Integrated Simulation of Domain-Specific Modeling Languages with Petri Net-Based Transformational Semantics

David Mosteller, Michael Haustermann(✉), Daniel Moldt, and Dennis Schmitz

Faculty of Mathematics, Informatics and Natural Sciences, Department of Informatics, University of Hamburg, Hamburg, Germany
haustermann@informatik.uni-hamburg.de
https://www.inf.uni-hamburg.de/inst/ab/art.html

Abstract. The development of domain specific models requires appropriate tool support for modeling and execution. Meta-modeling facilitates solutions for the generation of modeling tools from abstract language specifications. The Rmt approach (Renew Meta-Modeling and Transformation) applies transformational semantics using Petri net formalisms as target languages in order to produce quick results for the development of modeling techniques. The problem with transformational approaches is that the inspection of the system during execution is not possible in the original representation.

We present a concept for providing simulation feedback for domain specific modeling languages (DSML) that are developed with the Rmt approach on the basis of meta-models and transformational semantics using Petri nets. Details of the application of this new approach are illustrated by some well-known constructs of the Business Process Model and Notation (BPMN).

Keywords: Meta-modeling · BPMN · Petri nets · Reference Nets · Simulation · Graphical feedback

1 Introduction

The construction of abstract models is an essential part of software and systems engineering. Meta-modeling provides a conceptual base for developing modeling languages that are tailored to satisfy the demands of specific application domains. Tools may be generated from the language specifications to support the modeling process.

The definition of semantics for domain-specific modeling languages (DSML) and the provision of appropriate execution tools is one of the key challenges in model-driven tool development. One way to define semantics for a DSML is to transform it into another language with well-defined semantics and existing tool support. This is especially suited to produce quick results for rapid prototyping. Bryant et al. identified "the mapping of execution results (e.g., error messages, debugging traces) back into the DSML in a meaningful manner, such that the domain expert using the modeling language understands the result" [5, p. 228] as one of the challenges for the translation semantics approach. Concerning the user experience, meaningful visual representation of the domain concepts is vital for the communication between different stakeholders, especially for the domain experts that are often non-software engineers [1, p. 233]. The representation of DSML in execution is still considered a challenge in tool generation in general [28, p. 196].

We present a concept for the rapid prototyping and integrated simulation of DSML within the Renew simulation environment. This concept integrates the simulation feedback from the executed language into the graphical layer of the original DSML. The focus of this contribution is to equip the language developer with the means necessary to develop iteratively and prototypically the visually animated DSML without being dependent on a difficult implementation or graph transformation language. With our contribution, we combine and advance two branches of our current research: first, the development of domain-specific modeling languages using the Renew Meta-Modeling and Transformation (Rmt) framework [31] and secondly, the provisioning and coupling of multiple modeling perspectives during execution within Renew [29]. The first contribution does not consider the simulation in the source language; it presents the transformation of the source language into Reference Nets without the support for graphical simulation feedback. The second contribution does not use meta-modeling, but presents an approach for coupling and simulating multiple formalisms synchronously. The contribution mentions the execution of finite automata with the possibility of highlighting the active state and state transitions. However, the graphical simulation feedback was hard-coded. In this contribution, we consider the model-based customization of the visual behavior of a simulated DSML. This paper is an extended version of a workshop contribution [32].

The approach provided by the Rmt framework supports the rapid prototypical development of domain-specific modeling languages with Petri net-based semantics. The Rmt approach is based on the idea of providing transformational semantics for the modeling language in development (source language) through a mapping of its constructs to a target language. The latter is implemented using net components [6, Chapter 5], which are reusable and parameterizable code templates – quite comparable to code macros – modeled as Petri nets. The benefit of this approach consists in leveraging the intuitive notation of Petri nets in order to fulfill the task of specifying an explicit (operational) semantics for the target language. This, however, comes at the cost of limited applicability in the sense that the approach becomes less suited for modeling environments that do not conform to the state-based, process-oriented characteristics of Petri nets. This said, we acknowledge the limitations of our approach and develop a solution for the integrated simulation of executable DSML that aligns with the fundamental idea of the RMT approach and its focus on rapid prototypical development of DSML.

We choose Reference Nets [23] as a target language, but are not restricted to this formalism. Reference Nets combine concepts of object-oriented programming and Petri net theory. They are well-suited as a target language because of their concise syntax and broad applicability. With the presented solution Reference Nets provide the operational semantics for the target language. The simulation events are reflected in the source language during execution.

Tool support for our approach comes from Renew, which provides a flexible modeling editor and simulation environment for Petri net formalisms. Renew has a comprehensive development support for the construction of Reference Net-based systems [7, 24].

Various approaches to defining the semantics of DSML with the purpose of simulation exist. In Sect. 2, we start with a comprehensive review of the research on graphical feedback in executable DSML engineering and compare our approach with related work. In this contribution, we extend the Rmt framework (as presented in Sect. 3) with a direct simulation of the DSML's original representation and discuss the integration into the approach. The presented concept for simulation visualization is based on the highlighting of model constructs. This is achieved by reflecting simulation events of the underlying executed Petri nets (target language) into the DSML (source language). Several types of mappings are evaluated in Sect. 4 regarding their expressiveness and features for modeling. A major challenge for the provision of direct simulation support is the integration into model-driven approaches in the sense that the DSML developer can specify the desired representation of the executed models in a model-driven fashion. The concept presented here includes tools that enable DSML developers to create the necessary artifacts and configurations to manage this task. Section 5 introduces the current implementation that processes these artifacts and configurations to initialize a simulation of the DSML model with graphical feedback. In Sect. 6, we discuss multiple alternatives to provide support for DSML developers to specify the desired representation of the executed models in the Rmt approach. As a part of our contribution, a generic compiler is implemented in Renew. This is used in the previously mentioned processing of artifacts and configurations. On this basis, the generated technique may be executed within Renew's simulation engine in its original representation, as presented in Sect. 7. We summarize our results in Sect. 8.

2 Related Work

A vital issue of providing graphical feedback for the execution of DSML is the definition of the language semantics. In their analysis on *Challenges and Directions in Formalizing the Semantics of Modeling Languages* [5] Bryant et al. describe categories of approaches to the definition of semantics of modeling languages: first, the category of rewriting grammars that apply a set of graph transformation rules; second, approaches based on metalanguages that include the operational specifications in the meta-model; third, the transformational approaches, which transfer semantics by mapping the concepts to a target language.

Various approaches to the development of visual languages exist but the notion of graphical feedback or semantics of DSML for simulation purposes differs greatly. The demand for graphical feedback in executable DSML comprises, for instance, visualization of simulation states, feedback from a model checking counterexample, or smooth visual animation of graphical objects. Although the requirements are diverse, the following related work is presented respecting the categories of Bryant et al.: graph-grammar, metalanguages and transformational.

Graph grammars are widely applied to define the semantics of DSML. With the work from Biermann et al. graph rewriting rules are not only applied to provide operational semantics during execution but also to define the behavior of visual editors [2]. With AToM^3 de Lara et al. [26] maintain an approach that relies on graph rewriting systems. Rules may be edited graphically, allowing a low-level entry for the language designer comparable to our rapid prototyping approach. In AToM^3 the graph rewriting rules may be applied in step-by-step, continuous or animation mode [26, p. 199]. To visualize the behavior of the models, they are translated into Java and executed by an external application.

Several approaches aim at providing interactive visual behavior for domain-specific modeling tools by using metalanguages. GEMOC Studio is a comprehensive framework for the development of executable DSML (xDSML) based on the EMF. "The framework provides a generic interface to plug in different execution engines associated to their specific metalanguages used to define the discrete-event operational semantics of DSMLs" [3, p. 84]. The animation component, Sirius animator [18], facilitates reacting to simulation events (execution steps) in order to trigger visual effects in the xDSML. This permits, for instance, the adaption of graphical attributes, such as border width, colors or icon images through style specifications. With integration into the Eclipse debug UI it supports interactive (omniscient) debugging of models including the possibility to move forward and backward in the simulation trace [4]. The tool suite offers four execution engines that are integrated into the framework and conform to the event API. With GEMOC Studio the operational semantics are defined by the application of metalanguages, while we follow a transformational approach.

Hegedüs et al. [15] present a systematic approach for replaying information captured from execution traces into the source model. Graph transformation rules are applied to relate the trace model to the dynamic model of a DSML based on their respective meta-model. The trace information is evaluated externally to the original execution, which conceptually distinguishes the approach taken by the authors from the integrated approach, which is presented in this contribution.

Kindler has a comparable approach to ours using annotations with the ePNK framework to provide simulation capabilities for a meta-model-based modeling application [20]. The simulation in ePNK is realized in the form of different handlers that manipulate the annotations representing the simulation state. A presentation handler implements the representation of these annotations as labels or overlays in the graphical editor.

Cramer and Kastens [11] apply the simulation specification language DSIM as a metalanguage to define the semantics of DSML. The visualization engine interpolates the intermediate states between discrete simulation steps to display a smooth animation of the execution.

Sedrakyan et al. present model-driven graphical feedback in a model-to-code transformation to validate semantic conformance especially for novice modelers [36]. Figure decorations visualize inconsistencies, and error messages provide additional information. They focus on feedback for errors that occur in the compiled system rather than a complete interactive inspection of the execution.

Rybicki et al. [35] use model-to-model transformations to map high-level models to low-level executable code or models. With tracing capability added to their approach, they keep track of the relations between the transformed models. The tracing information is used to propagate runtime information during simulation to the high-level models, which permits the inspection of a simulation in the original representation. However, they do not cover the customization of the representation of the models at runtime and the highlighted constructs.

Petri nets are often used as the target language in transformational approaches in combination with high-level modeling languages. Research questions in the context of transformations to Petri nets are often discussed for specific languages. An overview of semantics for business process languages is, for example, provided by Lohmann et al. [27]. The utilization of semantic components resulting in a `1:n` mapping is related to static hierarchy concepts, such as the concepts for Coloured Petri Nets [17]. The research results in this area can be transferred to our approach. Object-oriented Petri nets may serve as a target language, where the encapsulation of properties is desired [34]. Additionally, the capabilities of Reference Nets regarding dynamic hierarchies can be helpful for the development of more complex semantics.

With the Access/CPN tool Westergaard enables domain-specific visualizations of Coloured Petri Nets by reflection of simulation events [38]. It even becomes possible to embed simulations into externally executed programs and vice versa. However, as the Access/CPN tool makes no assumptions about the external application, the domain-specific visualizations need to be provided individually. While Renew potentially offers an interface for accessing simulation entities through generated Java net stubs, the focus in this contribution is on model-based graphical simulation feedback.

To this end, we apply declarative inscriptions on semantic components to establish the link between simulation and graphical framework. The approach presented in this contribution is unique in the sense that it uses Petri nets as the target language, to provide a solid formal base, in combination with a focus on the representation customization of the executed models.

3 The Rmt Framework

The Rmt framework (Renew Meta-Modeling and Transformation) [31] is a model-driven framework for the agile development of DSML. It follows concepts from software language engineering (SLE [21]) and enables a short development cycle to be appropriately applied in prototyping environments. The Rmt framework is particularly well-suited to develop languages with simulation feedback due to its lightweight approach to SLE and the tight integration with the

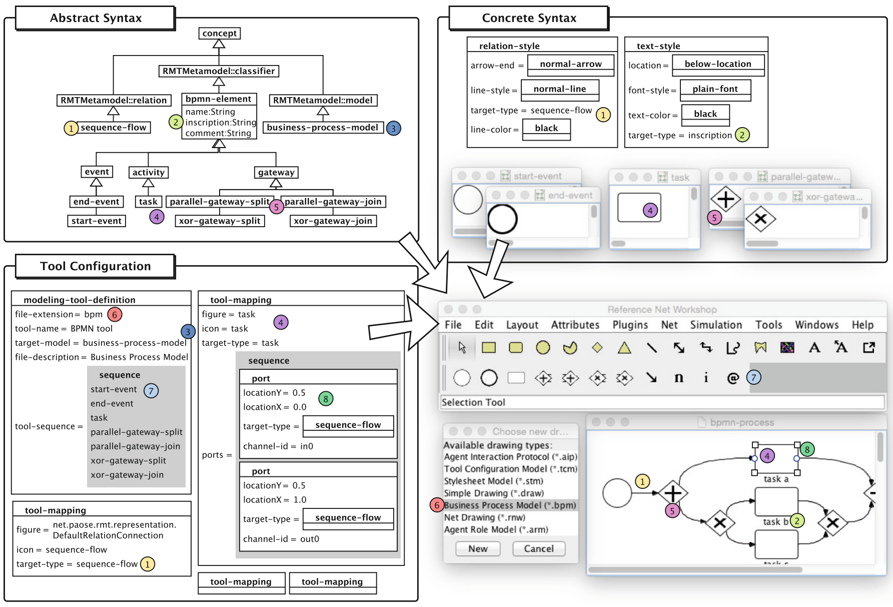


Fig. 1. Excerpt from the RMT models for a BPMN prototype with the generated tool. (Color figure online)

extensible RENEW simulation engine, which supports the propagation of simulation events. Other frameworks for model-driven engineering (MDE), such as the Eclipse Modeling Framework (EMF [13]), could also benefit from the proposed solution for provisioning of simulation feedback. However, this would require integrating an equally adequate simulation engine into the MDE framework.

With the RMT framework, the specification of a language and a corresponding modeling tool may be derived from a set of models, defined by the developer of a modeling technique. A meta-model defines the structure (abstract syntax) of the language, the concepts of its application domain, and their relations. The visual instances (concrete syntax) of the defined concepts and relations are provided using graphical components from RENEW's modeling constructs repertoire. They are configurable by style sheets and complemented with icons and a tool configuration model to facilitate the generation of a modeling tool that nicely integrates into the RENEW development environment.

In this contribution, we use a BPMN (Business Process Model and Notation) prototype as our running example to present how the RMT approach is extended with graphical feedback for simulations. Figure 1 displays a selected excerpt from the models required for the BPMN prototype, together with the tool that is generated from these models. The parts of the figure are linked with colored circles and numbers. The meta-model (upper left) defines the concepts for classifiers (4, 5) and relations (1) of the modeling language and the corresponding model (3). Annotations are realized as attributes of these concepts (2). The concrete syntax (upper right) is provided using graphical components, which are created with

Renew, as it is depicted for the task and the parallel gateway (4, 5). Icon images for the toolbar can be generated from these graphical components. The representation of the inscription annotation (2) and the sequence flow relation (1) is configured with style sheets.

The tool configuration model (lower left) facilitates the connection between abstract and concrete syntax and defines additional tool related settings. The concepts of the meta-model are associated with the previously defined graphical components (4), with custom implementations or default figure classes that are customizable by style sheets (1), and the icons. Connection points for constructs are specified with ports (8). The general tool configuration contains the definition of a file description, an extension (6), and the ordering of tool buttons (7).

With these models, a tool is generated as a Renew plug-in, as shown at the bottom right side of Fig. 1. A complete example of the Rmt models and additional information about the framework and the approach can be found in [31].

4 Net Component-Based Semantics

Our main goal is the provision of an easily applicable approach and easily usable tools for language developers, as well as users. The difficult part is often the definition of the semantics. We propose to apply a mapping of DSML constructs to Petri net constructs, in order to provide operational semantics for the DSML. Within this kind of scenario, graph transformation languages commonly facilitate the semantic mapping either by declarative inscriptions, imperative expressions, or a combination of both [22]. Either way, the developer of a DSML needs to learn the respective language in advance and finally develop the rules for the semantic mappings. With the RMT approach Petri net constructs can simply be modeled with the Renew editor, in a similar way this is done for the graphical components (cf. Fig. 1). The net elements are annotated with attributes and arranged together in one environment, in order to form a net component. This has the advantage that partly completed mappings become immediately executable and testable, which supports the prototypical development process.

On the downside, this has an impact on the number of mapped constructs in the source and target language. Different types of mappings are possible (`1:1`, `1:n`, `n:1`, `n:m`). A general solution that covers semantics for possibly any language would probably require a `n:m` mapping. In this contribution we mainly consider languages that are more abstract than Reference Nets, consequently utilizing `1:n` mappings. Using net components, which group multiple Reference Net constructs into one, the mapping is reduced to cardinalities of `1:1`. A `1:1` mapping restricts the expressiveness but there are options to overcome some of the restrictions. We discuss these further on in this section.

For a direct mapping approach, one of the main questions is how the components are bordered and how they are connected. This depends mainly on the handling of relations. A simple way of connecting two net components is using an arc between them, which requires the connection elements in the net components to be of opposing types. Regarding expressiveness, this approach may be

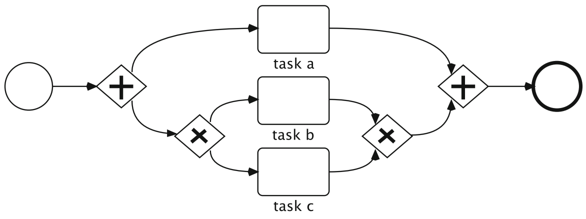
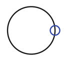
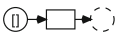
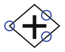
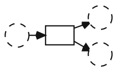
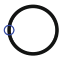
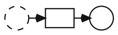
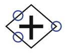
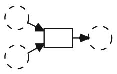
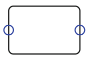
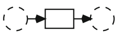
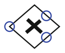
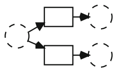
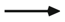
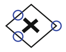
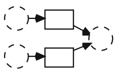


Fig. 2. A BPMN process containing three tasks.

Table 1. Mapping of BPMN and Petri net constructs [12, p. 7].

BPMN	Reference Net	BPMN	Reference Net
			
			
			
			

sufficient for simple P/T net semantics, but in more complicated scenarios that use colored Petri nets, it becomes necessary to maintain control over the data transported through the relations. One way to overcome this issue is to fuse the connection elements of the net components so that the connecting arc is part of one of the net components, which permits adding inscriptions to that arc.

4.1 BPMN Semantics

Consider the BPMN process displayed in Fig. 2, that shows three tasks, where `task a` is being executed in parallel to `task b` and `task c`, which are mutually exclusively alternative.

There are many ways of defining the semantic mapping. Regarding BPMN, the specification itself [33] describes informal semantics for the execution of BPMN processes based on tokens. Therefore, the Petri net semantics – at least for a subset of BPMN constructs – is straight-forward and may be implemented using a mapping to Petri nets that was proposed by Dijkman et al. [12, p. 7] as displayed in Table 1. Covering the full BPMN standard with Petri net implementations is a challenge of its own that we do not try to resolve in the context of this work. Even this small selected subset of BPMN constructs leaves enough room

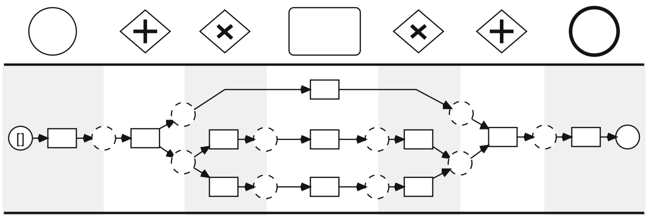


Fig. 3. Place bordered decomposition of the BPMN process.

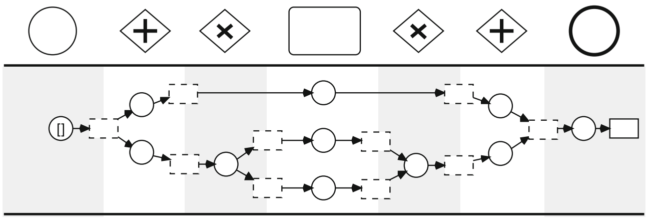


Fig. 4. Transition bordered decomposition of the BPMN process.

for interpretation and discussion about general concepts of Petri net mappings using semantic components.

Each of the Petri nets from Figs. 3 and 4 implements the BPMN process using a slightly different semantics. The vertical lines represent graphical cuts (not Petri net cuts) through the process, which indicate the fusion points between components (highlighted by dashed lines). The first Petri net implements the original mapping from Dijkman et al. [12]. It uses place bordered components and fuses the elements along the places. The sequence flow relations are mapped to places, which dissolve in the fusion points and have no individual semantics. The second net is very similar but uses transition bordered components. These components apply a mapping of sequence flow relations to transitions. Note how the semantics of the individual components slightly varies. For instance, the end event terminates with a marked place when applying the mapping from Dijkman et al. The BPMN standard prescribes that each token must eventually be removed from the process [33, p. 25], so the variant from Fig. 4 is more in conformance with the BPMN standard regarding this aspect. On the other hand, the second variant defines a task as a place surrounded by two transitions (incoming and outgoing), so its character is more like a state rather than an event or process. The BPMN execution semantics of a task is much more complex than this abstract interpretation, but it begins with a state (Inactive) and ends with a state (Closed) [33, p. 428].

There are many possible variants of the semantic mapping from Table 1. Each of the semantic components can be refined with additional net structure, as long as the bordering remains unaltered. The same applies to the relations, in case it becomes desirable to attach semantics to the relations in a similar way, this is done for the constructs.

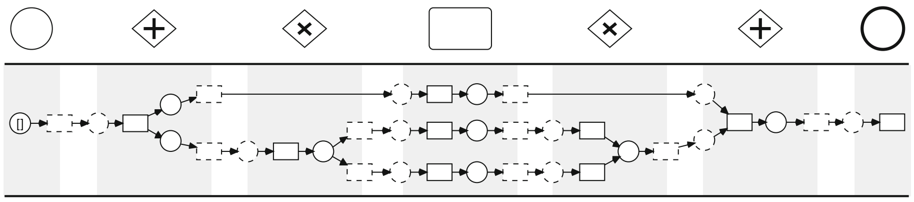


Fig. 5. A Petri net mapping of the BPMN process that consists of alternating semantic components.

Figure 5 again shows an alternative semantics of the presented BPMN process. The bordering of components is alternating in the sense that each BPMN construct is place bordered on the incoming and transition bordered on the outgoing side. The bordering of sequence flow relations is the opposite way around so that it provides the proper fusion points to complement the BPMN constructs. Following the BPMN standard, the outgoing sequence flows of a conditional gateway hold the condition inscriptions. With this mapping, it is possible to specify the conditions on the sequence flow. These could be transformed to guard expressions on the transitions of the mapped sequence flow, which are merged with the outgoing transitions of the conditional gateway in Fig. 5. This allows the mapping of the inscription to remain within the locality of the mapped construct. However, as we do not discuss inscriptions in detail in this contribution, this argument can be omitted. The alternating semantics has a different advantage for defining the highlighting in Sect. 6. Each component locally encapsulates its states and behavior. This property will be used to define two highlighting states "active" and "enabled" for the simulation of BPMN models.

4.2 Reference Nets as a Target Language

The (Java) Reference Net formalism[1] [23] is a high-level Petri net formalism that combines the nets-within-nets paradigm [37] with synchronous channels [9] and a Java inscription language. This formalism makes it possible to build complex systems using dynamic net hierarchies. The nets-within-nets concept is implemented using a reference semantics so that tokens can be references to nets. With the Java inscription language, it is possible to use Java objects as tokens and execute Java code during the firing of transitions. The synchronous channels enable the synchronous firing of multiple transitions distributed among multiple nets and a bidirectional exchange of information. An introduction to Reference Nets is available in the Renew manual [25], the formal definition can be found in [23] (in German).

In comparison to the previously introduced bordering semantics with Reference Nets, additional variants become possible, such as the connection via virtual

[1] The first paragraph originates from one of our previously published contributions [30].

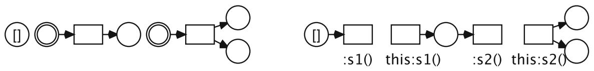


Fig. 6. Connection semantics of Reference Nets.

places or synchronous channels. Displayed on the left side in Fig. 6 are virtual places.

The double-edged place figures are virtual copies of their respective counterpart in the sense that the two places represent the same semantic place but with multiple graphical representations. These could be utilized to implement the place fusion. The synchronous channels displayed to the right are even more powerful. The synchronization of two transition instances supports the bidirectional exchange of information through unification of the channels. Besides supporting the possibility to move information along the edges, synchronous channels provide facilities to define interfaces to query and modify data objects. With syntactical constructs for net instantiation Reference Nets provide the capabilities of modeling dynamic hierarchies. In the context of BPMN, this is useful to implement hierarchies, such as pools and sub-processes. Corradini et al. [10] have identified that existing formalizations of BPMN do not sufficiently account for collaboration across multiple instances, especially regarding the exchange of data and data-based decisions. The authors provide a token-based operational semantics that focusses on these subjects and develop a visual editor that supports animation. According to this model, the synchronous channels of Reference Nets can be used for the formalization of data exchange in collaborative processes. It is something we have applied for quite some time [8]. The focus of this contribution is on the rapid prototypical development of DSML. The results are not yet transferred to BPMN.

Providing a complete formalization of BPMN is a challenge of its own. Van Gorp and Dijkman [14] and Kheldoun et al. [19] aim at covering large subsets of BPMN but still do not cover the full standard. Kheldoun et al. use high-level Petri nets in a similar fashion to our approach for the provision of a formal semantics but without execution support in favor of a verification focus.

5 Graphical Simulation Feedback for DSML

Up to now, there was no (sufficient) user feedback during the execution of models generated by the modeling technique. In this paper, we address the extension of our framework to enable simulation feedback from the underlying Petri net. The main idea is that within the domain models, the internal state (resp. marking) of the Petri net is reflected directly in the domain of the generated modeling technique. This allows for adaptive feedback individually depending on the transformational semantics for each generated modeling technique.

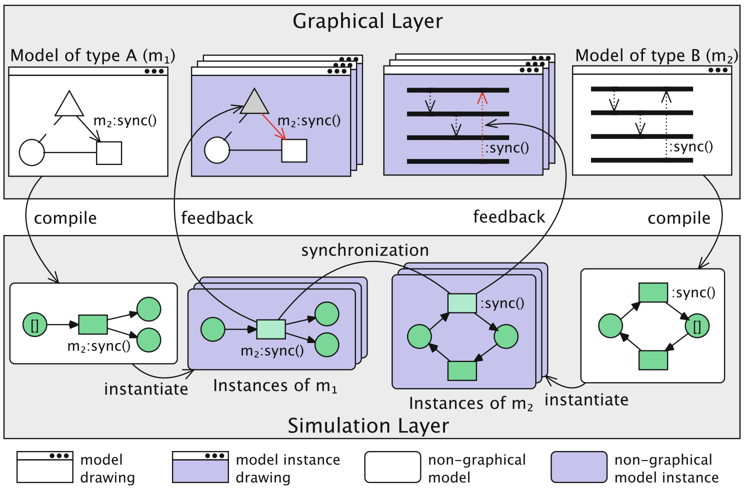


Fig. 7. Conceptual model of the model synchronization from [29, p. 8].

A conceptual image from the simulation of two modeling techniques that interact with each other is displayed in Fig. 7. The image originates from [29, p. 8], where we presented a concept for multi-formalism simulation with the synchronization of multiple modeling techniques based on Reference Nets. The presented solution sketched the idea of providing feedback into a DSML, but the realization was specifically for a finite automata modeling tool. With our current work, this idea is generalized to facilitate feedback for principally any DSML that is developed with the RMT approach, using model-driven development. This opens up the possibility to develop and research different simulation semantics or modes of simulation for these DSML.

The technical realization of our approach relies on RENEW as a simulation backend and the graphical infrastructure provided by the RMT framework. For the provision of graphical feedback in the execution of DSML, the framework is extended with a re-engineered transformation process. This process enables the backpropagation of events from the Petri net simulation and generic classes for graphical instance models that can change the representation depending on the simulation events and state.

The Petri net model displayed in Fig. 8 provides an overview of our solution to implementing graphical feedback for executable DSML. It represents the technical implementation of the compile and feedback steps from the abstract model depicted in Fig. 7. This involves the *Transformation* process (compile), the RENEW *Simulator* and the *Graphical Representation* (feedback) of the DSML model and model instances. A *graphical DSML model* created by a domain practitioner using a domain-specific tool (initially marked place in the *Graphical Representation* box) is the starting point for the simulation. When the simulation is initialized, executable Petri net models are generated through transformation,

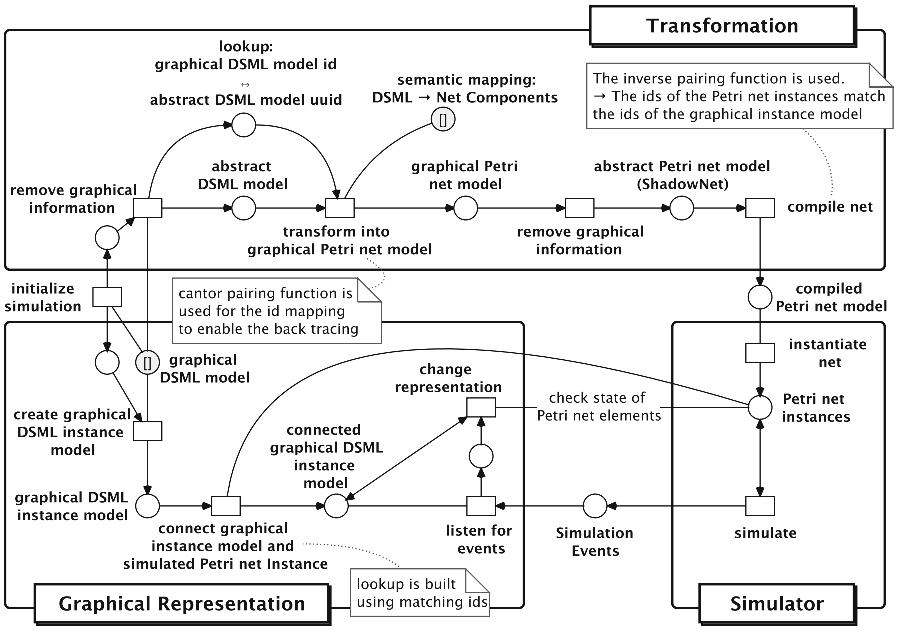

Transformation
lookup: graphical DSML model id ↔ abstract DSML model uuid
semantic mapping: DSML → Net Components
The inverse pairing function is used. → The ids of the Petri net instances match the ids of the graphical instance model
remove graphical information
abstract DSML model
graphical Petri net model
abstract Petri net model (ShadowNet)
compile net
transform into graphical Petri net model
remove graphical information
initialize simulation
cantor pairing function is used for the id mapping to enable the back tracing
compiled Petri net model
instantiate net
graphical DSML model
change representation
check state of Petri net elements
Petri net instances
create graphical DSML instance model
connected graphical DSML instance model
graphical DSML instance model
connect graphical instance model and simulated Petri net Instance
listen for events
Simulation Events
simulate
lookup is built using matching ids
Graphical Representation
Simulator

Fig. 8. Transformation and simulation process implementing the graphical feedback.

and the graphical representation classes are instantiated and linked with the simulated Petri nets. The connection between the graphical and simulated elements is established by matching their respective ids.

A prerequisite for the transformation is the provision of a semantic mapping like the one described in Sect. 4 (initially marked place *semantic mapping: DSML → Net Components*). The mapping has to be provided by the DSML developer, in the form of a referrer to the net components repository. Without additional configuration, the constructs are mapped to correspondingly named net components. This is a convenience feature to allow early testing with minimal configuration overhead. Many of the optional configuration parameters follow the *convention over configuration* principle. Alternatively, a semantic model can be used to configure, which construct is mapped to which net component (analogous to the tool configuration or stylesheet model described in Sect. 3).

The traceability of the constructs to properly facilitate the relation between the simulated elements and the graphical representations is a critical aspect of the transformation process. The first step converts the graphical representation of the DSML model into an abstract representation while a lookup is provided that enables the discovery of graphical elements for the abstract ones. Since the ids used in the graphical models are not persistent, the abstract DSML models use UUIDs.

Both, the abstract model and the lookup are combined with the provided semantic mapping to create a graphical Petri net model. During the development process, the DSML developer may want to iteratively examine the generated Petri nets, to find out if the result matches the expectations. The generation of graphical models offers this possibility, which is the reason to not directly generate non-graphical Petri nets.

In order to enable the back tracing from Petri net components to graphical DSML components, the Cantor pairing function is used to calculate the ids for the generated Petri net. This is a necessity since graphical Renew models use plain integer ids. Before the graphical Petri net can be simulated, the graphical information is removed, a *ShadowNet* is created and then subsequently compiled. In comparison to the graphical model, the compiled net elements may have multi-part ids. Thus, the compiling step uses the inverse pairing function in order to match the ids of the Petri net instance's elements and the graphical constructs of the DSML instance models.

The compiled Petri net model can then be instantiated and simulated. Many efforts in preparing previous releases of Renew were invested into the decoupling of the graphical interface and the simulator. As a result of this, it is possible to implement principally any graphical representation. The simulator provides a remote interface through which it is possible to listen for simulation events and check for the simulation state. The simulator sends an event when a transition starts or stops firing, or a marking changes, allowing listeners to react to these events (e.g., by changing the representation). In this implementation, the remote interface is used by the components of the graphical representation to enable the graphical feedback for the simulation.

While the graphical DSML model is transformed into an executable Petri net, a graphical DSML instance model is also created. As soon as the Petri net instance is available from the simulator, it is connected with the graphical instance model. In this step, the graphical DSML constructs are connected to the simulated Petri net elements based on matching ids. The graphical DSML instance model, in turn, listens for simulation events that are related to the connected Petri net instance. When such an event occurs, the representation of the graphical DSML model is changed accordingly. The state-dependent representations may be arbitrarily implemented in Java by customizing the classes for the instance model. However, this requires knowledge about programming with Java and the details of the representation classes. In the following section, we propose three variants for the declarative description of graphical representations for the simulation of DSML.

6 Model-Based Configuration of the Representation

Two tasks summarize the specification of the representation for DSML models during execution, based on the state of the underlying simulated Petri net in a declarative fashion: The language developer has to specify the connection between the simulation state and the representation, in order to specify on which simulation event the representation of the constructs should be changed. Furthermore, the visualization of the highlighted constructs has to be defined, i.e., how the representation should be altered or extended to represent a specific simulation state.

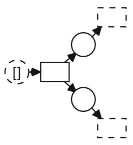
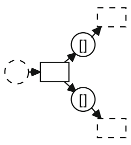
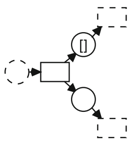
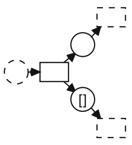
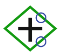
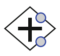
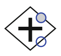
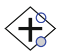
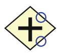
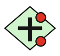
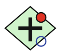
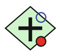
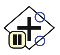
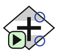
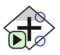
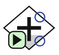
6.1 Trade-Off Between Expressiveness and Simplicity

We identified some requirements that need to be fulfilled in order to adequately support the developer of a modeling language in these two tasks. Generally, there are two requirements: expressiveness and simplicity. First, the developer must be able to implement the desired result, i.e., has to be able to specify the exact representation of the executed model. Secondly, the implementation and configuration overhead should be minimal. Multiple factors have an impact on the simplicity. The connection between simulator and representation should be specifiable without knowledge of the internals of the simulator and optimally without programming skills. It should be possible to use the same tools as used for the representation of the static constructs to provide representations for highlighted constructs. Often the highlighted representation only minimally differs from the original representation (e.g., by having a different color). It should be possible to specify these slight variations without the requirement to provide multiple copies of the same figure. This is especially important for Petri net components with a high degree of concurrency. These may result in a large number of global states and thus a large number of different representations.

There is a trade-off between expressiveness and simplicity. Based on the identified requirements, we present three variants for realizing model-based simulation highlighting, each with a different focus regarding expressiveness and simplicity. The specification of the connection between simulation and representation may be achieved with an annotation language for the semantic components, which can be used for all of the three variants. For the provision of the altered representations, we propose the strategies simple, style sheet-based, and graphical component-based. With the simple highlighting method, the generic highlighting mechanism of Renew is used. The style sheet-based mechanism uses style sheets analogously to the specification of the concrete syntax for the DSML constructs. With the annotated template-based highlighting, the representation of the highlighted constructs is provided by drawings that are created with the graphical editor of Renew.

These three variants are exemplarily illustrated for the parallel gateway in Table 2. The table shows some representations that can be achieved with the different highlighting variants. The semantic component of the parallel gateway may have four different states that are depicted in the first row of the table. With the true concurrency simulation of Renew there are actually more states, but these are omitted here and can be handled analogously (with the consideration of the state of a firing transition new states of the component emerge). The second row of the table contains the highlighted constructs for the simple highlighting strategy. Exemplary highlighting that can be achieved using the style sheet-based strategy is depicted in the third row. The fourth row shows state illustrations that can be produced with the graphical component-based strategy.

Table 2. Highlighting variants and example representations.

	activatable	activeboth	active1	active2
Semantic Component State				
simple				
stylesheet-based				
graphical component-based				

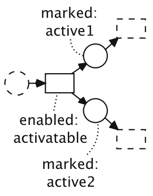
Before the three variants for *defining the visualization of the highlighted constructs* are described in detail, the basics for *specifying the connection between the simulation state and the representation* are presented in the following section.

6.2 Relating Simulation State and Representation

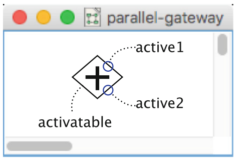
To obtain graphical feedback in the simulated DSML, one task defines the connection between the simulation events and the graphical representation. Since the underlying executable is a Petri net, the simulation events are net events. In the graphical simulation environment of Renew it is possible to respond to mainly two types of events: the firing of a transition (with start and end of the firing) and the change of a marking of a place. Additionally, it is possible to check the state of a single net element such as the marking of a place and whether a transition is enabled or firing.

The graphical elements in the representation of the running model are linked to the simulator via the ids of the net elements in the sense that a net element in the simulation (a place or a transition) has exactly one connected graphical component that observes the net element and listens to its events. One graphical component may observe multiple net elements (this is a result of the `1:n` mapping).

Depending on the DSML and the semantics, there are multiple possibilities to link simulation events and representations. An essential question in this context is, whether the classifiers and relations reflect the state of the places or the activities of the transitions. This depends mainly on the character of the DSML. For a language focusing on the behavior of a system (such as activity diagrams), it probably makes sense to target the transition events. A classifier, for example,



(a) semantic component



(b) graphical component

Fig. 9. Artifacts for simple highlighting. (Color figure online)

could be highlighted when one of the transitions in the corresponding semantic component is firing. A concentration on the place markings is more useful for a language with a strong state focus (such as state diagrams) where a classifier would be highlighted when a place in the semantic component is marked. Many languages (including BPMN) have a hybrid character, requiring different behaviors for different constructs.

Our approach facilitates the implementation of these hybrid languages in a simple way. The DSML developer can define the highlighting behavior for each DSML construct by using an annotation language that we present exemplarily for the parallel gateway. Figure 9a depicts the semantic component of the parallel gateway (as shown in Table 2) with annotations. For illustration purposes in this contribution, the connections between the annotation texts and the corresponding net elements (places and transitions) are presented as dotted lines. In our implementation in the Renew environment, these annotations are net element inscriptions, which can be added with a specific text tool. These inscriptions resp. annotation texts are stored as attributes of each net element. The graphical interface can retrieve these attributes during simulation in order to determine the highlighting of the corresponding DSML's graphical components. An annotation contains two parts divided by a colon. The first part represents the simulation state or a simulation event for the particular net element, and the second part is a state concerning the whole BPMN component. In this contribution we name the first part *element state/event*, as it describes the state/event of the single element (a place or transition), and the second part *component state/event*, as it describes the state/event of the entire construct. For example, the annotation `marked:active1` in Fig. 9a indicates that the parallel gateway is in the state `active1` whenever the upper place contains a token. These component related states are not disjoint because the upper and the lower place may be marked at the same time, which results in the construct being in the state `active1` and `active2`. These states can now be used to specify the representation of the highlighted components.

6.3 Simple Highlighting

The simple highlighting strategy uses Renew's generic highlighting mechanism where a color change highlights constructs and parts of constructs. This is already applied for simulation feedback of Petri net simulations. The mechanism reacts to two states of a net element: `enabled` and `marked`. For net elements that are in the state `enabled` the visual attribute *enabled* is applied. Thus, the net elements are presented with a green border. The visual attribute *highlighted* is applied to net elements in the state `marked`. The net elements therefor receive a change of their background color. We neglect the state `firing` of a transition for simplification purposes in this contribution. The color selection takes the original color of a figure into account to ensure that the color change is noticeable. This highlighting mechanism is also suitable for figures that are utilized in the Rmt approach. For example, the highlighted representations of the parallel gateway construct, which can be achieved with the simple highlighting strategy, are included in the second row of Table 2.

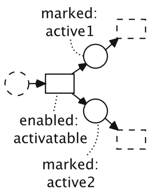
Figure 9 shows the artifacts that are required to achieve the result in Table 2. In this case, the annotations correspond to the component states defined within the semantic component (`activatable`, `active1` and `active2`). Together with the concrete syntax the highlighting information can be provided with a drawing that is created with Renew – the graphical component. The graphical component is extended with annotations for the graphical elements (these annotations can be created analogously to the annotations for net elements as described in Sect. 6.2). The annotation `activatable` is connected to the diamond element of the gateway construct. This results in a green coloring of the border when the gateway component is in the state `activatable`, because of the usage of Renew's generic mechanism the style class *enabled* is applied. Analogously, the two circles representing the ports receive a gray background in the states `active1` or `active2`.

Of the three strategies, this is the easiest to implement for the DSML developer, but in return, it is limited in the sense that it is not possible to customize the highlighted representation.

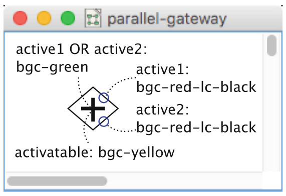
6.4 Stylesheet-Based Highlighting

The problem with the simple highlighting strategy is that highlighting is limited to a change of colors, which are not selectable. The style sheet-based highlighting makes it possible to create custom style classes and, e.g., specify the highlighting color and other style parameters (such as line style/shape, arrow shape). Many attributes are predefined. With this strategy, representations like the ones depicted in the third row of Table 2 become possible.

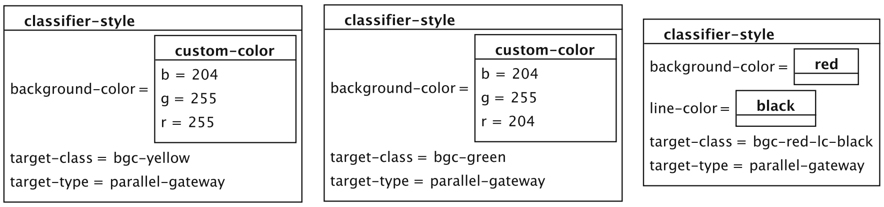
Figure 10 depicts the artifacts required to achieve this variant of highlighting. Similar to the simple highlighting the DSML developer has to annotate the graphical component in order to specify which part of the figure to style (see Fig. 10b). However, the annotation syntax is extended with boolean expressions over the component states (`active1 OR active2`) and a style class that will be applied to the respective figure (e.g., `bgc-green`). In this way, the DSML developer is free to create custom styles according to her/his requirements. The definitions for the custom styles are depicted in Fig. 10c.



(a) semantic component



(b) graphical component



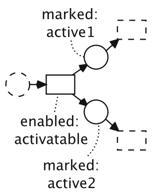
(c) stylesheets

Fig. 10. Artifacts for stylesheet-based highlighting. (Color figure online)

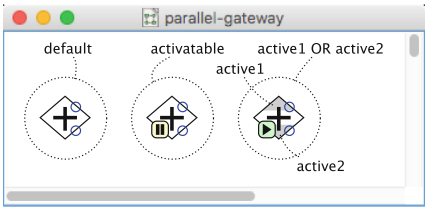
This variant provides more flexibility than the simple highlighting since the graphical representations may be customized with style sheets, but it requires more effort since these style sheets need to be explicitly defined. The style sheets-based highlighting is especially useful if the representation of the static models is already defined with style sheets. The amount of flexibility is still limited since the possibilities for customization are restricted to Renew's predefined visual attributes.

6.5 Graphical Component-Based Highlighting

Sometimes the requirements for the highlighted representations exceed the possibilities of predefined style attributes, such as in the last row of Table 2 where the highlighted constructs are extended with additional graphical objects or images. In this example the constructs are amended with a *pause* symbol to represent the



(a) semantic component



(b) graphical components

Fig. 11. Artifacts for graphical component-based highlighting.

activatable and a *play* symbol for the activated state. Additionally, the marking is represented via partial gray background coloring.

This form of highlighting requires the possibility to provide individual representations for each of these states. With the graphical component-based highlighting, it is possible to draw the representations directly within the Renew editor. The graphical representations in Fig. 11 are annotated similar to the examples of the previous strategies. The syntax of the annotations is slightly different since each state does not need to be mapped to a style sheet class. This way, the graphical representation for the animation is graphically defined instead of being declaratively described by a textual description. Only the graphical component with annotations that matches the current state is displayed in the simulation. The `default` keyword refers to the fallback state with no other state.

To prevent the effects of a state space explosion for semantic components with concurrency, we provide a mechanism to specify multiple representations in one. As depicted in Fig. 11b the graphical component to the right serves as the graphical definition for all of the three states `activeboth`, `active1` and `active2` from Table 2.

Compared to the other two variants, this is the most flexible one when it comes to visualization. However, providing only the graphical component-based highlighting is not optimal because it requires a lot of modeling effort for a simple style or color change. In combination, the three presented highlighting strategies provide the possibility of quickly implementing graphical feedback for a DSML by selecting the most suitable variant depending on the requirements.

7 Simulation of the BPMN Example

Figure 12 shows a snapshot from the simulation of a BPMN model. The topmost part shows Renew's main window with context menus, editor tool bars and the status bar. The two overlapping windows in the middle are the template (white background) and an instance (light blue background) of a BPMN process. The template drawing was modeled using the constructs from the BPMN toolbar.

The window on the right side contains an instance of this model and was created from that template. It represents the simulation state using the simple highlighting strategy. The simulation is paused in a state where the sequence flow and the conditional gateway at the bottom of the instance window are activated, which is reflected by the red color of the sequence flow and the gray background of the gateway. This state corresponds to the Petri net in the lowermost part of Fig. 12. In this state, the task at the top and the two sequence flows behind the conditional gateway (in conflict) are activatable, which is again reflected with the green coloring. The subsequent simulation step may be invoked by right-clicking on the activatable task figure or on one of the sequence flow connections in the BPMN model instance. All actions and executions are performed by the underlying Petri nets, which therefore determine the semantics of the domain specific language model while the interaction of the user is performed through the BPMN model. The behavior may be customized by providing

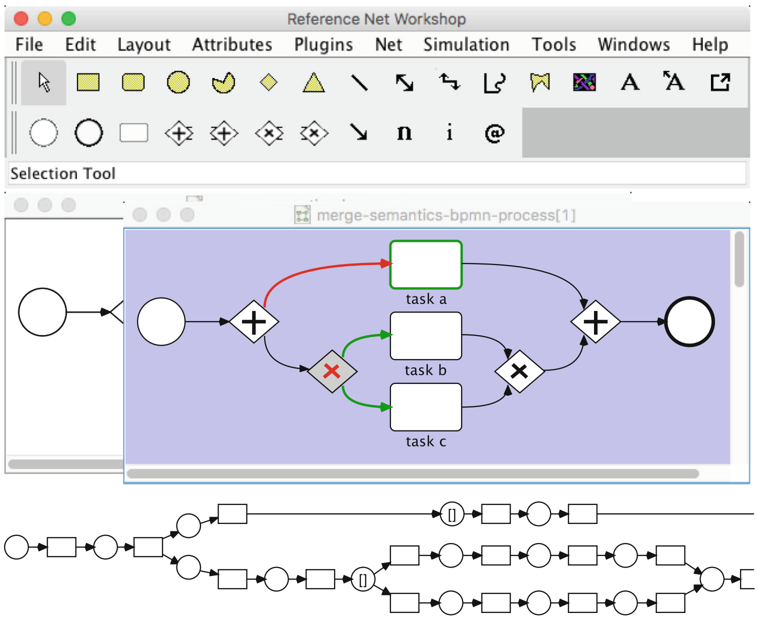


Fig. 12. Running BPMN simulation and the corresponding Petri net. (Color figure online)

alternative net components that may contain colored tokens, variables, inscriptions, synchronous channels, etc. The GUI interaction is provided with the RMT integration.

The Petri net in the lowermost part of the figure is the representation of the simulated net instance, which was generated using the semantic mapping from Sect. 4 (cf. Fig. 5). For the presentation in this paper, the Petri net model was created by hand. The generated Petri net that performs the simulation has no visual representation at all. This is a design decision in order to maintain the ability to execute these models without graphical feedback in server mode, which is essential to building large scale applications from net models. In all, this facilitates the provision of graphical feedback in the BPMN model by reflecting simulation events from the simulated (invisible) Petri net to the above layer.

8 Conclusion

In this contribution, we present a concept for providing simulation feedback for DSML that are developed with the RMT approach on the basis of meta-models and transformational semantics using Petri nets. Based on our new feature for direct visual feedback in the simulated domain specific model, we provide an improved experimentation environment for interactively experiencing with the behavior of newly designed domain specific languages without additional work on animating the models. The central part of our contribution consists of three alternative mechanisms for providing graphical feedback in the simulation of DSML: simple, style sheet-based, and component-based. Particularly the simple variant inherits functionality from Renew, but each of the presented concepts for highlighting should be transferable to other approaches and frameworks. In order to demonstrate the practicability of our approach, we present the integrated simulation of a selected subset of BPMN and refer to a straight-forward model transformation to Petri nets. The presented mechanisms do not cover every modeling technique, nor do they claim to be complete in any sense. Instead, they demonstrate a flexible concept, which allows customization for many use cases that is easily applicable without a lot of configuration overhead.

In the future, we will benefit from the presented conceptual approach by conceptualizing the transformation and variations of the target language. A `1:n` mapping from DSML constructs to Place/Transition nets would allow the analysis of the executable DSML. The choice of a more expressive formalism for the definition of semantics would, for example, make it possible to cover a larger part of the BPMN standard.

In the context of Renew, Reference Nets may be applied as a target formalism, which benefits from powerful modeling capabilities, Java integration, the underlying concurrency theory, and the Renew integrated development and simulation environment. The proposed transformation to a powerful (Turing complete) formalism is attractive on the one hand because the mentioned advantages of this formalism may be exploited. On the other hand, the possibilities to perform formal analysis are restricted due to the complexity of the formalism. The flexibility concerning the formalisms opens up the possibility of applying a whole array of methods from low-level analysis – e.g., using Renew's integration of LoLA[2] [16] – to normal software engineering validation like unit testing [39].

A graphical representation of analysis results that originate from the underlying Petri net in the DSML requires a connection to the applied analysis tools. When the graphical representation changes based on results of analysis tools instead of the simulation events, the visualization of structural properties, such as invariants, conflicts, concurrency, mutual exclusion, siphons, or traps becomes possible. In the future we plan to extend Renew's capabilities for the analysis of domain-specific models and improved tool connectivity.

References

1. Abrahão, S., et al.: User experience for model-driven engineering: challenges and future directions. In: 20th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems, MODELS 2017, Austin, TX, USA, 17–22 September 2017, pp. 229–236. IEEE Computer Society (2017). https://doi.org/10.1109/MODELS.2017.5
2. Biermann, E., Ehrig, K., Ermel, C., Hurrelmann, J.: Generation of simulation views for domain specific modeling languages based on the Eclipse modeling framework. In: 2009 IEEE/ACM International Conference on Automated Software Engineering, pp. 625–629, November 2009. https://doi.org/10.1109/ASE.2009.46

[2] LoLA: Low-Level Analyzer: http://www.service-technology.org/lola/.

3. Bousse, E., Degueule, T., Vojtisek, D., Mayerhofer, T., DeAntoni, J., Combemale, B.: Execution framework of the GEMOC studio (tool demo). In: van der Storm, T., Balland, E., Varró, D. (eds.) Proceedings of the 2016 ACM SIGPLAN International Conference on Software Language Engineering, Amsterdam, The Netherlands, 31 October–1 November 2016, pp. 84–89. ACM (2016), http://dl.acm.org/citation.cfm?id=2997384
4. Bousse, E., Leroy, D., Combemale, B., Wimmer, M., Baudry, B.: Omniscient debugging for executable DSLs. J. Syst. Softw. **137**, 261–288 (2017). https://hal.inria.fr/hal-01662336
5. Bryant, B.R., Gray, J., Mernik, M., Clarke, P.J., France, R.B., Karsai, G.: Challenges and directions in formalizing the semantics of modeling languages. Comput. Sci. Inf. Syst. **8**(2), 225–253 (2011). https://doi.org/10.2298/CSIS110114012B
6. Cabac, L.: Modeling Petri Net-Based Multi-Agent Applications, Agent Technology - Theory and Applications, vol. 5. Logos Verlag, Berlin (2010). http://www.logos-verlag.de/cgi-bin/engbuchmid?isbn=2673&lng=eng&id=. http://www.sub.uni-hamburg.de/opus/volltexte/2010/4666/
7. Cabac, L., Haustermann, M., Mosteller, D.: Renew 2.5 – towards a comprehensive integrated development environment for Petri net-based applications. In: Kordon, F., Moldt, D. (eds.) PETRI NETS 2016. LNCS, vol. 9698, pp. 101–112. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-39086-4_7
8. Cabac, L., Haustermann, M., Mosteller, D.: Software development with Petri nets and agents: approach, frameworks and tool set. Sci. Comput. Program. **157**, 56–70 (2018). https://doi.org/10.1016/j.scico.2017.12.003
9. Christensen, S., Damgaard Hansen, N.: Coloured Petri nets extended with channels for synchronous communication. In: Valette, R. (ed.) ICATPN 1994. LNCS, vol. 815, pp. 159–178. Springer, Heidelberg (1994). https://doi.org/10.1007/3-540-58152-9_10
10. Corradini, F., Muzi, C., Re, B., Rossi, L., Tiezzi, F.: Animating multiple instances in BPMN collaborations: from formal semantics to tool support. In: Weske, M., Montali, M., Weber, I., vom Brocke, J. (eds.) BPM 2018. LNCS, vol. 11080, pp. 83–101. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-98648-7_6
11. Cramer, B., Kastens, U.: Animation automatically generated from simulation specifications. In: 2009 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC), pp. 157–164, September 2009. https://doi.org/10.1109/VLHCC.2009.5295274
12. Dijkman, R.M., Dumas, M., Ouyang, C.: Semantics and analysis of business process models in BPMN. Inf. Softw. Technol. **50**(12), 1281–1294 (2008). https://doi.org/10.1016/j.infsof.2008.02.006
13. Eclipse Foundation Inc.: Eclipse Modeling Framework (EMF) (2018). https://www.eclipse.org/modeling/emf/. Accessed 24 May 2018
14. Gorp, P.V., Dijkman, R.M.: A visual token-based formalization of BPMN 2.0 based on in-place transformations. Inf. Softw. Technol. **55**(2), 365–394 (2013). https://doi.org/10.1016/j.infsof.2012.08.014
15. Hegedüs, Áb., Ráth, I., Varró, D.: Replaying execution trace models for dynamic modeling languages. Period. Polytech. Electr. Eng. Comput. Sci. **56**(3), 71–82 (2012). https://pp.bme.hu/eecs/article/view/7078
16. Hewelt, M., Wagner, T., Cabac, L.: Integrating verification into the PAOSE approach. In: Duvigneau, M., Moldt, D., Hiraishi, K. (eds.) Petri Nets and Software Engineering. International Workshop PNSE 2011, Newcastle upon Tyne, UK, June 2011. Proceedings. CEUR Workshop Proceedings, vol. 723, pp. 124–135. CEUR-WS.org, June 2011. http://ceur-ws.org/Vol-723/

17. Huber, P., Jensen, K., Shapiro, R.M.: Hierarchies in coloured Petri nets. In: Rozenberg, G. (ed.) ICATPN 1989. LNCS, vol. 483, pp. 313–341. Springer, Heidelberg (1991). https://doi.org/10.1007/3-540-53863-1_30
18. The GEMOC Initiative: The GEMOC initiative - breathe life into your designer! model simulation, animation and debugging with sirius animator, part of the GEMOC studio. http://gemoc.org/breathe-life-into-your-designer. Accessed 9 Oct 2018
19. Kheldoun, A., Barkaoui, K., Ioualalen, M.: Formal verification of complex business processes based on high-level Petri nets. Inf. Sci. **385**, 39–54 (2017). https://doi.org/10.1016/j.ins.2016.12.044
20. Kindler, E.: ePNK applications and annotations: a simulator for YAWL nets. In: Khomenko, V., Roux, O.H. (eds.) PETRI NETS 2018. LNCS, vol. 10877, pp. 339–350. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-91268-4_17
21. Kleppe, A.: Software Language Engineering: Creating Domain-Specific Languages Using Metamodels. Pearson Education, London (2008). https://dl.acm.org/citation.cfm?id=1496375
22. Kolovos, D.S., Paige, R.F., Polack, F.A.C.: The epsilon transformation language. In: Vallecillo, A., Gray, J., Pierantonio, A. (eds.) ICMT 2008. LNCS, vol. 5063, pp. 46–60. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-69927-9_4
23. Kummer, O.: Referenznetze. Logos Verlag, Berlin (2002). http://www.logos-verlag.de/cgi-bin/engbuchmid?isbn=0035&lng=eng&id=
24. Kummer, O., Wienberg, F., Duvigneau, M., Cabac, L., Haustermann, M., Mosteller, D.: Renew - the Reference Net Workshop, June 2016. http://www.renew.de/. release 2.5
25. Kummer, O., Wienberg, F., Duvigneau, M., Cabac, L., Haustermann, M., Mosteller, D.: Renew - User Guide (Release 2.5). University of Hamburg, Faculty of Informatics, Theoretical Foundations Group, Hamburg, June 2016
26. de Lara, J., Vangheluwe, H., Alfonseca, M.: Meta-modelling and graph grammars for multi-paradigm modelling in atom3. Softw. Syst. Model. **3**(3), 194–209 (2004). https://doi.org/10.1007/s10270-003-0047-5
27. Lohmann, N., Verbeek, E., Dijkman, R.M.: Petri net transformations for business processes - a survey. Trans. Petri Nets Other Models Concurr. **2**, 46–63 (2009)
28. Mayerhofer, T., Combemale, B.: The tool generation challenge for executable domain-specific modeling languages. In: Seidl, M., Zschaler, S. (eds.) STAF 2017. LNCS, vol. 10748, pp. 193–199. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-74730-9_18
29. Möller, P., Haustermann, M., Mosteller, D., Schmitz, D.: Simulating multiple formalisms concurrently based on reference nets. In: Moldt, D., Cabac, L., Rölke, H. (eds.) Petri Nets and Software Engineering. International Workshop, PNSE 2017, Zaragoza, Spain, 25–26 June 2017. Proceedings. CEUR Workshop Proceedings, vol. 1846, pp. 137–156. CEUR-WS.org (2017). http://CEUR-WS.org/Vol-1846/
30. Möller, P., Haustermann, M., Mosteller, D., Schmitz, D.: Model synchronization and concurrent simulation of multiple formalisms based on reference nets. Trans. Petri Nets Other Models Concurr. XIII **13**, 93–115 (2018). https://doi.org/10.1007/978-3-662-58381-4_5
31. Mosteller, D., Cabac, L., Haustermann, M.: Integrating Petri net semantics in a model-driven approach: the Renew meta-modeling and transformation framework. Trans. Petri Nets Other Models Concurr. XI **11**, 92–113 (2016). https://doi.org/10.1007/978-3-662-53401-4_5

32. Mosteller, D., Haustermann, M., Moldt, D., Schmitz, D.: Graphical simulation feedback in Petri net-based domain-specific languages within a meta-modeling environment. In: Moldt, D., Kindler, E., Rölke, H. (eds.) Petri Nets and Software Engineering. International Workshop, PNSE 2018, Bratislava, Slovakia, 25–26 June 2018. Proceedings. CEUR Workshop Proceedings, vol. 2138, pp. 56–75. CEUR-WS.org (2018). http://ceur-ws.org/Vol-2138/
33. OMG: Object Management Group: Business Process Model and Notation (BPMN) - Version 2.0.2 (2013). http://www.omg.org/spec/BPMN/2.0.2
34. Pedro, L., Lucio, L., Buchs, D.: System prototype and verification using metamodel-based transformations. IEEE Distrib. Syst. Online **8**(4) (2007). https://doi.org/10.1109/MDSO.2007.22
35. Rybicki, F., Smyth, S., Motika, C., Schulz-Rosengarten, A., von Hanxleden, R.: Interactive model-based compilation continued – incremental hardware synthesis for SCCharts. In: Margaria, T., Steffen, B. (eds.) ISoLA 2016. LNCS, vol. 9953, pp. 150–170. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-47169-3_12
36. Sedrakyan, G., Snoeck, M.: Enriching model execution with feedback to support testing of semantic conformance between models and requirements - design and evaluation of feedback automation architecture. In: Calabrò, A., Lonetti, F., Marchetti, E. (eds.) Proceedings of the International Workshop on domAin specific Model-based AppRoaches to vErificaTion and validaTiOn, AMARETTO@MODELSWARD 2016, Rome, Italy, 19–21 February 2016, pp. 14–22. SciTePress (2016). https://doi.org/10.5220/0005841800140022
37. Valk, R.: Petri nets as token objects. In: Desel, J., Silva, M. (eds.) ICATPN 1998. LNCS, vol. 1420, pp. 1–24. Springer, Heidelberg (1998). https://doi.org/10.1007/3-540-69108-1_1
38. Westergaard, M.: Access/CPN 2.0: a high-level interface to coloured petri net models. In: Kristensen, L.M., Petrucci, L. (eds.) PETRI NETS 2011. LNCS, vol. 6709, pp. 328–337. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-21834-7_19
39. Wincierz, M.: A tool chain for test-driven development of reference net software components in the context of CAPA agents. In: Moldt, D., Cabac, L., Rölke, H. (eds.) Petri Nets and Software Engineering. International Workshop, PNSE 2017, Zaragoza, Spain, 25–26 June 2017. Proceedings. CEUR Workshop Proceedings, vol. 1846, pp. 197–214. CEUR-WS.org (2017). http://CEUR-WS.org/Vol-1846/