Graphical Simulation Feedback in Petri Net-based Domain-Specific Languages within a Meta-Modeling Environment

David Mosteller, Michael Haustermann, Daniel Moldt, and Dennis Schmitz

University of Hamburg, Faculty of Mathematics, Informatics and Natural Sciences,
Department of Informatics, http://www.informatik.uni-hamburg.de/TGI/

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 translational 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 basis to develop 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.

We present a concept for the rapid prototyping and direct simulation (and simulation feedback) of domain specific modeling languages (DSML) within the RENEW simulation environment. The focus of this contribution is on the integrated simulation and graphical feedback of the executed language during simulation. 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 [15] and secondly, the provision and coupling of multiple modeling perspectives during execution within RENEW [14]. The first contribution does not consider the

simulation in the source language, instead 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 previous workshop contribution [16].

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 translational 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 [2, Chapter 5], which are reusable and parametrizable code templates – quite comparable to code macros – modeled as a Petri net.

We choose Reference Nets [11] as a target language but we 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 and 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 with comprehensive development support for the construction of Reference Net-based systems [3, 12].

In this contribution we extend the RMT framework (as presented in Section 2) with direct simulation in the original representation of the DSML and discuss the integration into the approach. The presented concept for simulation visualization bases on 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 Section 3 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 is able to specify the desired representation of the executed models in a model-driven fashion. We discuss multiple alternatives to provide support for this task in the RMT approach in Section 4. As a part of our contribution a generic compiler is implemented in RENEW. On this basis the generated technique may be executed within RENEW's simulation engine in its original representation as presented in Section 5.

2 The RMT Framework

The RMT framework (RENEW Meta-Modeling and Transformation) [15] is a model-driven framework for the agile development of DSML. It follows concepts

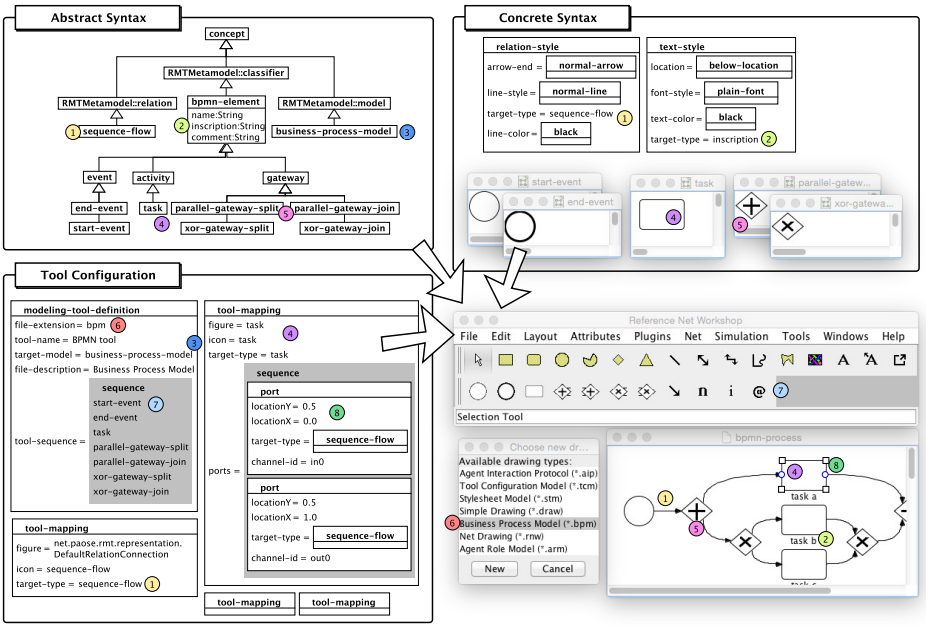


Figure 1: Excerpt from the RMT models for a BPMN prototype with the generated tool.

from software language engineering (SLE, [10]) and enables a short development cycle to be appropriately applied in prototyping environments. The RMT framework is specifically well-suited to develop languages with simulation feedback due to its lightweight approach to SLE and the tight integration with the extensible RENEW simulation engine, which supports the propagation of simulation events. Other frameworks for model-driven engineering (MDE), such as for instance the eclipse modeling framework (EMF, [6]), could also benefit from the proposed solution for the provision of simulation feedback. However, this would require integrating the MDE framework with an equally adequate simulation engine.

With the RMT framework the specification of a language and a corresponding modeling tool may be derived from a set of models that are 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 the repertoire of RENEW's modeling constructs. They are configurable by stylesheets and complemented with icon images 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 for simulation with graphical feedback. 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) for 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 tool bar can be generated from these components. The representation of the inscription annotation (2) and the sequence flow relation (1) is configured with stylesheets.

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

With these models a tool is generated as a Renew plugin as shown at the bottom right side of Figure 1. A complete example of the Rmt models and additional information about framework and approach can be found in [15].

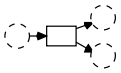
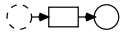
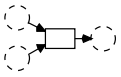
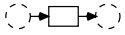
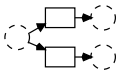
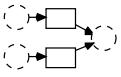
3 Net Component-based Semantics

Our main goal is the provision of an easily applicable approach and readily 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. The Petri net constructs can be created with the Renew editor in a similar way this is done for the graphical components (cf. Figure 1). The net elements are annotated with attributes and arranged together in one environment in order to form a net component.

The operational semantics and the visualization of simulation states are mutually dependent. The development of a semantic mapping should be discussed with respect to the intended visualization and if applicable, interaction. Anyhow, the semantics should not be neglected in favor of a decent visualization. We discuss several variants of defining the semantic mapping and choose the one that is in best conformance with the BPMN standard and at the same time suited for the development of a highlighting for the integrated simulation.

When providing transformational semantics through semantic mappings, a point to consider is the number of mapped constructs in the source and in the 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 our approach utilizes `1:n` mappings. Using net components, which group multiple Reference Net constructs

Table 1: Mapping of BPMN and Petri net constructs [5, p. 7].

BPMN	Reference Net	BPMN	Reference Net
			
			
			
			

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 that we discuss further on in this section.

There are many ways of defining the semantic mapping. Regarding BPMN, the specification itself [17] describes informal semantics for the execution of BPMN processes on the basis of 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. [5, 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 for interpretation and discussion of general concepts of Petri net mappings using semantic components.

One of the main questions is how the components are bordered and how they are connected with each other. This depends mainly on the handling of relations. A simple way to connect two net components is with a simple arc between them, which requires the connection elements in the net components to be of opposing type. Regarding expressiveness, this approach may be 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.

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

Each of the Petri nets from Figure 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 be-

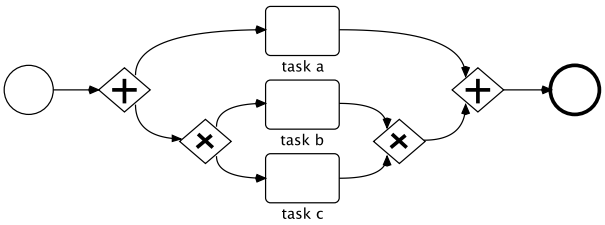


Figure 2: A BPMN process containing three tasks.

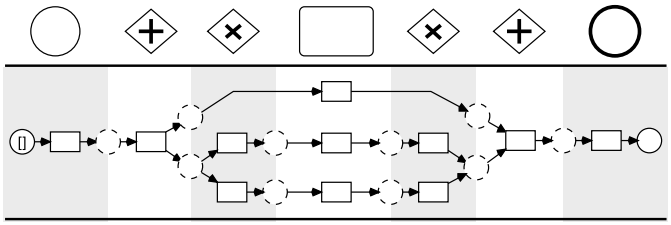


Figure 3: Place bordered decomposition of the BPMN process.

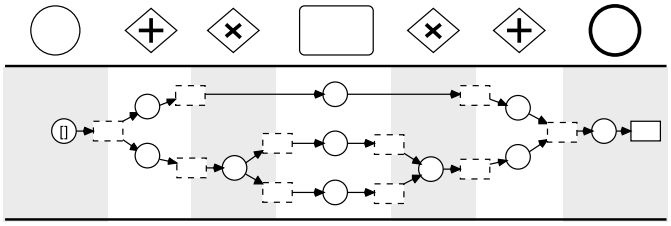


Figure 4: Transition bordered decomposition of the BPMN process.

tween components (highlighted by dashed lines). The first Petri net implements the original mapping from Dijkman et al. [5]. 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 in this sense 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 be eventually removed from the process [17, p. 25], so the variant from Figure 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 it has more the character of 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 basically begins with a state (Inactive) and ends with a state (Closed) [17, 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 holds for the relations, in case it becomes desirable to attach semantics to the relations in a similar way this is done for the constructs.

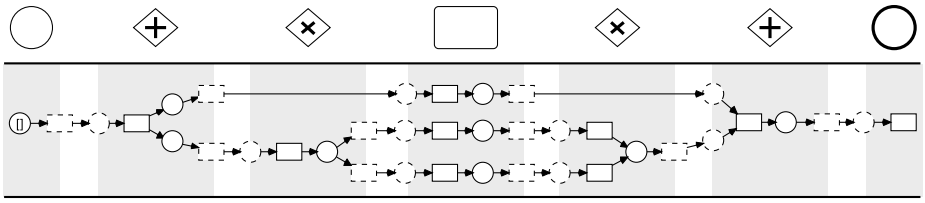


Figure 5: A Petri net mapping of the BPMN process that consists of alternating components.

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 and they 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 Figure 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 neglected. The alternating semantics have a different advantage for defining the highlighting in Section 4. 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.

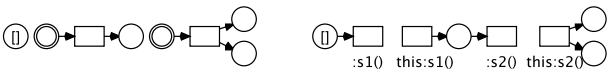
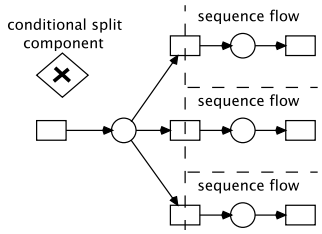


Figure 6: Connection semantics of Reference Nets.

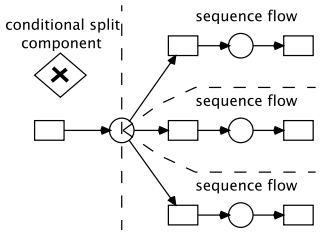
With Reference Nets additional variants become possible such as the connection via virtual places or synchronous channels. Displayed in Figure 6 are on the

left side virtual places. The double-edged place figures are virtual copies of their respective counterpart. These could be easily 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 simply move information along the edges, synchronous channels provide the 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 pools, sub processes, etc. However, this goes beyond the scope of this contribution.

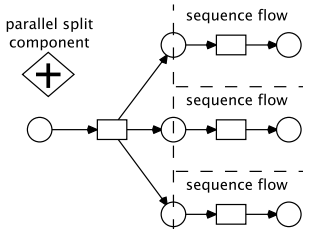
The choice of handling relations and connections leads to the question of the bordering of the net components. A uniform bordering for all classifiers is desirable because different bordering among the classifiers has a negative effect on developers and users regarding the complexity of the language. Furthermore, for each construct of the DSML the bordering for the related net components should not depend on the context, e.g. whether a sequence flow construct precedes or follows a gateway or how many outgoing relations a gateway has. Depending on the connection possibilities within the DSML, a net component for a classifier may have multiple input and output elements whose type affects which connections are possible. In BPMN the gateways are allowed to have an arbitrary



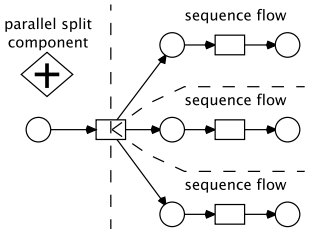
(a) Conditional split component with transition output elements



(b) Conditional split component with place output elements



(c) Parallel split component with place output elements

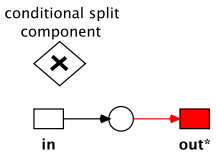


(d) Parallel split component with transition output elements

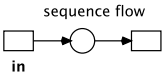
Figure 7: Conditional and parallel split net components with connected sequence flow.

number of connected sequence flow relations where the conditions for conditional gateways are directly attached to them. For a predefined number of choices or parallel paths the components are easy to build with either of the bordering variants. With an arbitrary number of connections, a uniform bordering is no longer possible. Figure 7 shows the bordering variants of the conditional and parallel gateway where the dashed line divides the net component for the gateway and the component for the sequence flow. The connection is again realized by fusion of the elements that are crossed by this line. Figure 7a shows the conditional split with transitions as output elements where the net component for the gateway depends on the number of outgoing connections. In Figure 7b the component is uniformly independent of the connections. The same holds for the parallel gateway with the complementary elements as shown in Figure 7c and 7d.

The presented example shows that a uniform bordering is not possible. Our solution to this problem are components with static parts that are independent of the connections and a dynamic part, which is duplicated for each connection. The challenge thereby is the integration into the approach because one of its key features is that language developers have a simple way to provide their semantics by using the graphical Renew editor and to draw and test the semantics directly within the environment. Therefore, a way to draw these components with variable parts is required. Figure 8a depicts the net component for the conditional split. The red part is duplicated for each connected sequence flow as displayed in Figure 8b. The variable part is specified by the name inscription terminated with the $*$, where the element with this inscription and all the connected arcs are duplicated. With this concept of variable components, a uniform bordering is possible for the presented subset of BPMN.



(a) Conditional split component with a variable part



(b) Sequence flow component

Figure 8: Conditional split net component with variable part and transition border.

4 DSML Tools with Graphical Simulation Feedback

Up to now there was no (sufficient) user feedback when executing models of the generated modeling technique. In this paper we address the extension of our

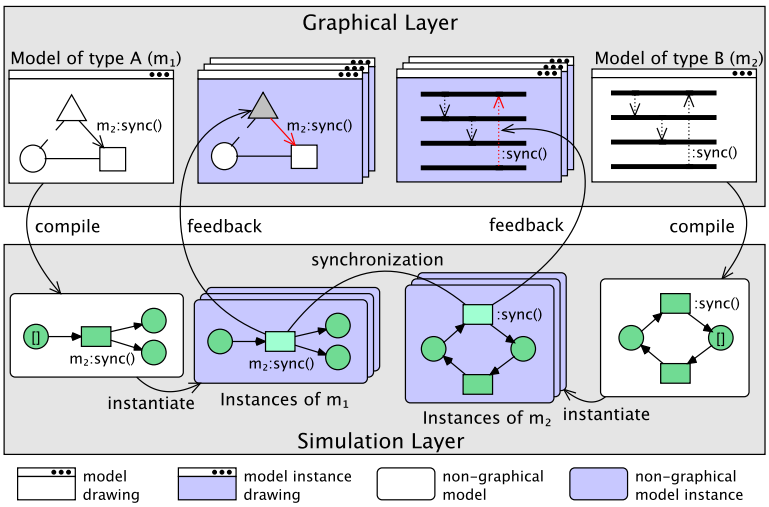


Figure 9: Conceptual model of the model synchronization from [14, p. 8].

framework to allow feedback from the simulation of 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 an adaptive feedback individually depending on the translational semantics for each generated modeling technique.

A conceptual image from the simulation of two modeling techniques that interact with each other is displayed in Figure 9. The image originates from [14, p. 8] where we presented a concept for multi-formalism simulation with the synchronization of multiple modeling techniques on the basis of Reference Nets. The presented solution sketched the idea of providing feedback into a DSML but the realization was specific for a finite automata modeling tool. With our current work this idea is generalized to facilitate feedback into 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 provision of graphical simulation feedback in the original representation of a DSML is summarized by two tasks: 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 certain simulation state.

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. To provide representations for highlighted constructs it should be possible to use the same tools as used for the representation of the static constructs. Often the highlighted representation only minimally differs from the original representation (e.g. by changing a 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, which may result in a large number of global states and thus to a large number of different representations.

There is a trade-off between expressiveness and simplicity. Based on the identified requirements, we present three different 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. For the provision of the altered representations we propose three different strategies: simple, stylesheet-based and graphical component-based. With the simple highlighting method, the generic highlighting mechanism of Renew is used. The stylesheet-based mechanism uses stylesheets 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.

4.1 Relating Simulation State and Representation

In order to obtain graphical feedback in the simulated DSML one task is defining 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 figures 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. The relevant 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, 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 so that it is required to have different behaviors for multiple constructs.

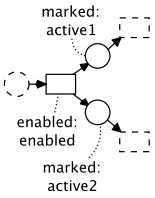


Figure 10: Annotated semantic component for parallel fork gateway.

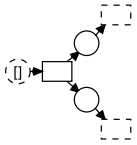
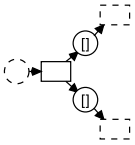
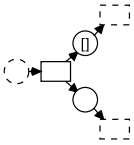
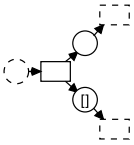
Our approach facilitates the implementation of these hybrid languages in a simple way. The DSML developer is able to define the highlighting behavior for each construct by using an annotation language that we present exemplarily for the construct of the parallel gateway. Figure 10 depicts the semantic component of the parallel gateway (as shown in Table 1) with the annotations. The annotations in this contribution are presented as dotted lines connecting the annotation text with the corresponding element. In the case of our implementation within the Renew environment these annotations are attributes set to the elements. An annotation contains two parts divided by a colon where the first part represents the simulation state or a simulation event for the respective element and the second part is a state concerning the whole component. For example, the annotation `marked:active1` in Figure 10 means 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 component being in state `active1` and `active2`. These states can now be used to specify the representation of the highlighted constructs.

4.2 Simple Highlighting

The simple highlighting strategy uses a generic highlighting mechanism where constructs and parts of constructs are highlighted by a change of the color. 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.

Renew has a generic mechanism for highlighting figures, which is already applied for simulation feedback of Petri net simulations. The highlighting is

Table 2: Highlighting variants and example representations.

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

based on changing the color of figures where it is ensured that the color change is noticeable by choosing a color depending on the original color of the figure. This mechanism is also suitable for figures that are utilized in the RMT approach.

Table 2 exemplarily shows some representations that can be achieved with the different highlighting variants. The semantic component of the parallel gateway may have four different states. With the true concurrency simulation of RENEW there are more states (with firing transitions) but these are omitted here and can be handled analogously. The second row contains the highlighted constructs for the simple highlighting strategy.

Figure 11 shows the artifacts that are required to achieve the result in Table 2. Together with the concrete syntax the highlighting information can be provided with a drawing that is created with RENEW. The graphical component is extended with annotations for the graphical objects. In this case, the annotations correspond to the states defined within the semantic component. The annotation `enabled` is connected to the border of the gateway. This means that the border is highlighted in the `enabled` state, which results in a green color. The two circles representing the ports receive a gray background in the states `active1` or `active2`.

4.3 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 stylesheet-based highlighting makes it possible to specify the highlighting color and other style parameters via stylesheets. With this strategy representations such as the ones depicted in the third row of Table 2 become possible. The DSML developer now has to

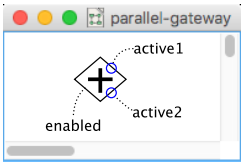


Figure 11: Artifacts for simple highlighting.

specify which part of the figure to style, similar to the simple highlighting, but additionally she/he has to define which style to apply.

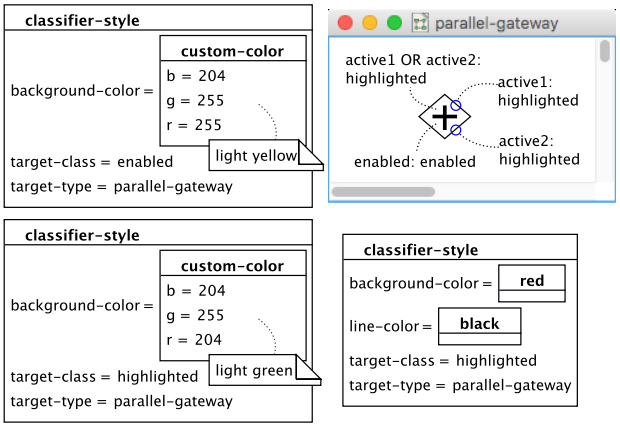


Figure 12: Artifacts for stylesheet-based highlighting.

Figure 12 depicts the artifacts required to achieve this variant of highlighting. The annotation syntax is extended with boolean expressions over the component states (`active1 OR active2`) and a style class that is attached to the respective figure (`highlighted`). Multiple style aspects can be defined with stylesheets. It is possible to set the colors of the background, or the lines, or the shape of lines (dashed, dotted, etc.). Many attributes are pre-defined.

4.4 Graphical Component-based Highlighting

Sometimes the requirements for the highlighted representations exceed the possibilities of pre-defined style attributes, such as in the last row of Table 2 where the highlighted constructs are extended with additional graphical objects. In this example the constructs are amended with a graphical e or a to represent the enabled and activated state and 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 model these representations

directly within the Renew editor. To prevent the effects of a state explosion for semantic components with concurrency, we provide a mechanism to specify multiple representations in one. The graphical representations in Figure 13 are annotated again. In this case only graphical components with annotations matching the current state are displayed in the simulation. This allows specifying the representation of the states `active1` and `active2` in a single component. The `default` keyword refers to the fallback state with no other state.

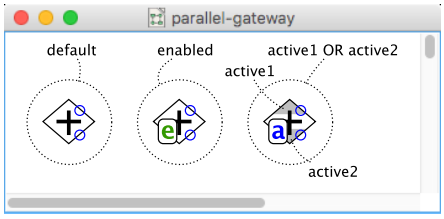


Figure 13: Artifacts for graphical component-based highlighting.

5 Simulation of the BPMN Example

Figure 14 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 (violet 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 and 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 with 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 Figure 14. In this state the task at the top and the two sequence flows behind the xor gateway (in conflict) are activated, which is again reflected with the green coloring. The subsequent simulation step may be invoked by right-clicking on the activated 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 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

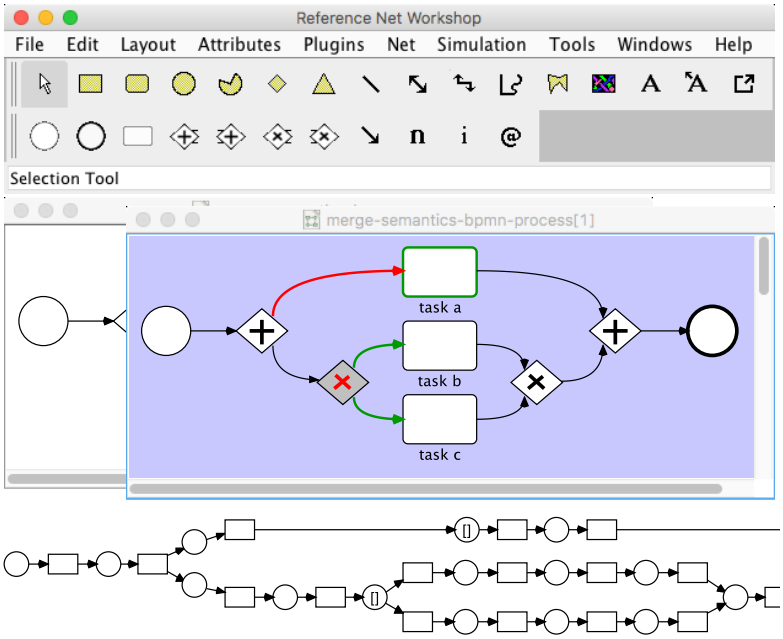


Figure 14: Running BPMN simulation and the corresponding Petri net.

Section 3 (cf. Figure 5). For the presentation in this paper, the Petri net model was created by hand, the generated Petri net that actually performs the simulation has no visual representation at all. This is a design decision 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.

6 Related Work

The utilization of model transformation in order to provide execution or simulation support for different types of models is present in many approaches. Some of them provide simulation feedback in the original representation.

Sedrakyan et al. present a model-driven graphical feedback in a model-to-code transformation with the goal of validating semantic conformance especially for novice modelers [20]. They focus on feedback for errors that occur in the compiled system rather than a complete interactive inspection of the execution.

Rybicki et al. [18] 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. These tracing information are used to propagate runtime information during simulation to the high-level models, which allows to inspect 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.

Kindler has a comparable approach to ours using annotations with the ePNK framework to provide simulation capabilities for a meta-model-based modeling application [9]. 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.

Other approaches aim at providing interactive visual behavior for domain specific modeling tools by using model-driven techniques [1, 4].

Petri nets are often used as target language in transformational approaches in combination with high-level modeling languages. Research questions in the context of transformation into Petri nets are often discussed for specific languages. An overview about semantics for business process languages is for example provided by Lohmann et al. [13]. The utilization of semantic components resulting in a `1:n` mapping is related to the static hierarchy concepts, such as the ones for Coloured Petri Nets [8] and other formalisms. The research results in this area can be transferred to our approach. The capabilities of Reference Nets regarding dynamic hierarchies can be helpful for the development of more complex semantics.

The approach presented in this contribution is unique in the sense that it uses Petri nets as target language as a solid formal basis in combination with a focus on the representation customization of the executed models.

7 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 translational semantics using Petri nets.

In order to demonstrate the practicability of our approach, we present the integrated simulation of a selected subset of BPMN and refer to a straightforward model transformation to Petri nets. Alternative possibilities of defining the translational semantics for process-oriented modeling languages are pointed out, but not exhaustively answered, as this is part of our ongoing research. For this contribution we restrict ourselves to a `1:n` mapping from DSML constructs to Petri net components and introduce the notion of dynamic components to increase the expressiveness, all with the practicability of our approach in mind. The main part of our contribution consists in three alternative mechanisms for the provision of graphical feedback in the simulation of DSML: simple, stylesheet-based and component-based. Especially the simple variant inherits functionality from RENEW, but each of the presented concepts for highlighting should be transferable to other approaches and frameworks. 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 much configuration overhead.

Reference Nets are applied as a target formalism, which benefit 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.

In the future we may benefit from the presented conceptual approach by conceptualizing the transformation and restrictions of the target language, e.g. to Place / Transition nets, to perform analysis. The flexibility with respect to 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 [7, 19] – to normal software engineering validation like unit testing [21]. Based on our new feature for visual feedback directly in the simulated domain specific model we provide an improved experimentation environment to have interactive experiences with the behavior of newly designed domain specific languages without extra work to animate the models.

References

1. 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 (Nov 2009). https://doi.org/10.1109/ASE.2009.46
2. 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/
3. 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.) Application and Theory of Petri Nets and Concurrency - 37th International Conference, PETRI NETS 2016, Toruń, Poland, June 19-24, 2016. Proceedings. Lecture Notes in Computer Science, vol. 9698, pp. 101–112. Springer-Verlag (2016). https://doi.org/10.1007/978-3-319-39086-4_7
4. Combemale, B., Crégut, X., Giacometti, J.P., Michel, P., Pantel, M.: Introducing Simulation and Model Animation in the MDE Topcased Toolkit. In: 4th European Congress EMBEDDED REAL TIME SOFTWARE (ERTS). p. http://www.erts2008.org/. Toulouse, France, France (Jan 2008), https://hal.archives-ouvertes.fr/hal-00371596
5. Dijkman, R.M., Dumas, M., Ouyang, C.: Semantics and analysis of business process models in BPMN. Information and Software Technology **50**(12), 1281–1294 (Nov 2008). https://doi.org/10.1016/j.infsof.2008.02.006
6. Eclipse Foundation, Inc: Eclipse Modeling Framework (EMF) (2018), https://www.eclipse.org/modeling/emf/, accessed on 2018-05-24
7. 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'11, Newcastle upon Tyne, UK, June

2011. Proceedings. CEUR Workshop Proceedings, vol. 723, pp. 124–135. CEUR-WS.org (Jun 2011), http://ceur-ws.org/Vol-723/paper9.pdf
8. Huber, P., Jensen, K., Shapiro, R.M.: Hierarchies in coloured petri nets. In: Rozenberg, G. (ed.) Advances in Petri Nets 1990 [10th International Conference on Applications and Theory of Petri Nets, Bonn, Germany, June 1989, Proceedings]. Lecture Notes in Computer Science, vol. 483, pp. 313–341. Springer (1989). https://doi.org/10.1007/3-540-53863-1_30
9. Kindler, E.: ePNK applications and annotations: A simulator for YAWL nets. In: Application and Theory of Petri Nets and Concurrency - 39th International Conference, PETRI NETS 2018, Bratislava, Slovakia, June 27-29, 2018, Proceedings (2018), to be published
10. Kleppe, A.: Software Language Engineering: Creating Domain-Specific Languages Using Metamodels. Pearson Education (Dec 2008)
11. Kummer, O.: Referenznetze. Logos Verlag, Berlin (2002), http://www.logos-verlag.de/cgi-bin/engbuchmid?isbn=0035&lng=eng&id=
12. Kummer, O., Wienberg, F., Duvigneau, M., Cabac, L., Haustermann, M., Mosteller, D.: Renew – the Reference Net Workshop (Jun 2016), http://www.renew.de/, release 2.5
13. Lohmann, N., Verbeek, E., Dijkman, R.M.: Petri net transformations for business processes - A survey. Trans. Petri Nets and Other Models of Concurrency **2**, 46–63 (2009). https://doi.org/10.1007/978-3-642-00899-3_3
14. 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'17, Zaragoza, Spain, June 25-26, 2017. Proceedings. CEUR Workshop Proceedings, vol. 1846, pp. 137–156. CEUR-WS.org (2017), http://CEUR-WS.org/Vol-1846/
15. Mosteller, D., Cabac, L., Haustermann, M.: Integrating Petri net semantics in a model-driven approach: The Renew meta-modeling and transformation framework. Transaction on Petri Nets and Other Models of Concurrency XI **11**, 92–113 (2016). https://doi.org/10.1007/978-3-662-53401-4_5
16. Mosteller, D., Haustermann, M., Moldt, D.: Prototypical graphical simulation feedback in reference net-based domain-specific languages within a meta-modeling environment. In: Bergenthum, R., Kindler, E. (eds.) Algorithms and Tools for Petri Nets Proceedings of the Workshop AWPN 2017, Kgs. Lyngby, Denmark October 19–20, 2017. pp. 58–63. DTU Compute Technical Report 2017-06 (2017)
17. OMG, Object Management Group: Business Process Model and Notation (BPMN) – Version 2.0.2 (2013), http://www.omg.org/spec/BPMN/2.0.2
18. 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.) Leveraging Applications of Formal Methods, Verification and Validation: Discussion, Dissemination, Applications - 7th International Symposium, ISoLA 2016, Imperial, Corfu, Greece, October 10-14, 2016, Proceedings, Part II. Lecture Notes in Computer Science, vol. 9953, pp. 150–170 (2016). https://doi.org/10.1007/978-3-319-47169-3_12
19. Schmidt, K.: LoLA: A low level analyser. In: Nielsen, M., Simpson, D. (eds.) ICATPN. pp. 465–474. Springer-Verlag (2000). https://doi.org/10.1007/3-540-44988-4_27
20. 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, February 19-21, 2016. pp. 14–22. SciTePress (2016). https://doi.org/10.5220/0005841800140022

21. 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'17, Zaragoza, Spain, June 25-26, 2017. Proceedings. CEUR Workshop Proceedings, vol. 1846, pp. 197–214. CEUR-WS.org (2017), http://CEUR-WS.org/Vol-1846/