improving planning quality.

In a two-stage planning process, like pictured in Figure 1, both planning stages impose different requirements on a simulation model: While production smoothing considers long planning horizons and therefore needs runtime-efficient simulations, short-term PPC benefits from more accurate and comprehensive models that include energy simulations. Yet, developing two distinct simulation models for one use case is too time-consuming for most practical applications as keeping them consistent creates significant overhead. What is needed in practice is efficient model management with a unified model specification that only needs to be developed once, see Figure 1.

In this paper, we focus model management and simulation as part of an overall multi-stage optimization strategy for energy efficiency and production optimization.

B. Hybrid Simulation

For energy-aware PPC with simulation-based optimization, the dynamic simulation model has to capture the complex interactions between energy and material flow within industrial facilities with sufficient level of detail. For comprehensive energy investigations, it also has to incorporate aspects from different engineering domains (production, energy system, building physics). While material flow entities are typically modelled using discrete-event methods, energy flow (including transient effects) is best described using time-continuous dynamics with differential equations. Integrating discrete and continuous simulation as part of a hybrid modelling approach remains a challenging task. As [7] points out, only few publications focus on hybrid systems in the context of production simulation.

Common approaches for hybrid simulation couple different simulation tools and methods as part of a multi-method co-simulation [8], [9]. These simulation tools are coupled on the “application level” [10], meaning that each new application model requires re-implementing simulator coupling [11]. The user is also forced to split the overall application model into different sub-models for each simulator (e.g. one discrete and one continuous sub-model), thereby loosing modularity of hybrid model components. It quickly becomes cumbersome to maintain these kinds of models [12] and many co-simulation prototypes are one-shot implementation with no real possibility of reuse [13], [14].

Other methods follow a more systematic approach of integrating discrete and continuous models on the “component level”. For example, hyPDEVS is a modeling formalism that extends the basic DEVS (Discrete-Event System Specification) [15] for hybrid systems. The formalism allows to build models from components in a modular and hierarchical manner by distinguishing between atomic and coupled components. While coupled components are solely built from other components (atomic or coupled) and their coupling relationships, atomic components specify internal functions (dint, dext, lambda, f, etc.) that define a certain (discrete as well as continuous) dynamic behavior. The formal model description is accompanied by an abstract simulator algorithm that specifies how hyPDEVS models are executed in practice. More details on hyPDEVS are given in [9], [16].

As a drawback, the hyPDEVS formalism is often difficult to understand for non-experts, which significantly hinders potential adoption for industrial applications. Some implementations may appear counter-intuitive at first, and the generic nature of hyPDEVS requires to take care of low-level implementation details (e.g. message acknowledgment) that are relevant for practical application. More details on this topic are given in [17].

Nevertheless, DEVS formalisms have their advantages due to their rigorous nature and they are well-established in the academic community.

III. MODEL-DRIVEN ENGINEERING APPROACH

In an effort to improve the ease of model development for application engineers – which typically are domain experts but not hyPDEVS formalism experts – as well as to
improve model reuse, we formalize a domain-specific abstraction from hyPDEVS as a platform-independent modeling layer [18]. This increased modeling abstraction allows engineers without in-depth knowledge of the hyPDEVS formalism to develop application models for a particular domain (in this case industrial production facility) by instantiation, parametrization and concatenation of predefined and domain-specific model components.

Following a general Model-Integrated Computing (MIC) paradigm [19], we envision three development stages, as depicted in Figure 2. First, software engineers specify a domain-specific modeling environment (metalevel process) by formalizing a metamodel and a concrete syntax for a domain-specific modeling language (DSL).

This DSL can then be used in the second stage (application development) by domain engineers to create application models conformant to the metamodel. Finally, model transformations and interpreters automatically synthesize executable simulation models.

The application development process in particular employs a model-driven engineering approach, see Figure 3. Starting from a first (informal) Conceptual Model (CM), application engineers develop and formalize a platform and simulation system-independent model description (PIM) in a domain-specific modeling environment using a DSL by instantiating and configuring predefined model components.

The PIM is intended to be independent of specific implementation details in order to facilitate reusability of the higher-level specification for different implementations, cf. Figure 3.

The developed PIM can then be transformed into different simulation system-specific implementations using automated model transformations that enrich the high-level specification with implementation-specific details. The transformation process is illustrated in Figure 4. A model-to-model transformation (M2M) between the PIM and PSM is specified using the respective metamodels. For that, all metamodels conform to a common meta-metamodel. Subsequent model-to-text transformations (M2T) generate executable source code from the PSM. Thereby, information about parameters and couplings is supplemented with source code describing the internal dynamics, which is implemented in custom PSM component libraries. More details on the transformations and their implementation are given later.
Figure 4. Transformation process from PIM to PSM to executable source code.

The so created executable simulation model can then be deployed and used in an industrial environment, where it can perform tasks like evaluating different planning scenarios regarding their energy efficiency as part of a simulation based production planning process.

The platform-independent model abstraction allows to separate the specification from the implementation [20]. This way, different executable models can be derived from the same specification, hybrid models as well as purely discrete models that replace the continuous parts with a discrete approximation in order to gain simulation speed at the expense of accuracy.

IV. PIM LAYER: HYPIM

The platform-independent modeling layer is formalized by providing a formal metamodel, to which all application models have to conform. The metamodel also forms the basis for the abstract syntax of a domain-specific modeling language [21].

A. The Cube Concept

A platform-independent model formalization requires defining what constitutes a “component” within a model and describing a component in an abstract manner. For this purpose, we introduce the concept of Cubes [11], [22]. A Cube represents a uniform modeling unit in the PIM layer. A Cube may incorporate continuous and discrete dynamic aspects, thus integrating hybrid modelling on the “component level”. Figure 5 shows an illustrative example of an industrial conveyor oven as a Cube.

The Cube concept follows a component-based paradigm for model development [23], where well-defined model components encapsulate a certain internal dynamics, which, when composed with other components into larger models, together describe the overall model behavior. Component-based modeling facilitates modularity and separation of concerns, which is indispensable for managing the complexity of today’s industrial systems [24].
Figure 5. Industrial oven as a Cube model. The model includes discrete dynamics (for material flow), continuous transient dynamics (heating/cooling) as well as interface ports to other cubes.

Model development can be distributed among different development experts, i.e., software engineers implement libraries of tested and validated model components, which application engineers can then instantiate and use in different contexts, thereby facilitating model reuse. Model reuse is crucial for reducing development effort (and costs) of new application models [20], [23]. However, in order to retain modularity (and thus reusability), it is necessary to encapsulate all aspects of a model component within uniform component boundaries. This can present a challenge in the context of hybrid simulation, where discrete and continuous aspects have to be combined in a modular manner.

B. hyPIM Metamodel

PIMs are specified as an instance of the Cube metamodel, called hyPIM. This metamodel, shown in Figure 6, formalizes how platform-independent models can be specified out of predefined Cube components, like the oven Cube presented above. Some details of the metamodel have been omitted for reasons of clarity.

Every model is an instance of class System, which contains instances of Cubes, which are separated into four domains: BuildingCube, EnergyCube, LogisticsCube, and ProductionCube.

Cubes can be arranged hierarchically, i.e., Cubes can contains other Cubes. A Cube also comprises ports (EntityPorts, EnergyPorts, and InformationPorts) as well as Connections. Also, each Cube may include a list of parameters (ParameterList).

This metamodel focuses primarily on composition of predefined Cube components by instantiation, parametrization and specifying coupling relationships.

V. PSM LAYER: HYPDEVS

A. hyPDEVS Metamodel

For transforming a model specification into a hyPDEVS-specific implementation using model-to-model transformation, we formalize a hyPDEVS metamodel, shown in Figure 8. The metamodel is based on several metamodels for classic DEVS that have been published in the literature [25]–[28] and has been adapted for the hyPDEVS formalism.

B. Model-to-Model Transformation

As a proof of concept, all metamodels and transformations have been implemented using the Eclipse EMF Framework [29], [30]. The metamodels are instances of the Ecore metamodel (cf. Figure 4), which is provided by the EMF. Model-to-model transformations are implemented using ATL (ATLAS Transformation Language) [21]. ATL allows to specify transformation rules between metamodels in a declarative manner. A transformation engine executes these
rules on a given source model (PIM) to produce a target model (PSM).

An ATL transformation rule can produce a single or multiple \textit{DEVSComponents} (Atomic or Coupled) from a single Cube. Also, additional ports and connections may be created. Such additional elements may be necessary for certain platform-specific implementations but are not explicitly modeled in the PIM because they can be derived from other information given in the PIM and these details can therefore be abstracted away from the user. For example, for each entity connection (EOUT–EIN) in the PIM, an additional feedback connection (EINcom–EOUTcom) can be added that serves as an acknowledgment channel, see Figure 7. This feedback channel is necessary in the hyPDEVS implementation due to the event handling in hyPDEVS to make sure that no entities are lost. More details on these modeling issues are given in [17].

![Figure 6. hyPIM metamodel (simplified)](image1)

![Figure 7. An entity connection in the PIM (left) is transformed to two connections and corresponding ports (EOUT–EIN, EINcom–EOUTcom) in the PSM (right).](image2)

**C. Code Generation**

For executing hyPDEVS models, we employ the MatlabDEVS toolbox [16], for which we generate MATLAB source code using Xtend as a model-to-text (M2T) transformation language. Code generation is based on code templates, which are populated with dynamic data from the PSM [21] to instantiate, parametrize and couple together hyPDEVS components, which themselves have been implemented directly in MatlabDEVS as part of a hyPDEVS component library, cf. Figure 4. Together, this code forms the application model, which can then be deployed and simulated using the simulation engine provided by the MatlabDEVS toolbox.

**VI. PSM LAYER: SIMEVENTS**

Besides hybrid simulation models, also purely discrete simulation models can be derived from the same hyPIM Cube model, since all the necessary information is already available in the model description. Such discrete models can be beneficial for investigating material flow and optimizing production scheduling without looking at energy aspects. While one could of course also use a hybrid model for this
purpose, a purely discrete simulation might be more efficient in terms of runtime, which is especially important for simulation-based optimization algorithms that use possible thousands of simulation runs [31]. In the following, we employ MATLAB/SimEvents as a simulator for discrete-event simulation.

A. SimEvents Metamodel

Similar to above, we use model-to-model transformation for generating a SimEvents-specific model from the platform-independent Cube model. The purpose of this transformation is to reduce the semantic gap and to enrich the model with platform-specific information. To this end, we devised a simple SimEvents metamodel. The core of this metamodel is similar to the hyPDEVS metamodel (see Figure 8), with SimEvents subsystems corresponding to coupled components and SimEvents blocks corresponding to hyPDEVS atomics.

B. Model-to-Model Transformation

The model-to-model transformation is implemented again using ATL. Compared to the hyPDEVS ATL transformation, the rules are more straightforward. For example, there is no need to create acknowledgement channels for entity ports. On the downside, the transformation has to map each port to a number corresponding to the port numbers in the SimEvents library. These port numbers are necessary for accessing ports during code generation for the port connections, see the code in Figure 9. Also, implementing the Cube’s internal behavior is more cumbersome due to some of SimEvent’s limitations.

As a result, many SimEvents library blocks are necessary to implement a single Cube, like shown in Figure 10.

C. Code Generation

Using Xtend code generation, we generate a MATLAB script from the PSM, which then builds the SimEvents model using add_block(), add_line(), and set_param() commands to instantiate, connect, and configure components from a custom-built SimEvents library. The Figure 9 shows a snippet of this code for instantiating an oven component (which is part of the Cubeslib SimEvents library).
VII. APPLICATION EXAMPLE

To demonstrate the presented approach in a practical application, we developed a model of an industrial production facility [17]. The conceptual model is illustrated in Figure 11 and represents a simplified version of an industrial bakery that produces fresh as well as frozen baked goods. The model includes production machines, conveyor belts, energy systems for supplying electric as well as heating and cooling energy (including energy storage), and a building model with four thermal zones. The model combines material flow (discrete models) with energy flow (continuous models) in a modular manner and considers interdependencies between different domains (e.g., waste heat from the oven provides a heat gain for the thermal building model, which in turn influences the heating and cooling demand for the heating grid).

This conceptual model has been formalized and implemented as a hyPIM model. Also, component implementations have been developed in MatlabDEVS and SimEvents Cube libraries, respectively.

Figure 10. SimEvents implementation of the oven model.

Figure 11. Conceptual model of an industrial bakery featuring production machines, logistics components, energy supply, and thermal building zones.
A. Hybrid Simulation

The implemented hyPIM model of the bakery can be transformed automatically into an executable MatlabDEVS hybrid simulation using the M2M and M2T transformation process described above. The final result is depicted in Figure 12. The code snippet shows instantiation, parametrization and connection of an oven model (oven cube), which has been implemented as part of a MatlabDEVS cube library.

This code can directly be executed in MatlabDEVS as a hybrid discrete/continuous simulation. Such a simulation can be used, for example, in an industrial environment to evaluate different production schedules and operational strategies of energy supply systems with regard to energy demand, material flow as well as production and storage costs. All these aspects can be combined in a multi-criteria optimization problem, where metaheuristic optimization techniques can be employed to find the best operational settings [31].

```matlab
... %Oven
a_name = 'oven';
inistates = struct('sigma','inf','p','standby',... 'h','off','ent',[]);
parameters = ...
struct('W','1500','P','500','tB',... 'tset',1,1.5,'oven':'oven','tset',... 'num','10','s','100','o','on','sign',1);
end_model = {'belt1_stat','EOUT','oven','EIN';...
'oven','EINcom','belt1_stat','EOUTcom';...
'oven','EA','sink_waste','EIN';}

Figure 12. MatlabDEVS code snippet of the hybrid simulation model.

B. Discrete-Event Simulation

As presented above, we can also generate a SimEvents simulation from the same hyPIM model description, where predefined library components are instantiated, connected and configured to form the overall model. The library components themselves have been implemented directly in SimEvents, see Figure 10.

In contrast to the hybrid simulation, this purely discrete simulation in SimEvents is more useful for longer observation periods (because of its higher simulation speed), as part of a long-term production planning method with focus on production smoothing and less focus on short-term energy optimization (since the simulation provides less accuracy in predicting energy demand).

VIII. DISCUSSION

On the one hand, platform-independent modeling provides a simplified abstraction from the hyPDEVS formalism that allows non-expert engineers to easily develop hybrid simulation models of industrial facilities. The Cubes provide higher-level semantic integration [32] of discrete and continuous aspects on the component level, thereby facilitating component reuse of hybrid simulation models.

The presented proof of concept focuses on model development by instantiation, connection, and configuration of predefined components from a model library. This approach works well for building relatively similar models that do not need specialized components. For models with special requirements that are not covered by the model library, new components have to be developed in a combined effort between the domain expert and the modeling/software expert. However, as the library grows, the effort for new component development will become less and less while component reuse should increase, resulting in accelerated application development.

This acceleration is magnified by the fact that the presented approach allows to distribute model development among different expert engineers. While application engineers continuously implement and deploy new application models, modeling/software engineers can focus on developing reusable model components.

While it is possible to extend the presented approach to include modeling the internal dynamics of Cubes in a platform-independent manner, it does come with significant challenges because of the large number of domain-specific details that would have to be incorporated. The potential benefit compared to the functionality provided by off-the-shelf domain-specific simulators is subject of future evaluation.

IX. CONCLUSION

This paper demonstrates a method how simulation models with different characteristics, in particular hybrid and discrete models, can be derived from a single platform-independent model specification in a systematic manner.

We defined a platform-independent modeling layer (PIM) by means of a metamodel. Model specifications can be composed from predefined components, called Cubes. M2M and M2T transformations automatically generate executable simulations from the PIM.

This work is intended to facilitate interdisciplinary simulation of industrial production facilities with different level of accuracy for use in simulation-based planning frameworks [31]. Short-term Production Planning and Control (PPC) can be combined with long-term production smoothing, each using a simulation model targeted to their requirements (runtime, accuracy). A unified model specification, which is separated from the actual
implementation, reduces development and maintenance effort for these simulation models.

In the future, we intend to investigate other platform-specific implementations that can be derived from the PIM, especially for multi-simulator co-simulation frameworks [33]. The platform-independent model specification will help managing large-scale models and couplings across different simulators and thus facilitate reuse of co-simulations.

REFERENCES


