Providing Petri Net-Based Semantics in Model Driven-Development for the Renew Meta-Modeling Framework

David Mosteller, Lawrence Cabac, Michael Haustermann

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

Abstract. This paper presents an approach to the development of modeling languages and automated generation of specific modeling tools based on meta-models. Modeling is one of the main tasks in engineering. Graphical modeling helps the engineer not only to understand the system but also to communicate with engineers and with other stakeholders that participate in the development (or analytic) process.
In order to be able to provide adequately adapted modeling techniques for a given domain, it is useful to allow to develop techniques, which are designed for their special purpose, i.e. domain-specific modeling languages (DSML). For this cause meta-modeling comes in handy. Meta-models provide a clear abstract syntax and model-driven design approaches allow for rapid prototyping of modeling languages. However, often the transformation and also the original (source model) as well as the transformed (target) model do not provide a clear semantics.
We present an approach to model-driven development that is based on Petri nets: high- or low-level Petri nets in various formalisms can be used as target models. Starting from the conceptual background and underlying thinking tool, following up with code templates, transformation engines, underlying semantics and the way our process support is implemented up to the final target engine Petri nets and Petri net tools can be used.

Keywords: Renew, Petri nets, model-driven development, meta-modeling

1 Introduction

Meta-modeling enables us to build models in a more abstract way than we are used to today. For many purposes we prefer languages that solve a specific modeling quest. While there are several well established modeling techniques with a clear semantics, the purpose of the incorporated languages is more or less fixed. Annotations like those in UML can in combination with profiles enhance the expressiveness. However, it is difficult to build lean languages that cover exactly those domain aspects that are required in a certain context. In addition, normally there exist no tools that directly support those languages with

specific language constructs. To make a language easy to use, one usually needs direct tool support. The development of tools for building graphical models was a challenge some years ago. Nowadays it is relatively easy within environments like Eclipse and its meta-modeling plugins. [1] Even extensions that allow a simulation of models built with those languages are available. However, usually these execution environments are relatively restricted and do not scale. This is due to the fact that the execution engine has to be built separately.

The development of a DSML and a corresponding modeling tool includes a whole range of tasks. We will address in this contribution: (1) providing the possibility to define an abstract syntax to allow users to build a special purpose language, (2) providing a graphical environment to allow users to build special language constructs for their specific language concepts (based on textual and graphical representations), (3) providing a tool set that allows to build models based on the previously defined languages and (4) providing a simulation environment (especially based on reference nets [10]) that allows users to execute and simulate their models. The presented approach to developing modeling languages and tools (RMT approach) is extensively applied within our approach to developing agent-oriented software based on Petri nets (P*AOSE approach, [3,13,17]), in which the mutual interplay of modeling languages is omnipresent. It is however equally applicable to other domains. We provide a prototype, which offers the possibility to develop modeling languages and to generate corresponding modeling tools. The RENEW Meta-Modeling Framework (RMT framework)[2] was applied in several settings. The RMT framework constitutes a further development step of the model-driven approach, which has been already envisioned and partly applied during the development of the Agent Role Modeler (ARM, [4]). The ARM tool, which was developed without appropriate meta-modeling tool support, provides the modeling facility for agent organizations and knowledge bases.

The remainder of this paper is structured as follows: The conceptual background, which comprises the model-driven tool development, encompassing meta-modeling, graphical modeling, transformations and semantical issues, is discussed with respect to the requirements and specification of our solution in Section 2. The example presented in Section 3 demonstrates the approach and the applications of Petri nets as well as the application of other techniques during DSML development. Section 4 elaborates on the wider context of model-driven development and the approach of providing transformational semantics for modeling languages with Petri nets. In Section 5 we will summarize our results and will give an outlook of our further research directions opened by these results.

[1] Eclipse Modeling Framework, EMF, `https://www.eclipse.org/modeling/emf/`
[2] RMT: RENEW Meta-Modeling and Transformation Framework,
tools and examples: `http://www.paose.net/wiki/Metamodeling`

2 Conceptual Approach

As our approach to software development is based on the model-driven construction of software systems our aim is to provide a tool chain using model-driven techniques. We want to support the agile development of graphical modeling languages. Therefore, we rely on the concepts of software language engineering [9] and apply model-driven techniques to generate tools from abstract models. In the following we elaborate on the techniques required to realize a framework based on generating modeling tools. Renew provides the basis for our meta-modeling framework. It serves as a graphical framework for the flexible construction of graphical models and at the same time provides the execution and simulation environment of Petri net models, which serve as target languages that provide the transformational semantics for the designed languages. This approach allows for the analysis of Petri net models and for the validation of model properties. Our conceptual approach is based on the idea of bootstrapping the required modeling tools using model-driven techniques. Following the concepts of software language engineering the development of modeling languages encompasses three aspects: abstract syntax, concrete syntax and semantics. Translating these concepts into the area of generative tool development leads to a set of descriptions defining the different aspects of software languages [15]: structure, constraints, representation and behavior. The structure (abstract syntax) and the representation (concrete syntax) of modeling languages will be addressed in the following section. The behavior (semantics) is covered in Section 2.2.

2.1 Meta-Modeling and Tool Generation

In this section we elaborate on the first part of the DSML development process. First we need to define the syntax of the new language (or technique). The abstract syntax of a language is specified by a meta-model, which defines the structure of the language. Our tool set, which is based on Renew, supports the modeling of the abstract syntax directly through the technique of Concept Diagrams (cf. [3, Chapter 12]). Concept Diagrams are simplified Class Diagrams, which are usually used to design type hierarchies or agent ontologies (in the context of P*aose). In the context of this work the type hierarchies of Concept Diagrams are utilized to model the meta-models of the designed DSML, i.e. the abstract syntax.

Additionally, in order to define the representation of the elements we also need to define the concrete syntax. The concrete syntax, i.e. the representation of the syntactic elements, is defined through a mapping from the syntactic element to it's graphical representation (representational mapping). The representational mapping includes concrete graphical or textual syntax as well as serialization representations. Typically, if the language should not be restricted to graphical standard figures, the layout, concrete form, etc. has to be defined in a form close to the implementation language. However, we provide also the possibility that the syntactic elements may be defined directly within the Renew environment using the graphical user interface. Each provided graphical representation is

stored as one template drawing and the representational mapping refers not to an implementation but to a template (graphical component). Alternatively some standard elements are provided, which can be configured in terms of stylesheets to define the representation for the language constructs.

In addition to the abstract and the concrete syntax, we need to configure the user interface of the modeling tool that provides the modeling facility – in the Renew environment the modeling tool is integrated as a plugin. The configuration is done defining another mapping for task bar tool buttons and their design together with some general information about the modeling tool, such as the file extension or the ordering of tool buttons in the task bar.

The Renew plugins that provide the modeling facility for the stylesheets and the tool configurations are themselves meta-model based. They have been generated – in a bootstrapping fashion – using the RMT approach.

Figure 1 shows the defining artifacts of a modeling language's syntax in the top. These artifacts are expressed within the scope of the meta-meta-model – the RMT meta-model – and can thus be used to generate a domain-specific modeling tool, which then provides the possibility to design a model, using the technique; e.g. a (domain-specific) modeling language.

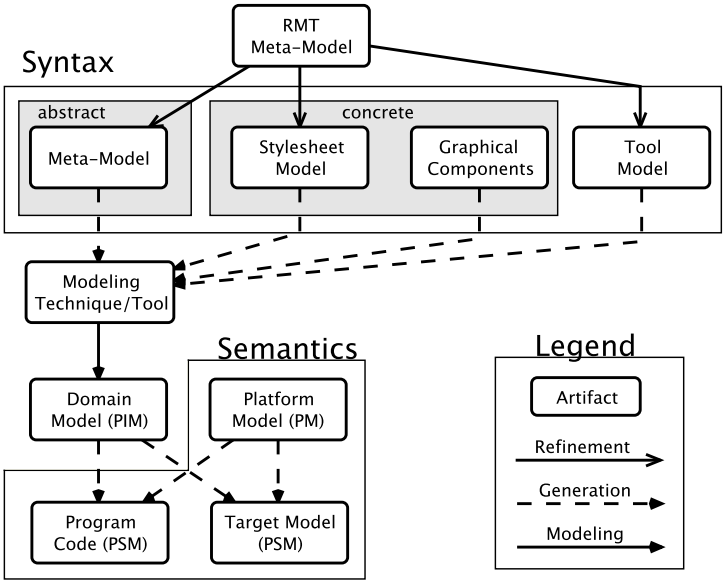

Figure 1. An abstract view on the models of a meta-modeling project.

A modeler may use the generated tool to model, store and retrieve graphical models (diagrams) in the syntax of the newly developed modeling language.[3] For

[3] In the following we will address these models as *domain model* or *source models*.

operational or analytic models, however, it is not enough to be able to provide graphical descriptions of the models. In these cases we need to define a clear semantics. Following the idea of the model-driven architecture (MDA) the semantic interpretation of a source model can be defined through a transformation into specific target models using a generator as shown in the lowermost part of Figure 1, which references the schematic view of Petrasch et. al. [18, p. 107].

We elaborate on this in the following section. But before we present the approach to the definition of the semantics, we stress the flexibility of the given approach, so far. The meta-modeling approach in itself offers a high degree of flexibility. By changing (augmenting, modifying or restricting) the meta-model we are able to quickly produce variations of modeling techniques, which may subsequently be compared with each other (see for instance Section 3.2). Additionally, we are able to change the representation of the modeling language by either changing the representational mapping or by editing the graphical components. Especially the latter can be done by someone without knowledge in the development details and thus create his own representation.

2.2 Transforming Source Models to Target Models

The semantics of a modeling language is defined – as semantic mapping, cf. [6] – either through formalization, through an operationalization or through the transformation into other models that already own a formal or an operational semantics. As we use the Renew environment as basis for our approach, we transform given source models to Petri net models, i.e. our *target languages* are Petri nets formalisms. The RMT approach is not restricted to behavioral modeling languages. By choosing the proper target languages, such as Reference Nets, modeling structural properties can be performed by applying the same approach. In Figures 1 and 2 we can identify the domain specific model (source model), which is transformed into a target model (Platform Specific Model, PSM) within the application domain layer (M1). The Renew environment together with its provided Petri net formalisms serve as Platform Model (PM, compare with Figure 1). In the context of model-driven development the source model is often described as Platform Independent Model (PIM).

The transformation process is depicted in Figure 2 as a schematic Petri net. Transitions represent actions provided by either the RMT tool set (generation, transformation, execution or analysis) or by the source model developer (modeling). Necessary artifacts for the development of the modeling language workflow are provided by the language developer using the RMT framework. These artifacts comprise the syntax meta-models, the transformer and the semantic elements provided as net components [3, Chapter 5]. Net components are Petri net snippets that are used as patterns to be mapped by a generator and combined to constitute the target models. In this sense the model transformation process can be characterized as a pattern-oriented transformation following the categorization of Petrasch et. al. [18, p. 132].

Besides supporting the agile development of graphical languages the RMT approach also provides a high level of flexibility regarding the semantic trans-

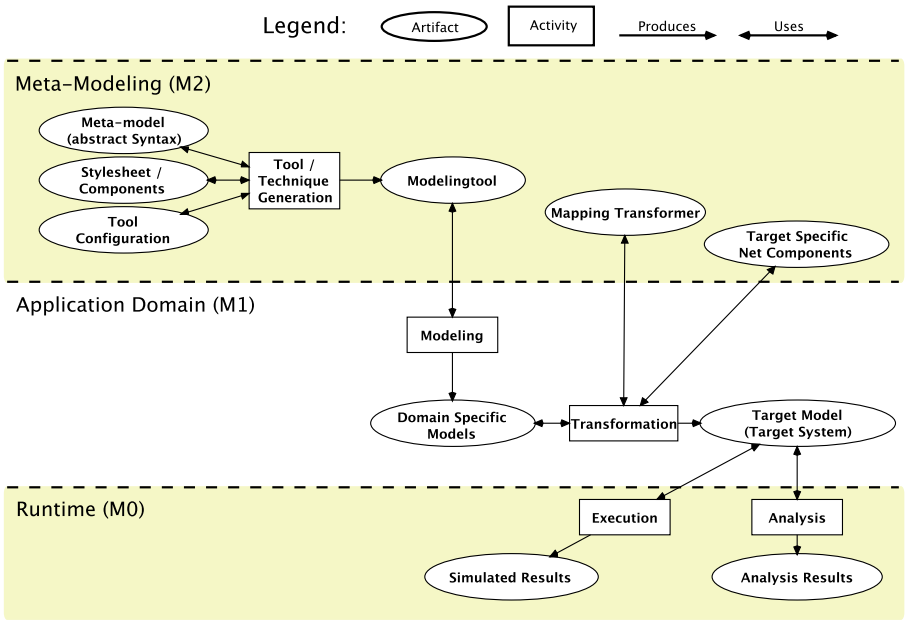

Figure 2. Artifacts and process within the RMT usage workflow.

formation. First, the semantic targets for the syntactic elements are defined as net components, which can be modified or exchanged easily. We are even able to provide several target mapping sets of net components, which can be expressed using distinct formalisms. Thus we are able to transform one source model into multiple forms of target models. For instance, we could transform a workflow description into a PT net for analytic examination and transform the same source model to a colored Petri net for simulation / execution within a real world application.

3 Developing a Prototype for BPMN

In the previous section we introduced a conceptual approach to developing modeling languages. We now show the concept in practice and demonstrate the concrete models, which are utilized in the development process. We have chosen to present, as example, the well-known modeling technique BPMN (Business Process Model and Notation [16]), in order to demonstrate the presented tool set.

In Section 3.1 we develop a (rather simple) modeling language that implements a subset of BPMN. We show, how model transformations can be used to generate Petri net models, which provide formal semantics to the abstract BPMN models. The generated Petri net models can be referred to for analyzing a BPMN process.

In a subsequent step a more specific modeling language is developed in Section 3.2. This second language – the BPMN_{AIP} formalism – enriches concepts from BPMN with domain-specific elements from the context of P*AOSE (see Section 1). The intention is to demonstrate the flexibility of the RMT approach and the appropriateness for agile, rapid and prototypical model-driven language development.

3.1 BPMN

We start with a simple subset of BPMN. Since BPMN has been described extensively in the context of modeling, meta-modeling and also in the context of Petri nets, we do not need to go into detail about the underlying semantics. A mapping of syntactic elements of BPMN to PT net components has been proposed by Dijkman et. al. [5]. Using these Petri net mappings we can focus on the aspects of agile language development instead. We concluded in Section 2.1 that a modeling language is based on the specifications of abstract and concrete (graphical) syntax.

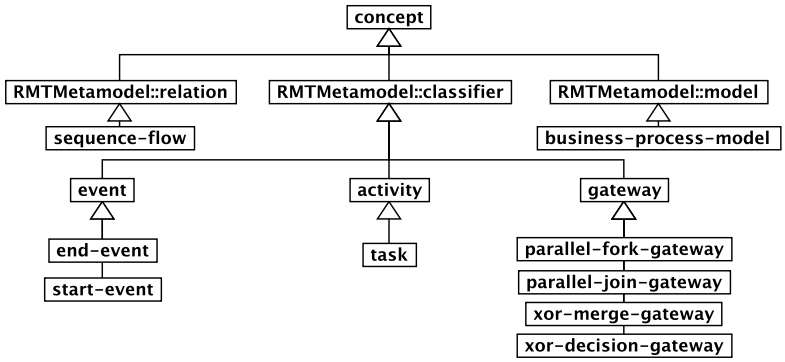

Figure 3. A meta-model for a subset of BPMN language constructs.

Figure 3 shows a meta-model for the chosen fragment of the BPMN. All concepts defined in this meta-model are instances of three basic concepts from the RMT meta-model: *model*, *classifier*, *relation*. Also shown in Figure 3 is that the three basic concepts are themselves instances of the single core concept (*concept*). The developed BPMN language defines a model type, the *business-process-model*. Also defined are *events*, *activities* and two different *gateways*, one for parallel processing and one with exclusive alternatives. These concepts can be connected through the *sequence-flow* relation. These concepts alone define the abstract syntax of the simplified BPMN formalism.

In order to complete the modeling language and generate the respective supporting modeling tool, the RMT approach requires additional information. One

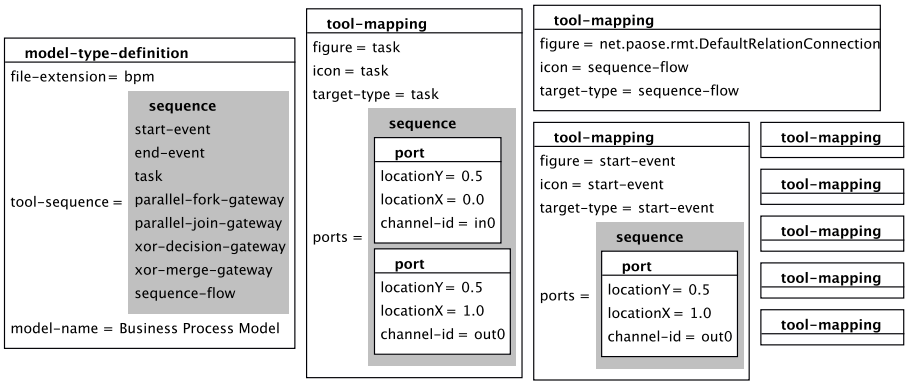

Figure 4. The tool configuration model for the BPMN modeling tool (with partly collapsed tool mappings).

is the visual representation of graphical constructs. These are developed using the built-in graphical constructs of the Renew drawing framework. Each graphical figure is stored in a separate drawing file (template) and can be used as syntactic element for modeling later on. Another required information is a specification of properties for the modeling tool. An example of a tool configuration is shown in Figure 4. This model contains basic properties such as a model name and a file extension as well as a set of tool mappings. The latter define mappings from concepts of the meta-model (*target-type*) to graphical constructs (net components). Connectors of the constructs are specified as *ports*, relative to their position. All elements of the tool configuration are expressed in Semantic Language (SL), which can be compared to Yaml or JSON and defined using the SLEditor plugin for Renew, which provides a UML-like representation as well as editing support for the modeler.

Figure 5 shows the graphical components representing the syntactic elements of the BPMN language alongside with the Renew UI, which presents the loaded palette for the BPMN drawing tools. The graphical components are defined in

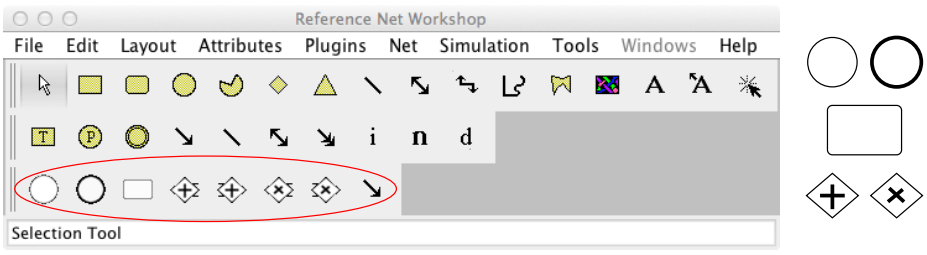

Figure 5. The Renew UI with the tool palette providing BPMN elements.

separate template drawings. The templates define the concrete syntax for the BPMN technique. This concludes the specifications for the modeling language and enables us to generate the plugin for the modeling tool. During the generation process the RMT generator (automatically) prepares the images that are used for the tool buttons on the basis of the graphical templates. The icon images of parallel and alternative gateways where slightly modified as shown in the encircled part of Figure 5 to better distinguish the complementary constructs of split and join figures.

Using the generated BPMN plugin we are now able to model with this new technique using the RENEW editor. Figure 6 shows a ticket workflow described in BPMN. The process reflects the lifecycle of support tickets in a conventional issue tracking system. Issues are created and at some point assigned to the holder of a certain role. They can be either rejected or accepted in which case the corresponding task will be carried out by the assignee. Later on, the task may be discontinued (*unassigned*) or completed (*finish*).

With the mapping of Dijkman et. al. [5] for the transformation to Petri nets we are able to transform the given workflow to a PT net model. The generated Petri net constitutes the transformational semantics of the BPMN process in the context of the RENEW simulation environment. In consequence, the resulting model can now be executed or analyzed using for instance the RENEW simulator.

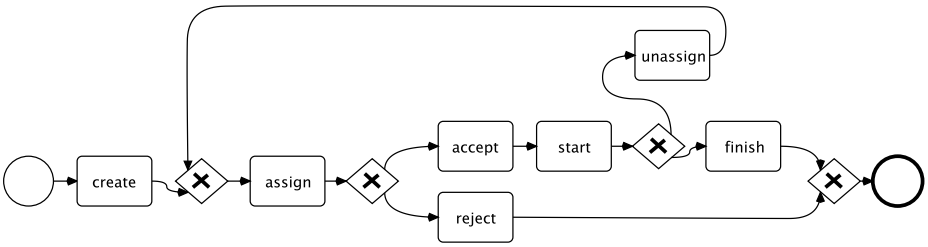

Figure 6. The lifecycle of tickets in a issue tracking system, modeled as BPMN.

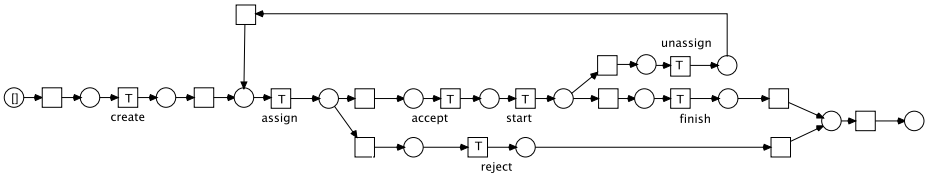

Figure 7. The target model of the lifecycle of tickets as PT net.

3.2 BPMN$_{\text{AIP}}$

The example presented in the preceding section describes the development of a modeling language together with a corresponding modeling tool (as RENEW plugin) following the RMT approach. Based on meta-models the approach provides a high level of flexibility in all stages during the development of modeling languages. This enables language developers to rapidly prototype specific languages, evaluate them and adapt them according to their needs. To further illustrate this flexibility, we now present a domain-specific variation of BPMN, called BPMN$_{\text{AIP}}$, which is used within the P*AOSE approach.

We use BPMN$_{\text{AIP}}$ to model agent interaction protocols. In contrast to Agent Interaction Protocol Diagrams (a variation of Sequence Diagrams [3, Chapter 13]), the BPMN$_{\text{AIP}}$ formalism allows to shift the focus to the internal agent processes. The presented agent-specific extensions have been proposed by Hausternann [7] in order to augment a subset of BPMN for the use within the P*AOSE approach. With the RMT approach it is possible to refine the BPMN language in an agile process and develop a corresponding modeling language (BPMN$_{\text{AIP}}$), which satisfies the demands of a given domain-specific context.

BPMN$_{\text{AIP}}$ extends the BPMN subset in the previous section by incoming (drawn as white envelopes) and outgoing (black envelopes) message events and special tasks for agent-specific operations. The *dc-exchange-task* represents the synchronous or asynchronous call of an internal service. The *kb-access-task* serves for accessing the agent's internal knowledge base. To use these constructs in the modeling tool, they have to be added to the meta-model. Figure 8 shows the extensions of the meta-model in Figure 3. With these extensions the modeling tool can already be used with the added constructs. The generated modeling tool uses a standard representation and standard task bar tool buttons to allow running early tests if none are provided by the developer. In order to define a customized concrete syntax, analogously to the previous example, a representation template drawn with the RENEW tool, a button icon generated from the template image and a tool mapping entry in the tool configuration as shown in Figure 4 is sufficient.

In addition to the agent-specific constructs, BPMN$_{\text{AIP}}$ also has a domain-specific semantics. The semantics is based on the agent framework that is applied in the P*AOSE approach, which uses Petri nets to implement agents and the agents' behavior. Therefore the semantic net components for the target model are tailored for the fitness within the used framework.

In order to obtain another semantics it is possible to provide a different set of net components. The RMT framework is able to handle different transformation engines and multiple net component sets. For the BPMN$_{\text{AIP}}$ formalism the MULAN net components by Cabac are used [3, Chapter 5].

Figure 9 shows an adaptation of the ticket service example using BPMN$_{\text{AIP}}$. The management of the ticket status is now provided by an agent, the Ticket Agent, which can delegate tasks to other agents. In this example the task to export some drawing to an image is assigned to an Export Agent (as described by Cabac et. al. [1]), which is informed about the assignment with a message.

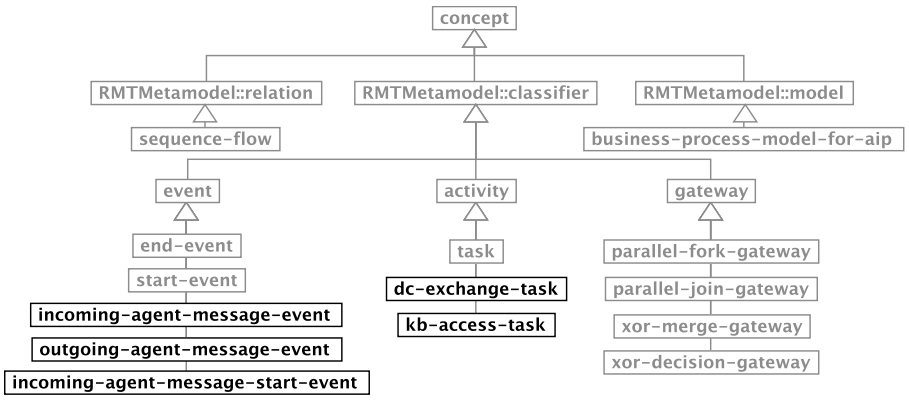

Figure 8. BPMN_{AIP} extensions to the BPMN meta-model (Figure 3).

This message results in an instantiation of the process depicted in Figure 9b. The Export Agent checks his knowledge base if it can export the drawing and delegates the task to an internal service, if possible. The Ticket Agent changes the status of the ticket according to the messages he receives as answer from the Export Agent.

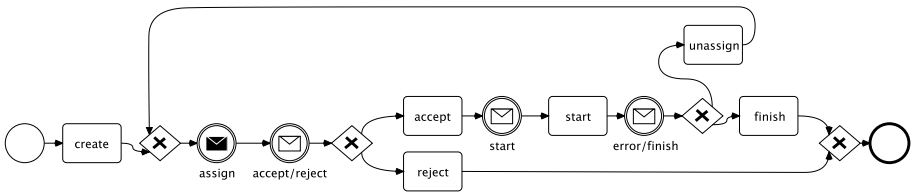

Figure 9a. The Ticket Agent.

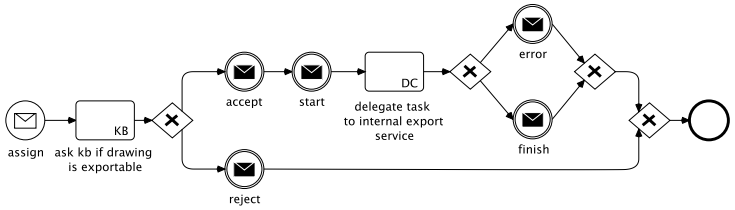

Figure 9b. The Export Agent.

With the described semantics the two agent processes in Figure 9 are transformed by the RMT-based BPMN_{AIP} modeling tool into the Petri nets shown in Figure 10. These nets are protocol net skeletons, which have to be completed by additional implementation details in order to be runnable with the used frame-

work. The figure illustrates the structure of the generated nets. A part of the net is zoomed in to exemplarily show the details of the net components. The zoomed part refers to the internal delegation of the task and the answer to the Ticket Agent.

4 Related Work

In this publication we motivate the rapid and prototypical development of domain-specific modeling languages. There are a number of related publications on prototyping domain-specific languages (DSL), each focussing on different aspects or application domains. Blunk et. al. [2] see the best gain for prototyping DSL as an extension of a general purpose programming language. Sadilek et. al. [20] stress the increasing demand for supporting agile approaches to the development of DSML. They "argue that for prototyping a DSML on the platform independent level, its semantics should not only be described in a transformational but also in an operational fashion" [20, p. 63]. However, they use the Query View Transformation (QVT) language to implement operational semantics of Petri nets, rather than exploiting the operational semantics to formalize the semantics of a second modeling language, as done here. Rouvoy et. al. [19] specialize on the domain of architecture description languages (ADLs) and develop a modular framework for prototyping ADLs based on the Scala language. The presented method emphasizes the high degree of automation through generative methods, the automated generation of modeling tools and also the automated transformation of abstract models to Petri nets.

Nytun et. al. [15] provide a categorization for different approaches to automated tool generation in the context of meta-modeling and DSML. The authors examine various meta-modeling approaches according to the four categories: structure, constraints, representation and behavior. With the RMT approach we cover most of these aspects by utilizing concept diagrams, representational mappings and Petri net-based target models. At the current time we do not provide any means to define constraints, but we plan to introduce constraints in the future.

With the claim of addressing general problems of defining DSML semantics our goal consists in developing a Petri net based framework through combining techniques of meta-modeling with Petri nets engineering. With the Event Coordination Notation (ECNO) Kindler [8] takes a model-driven approach, which uses Petri net models to implement local components behavior. The collaboration of components is defined in abstract coordination diagrams. The implementation is based on the wide spread Eclipse Modeling Framework (EMF). In combination with the EMF, the graphical modeling framework (GMF) can be used to automatically generate specific modeling tools from meta-models. The idea of generating domain-specific tools from models was adopted for this work, but we try to take a minimalistic approach instead of overcharging the tool with features, thus increasing complexity. The intrinsic complexity is a point of criticism concerning meta-modeling frameworks [19, p. 14].

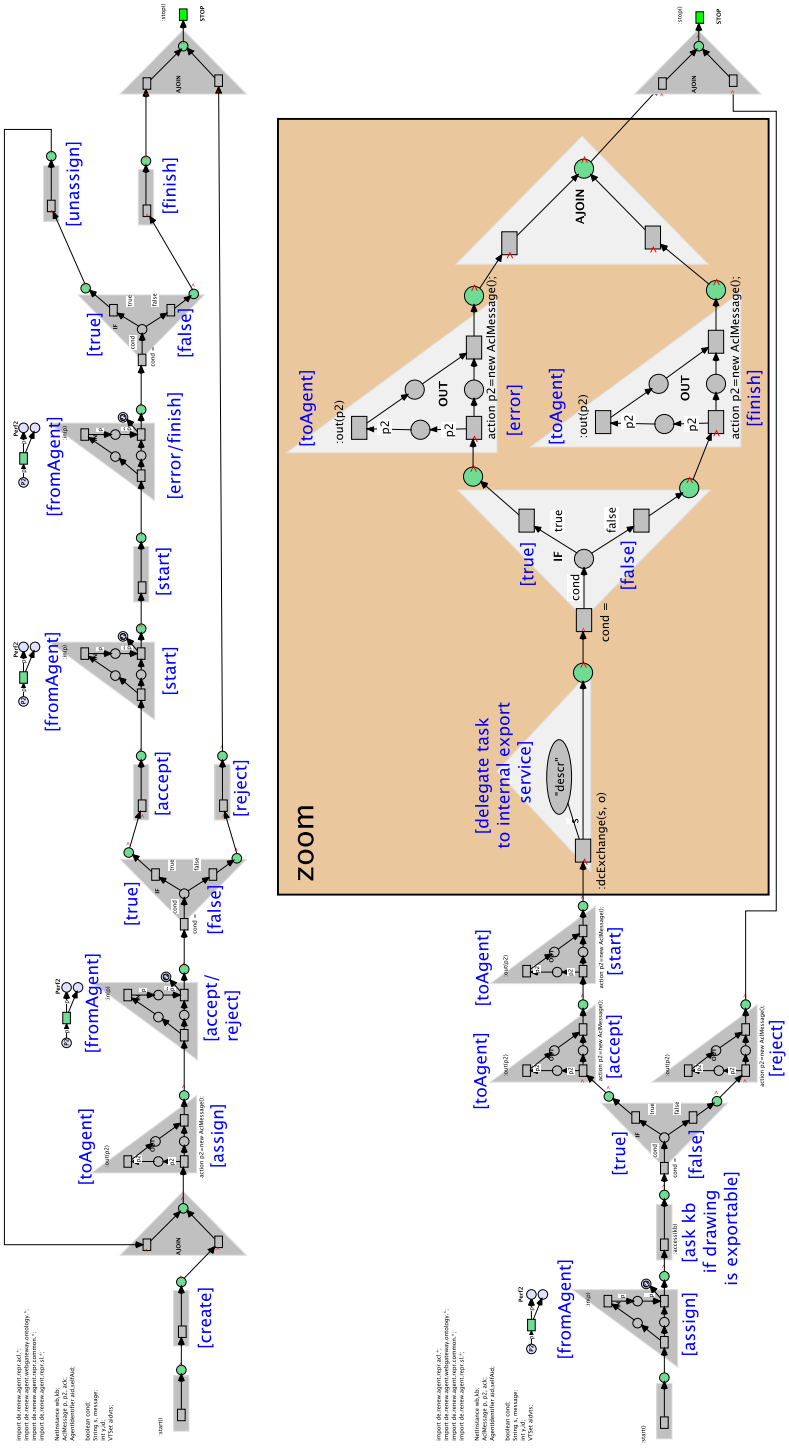

Figure 10. Generated protocol net skeletons.

Dijkman et. al. [5] show a mapping of BPMN constructs to Petri nets and elaborate on the semantics of such transformations. On the basis of that work there exists a tool for converting BPMN models to PNML and a tool for converting BPMN to YAWL. With the flexible tool presented in this work the languages can be quickly adopted and the concerns about problems in evaluating BPMN models using Petri net semantics can be empirically investigated. Lohmann et. al. [12] provide a basis for analyzing different business process modeling languages with respect to their realizability using Petri nets semantics for BPEL, BPMN, EPC, YAWL. This can be a good starting point for further research using the presented tool. The RMT framework has been applied by Möllers [14] for the development of a modeling tool for the design and execution of Deployment Diagrams.

5 Conclusion

In this contribution we present the RMT approach, which enables us to develop modeling languages and modeling tools by applying concepts of model-driven development. The key aspects of this approach are the use of meta-models for automatic tool generation and transformation of models, exploiting the formal semantics of Petri nets. Based on our continuously developed graphical modeling tool and Petri net simulation environment Renew (see [11]) we provide the technical realization of the RMT approach. The RMT framework provides the means to describe modeling languages building on the concepts of software language engineering (cf. Section 2). The abstract syntax, concrete syntax and tool configurations are provided as model-based specifications of the desired modeling languages and tool behavior. The semantics is defined as transformation-based operational semantics using Petri net formalisms as target models. With this environment we are able to provide the representation directly within our graphical framework, leading to appropriate language constructs, which can be designed for special purposes that fit the needs and expectations of its users. With the RMT approach the users are able to develop and adapt their own languages/ńodeling techniques, define constructs based on graphical representations and finally generate modeling tools, which empower them to draw models in domain-specific languages.

Depending on the chosen and intended formalism, we could even go one step further. We are able to simulate the transformed models directly, if there exists an operational semantics that can be mapped to the formalisms we already have implemented within the Renew context. For experimental environments where users want to define a special purpose language that suits exactly their current needs we can therefore provide a powerful tool set.

While the prototypical development of languages is already quite fast, we now have to address the question of sustainable meta-modeling-based tools. We have already successfully applied the tool several times within our P*aose approach. In this context we expect that further new modeling languages can be developed in a prototyping approach. In the future we wish to provide the means

to support hierarchical modeling within the RMT framework. With the Nets-within-Nets paradigm [21] the concepts to support hierarchical target models already exist. Since the whole P*AOSE approach is Petri net-based, the direct support by simulation of target models within RENEW is implicitly given. The prototyping approach of languages empowers us to evaluate several languages in order to improve specific frameworks that are already at hand.

Acknowledgment We thank Dr. Daniel Moldt and the TGI group of the Department of Informatics, University of Hamburg for the support, constructive criticism and fruitful discussions.

References

1. Betz, T., Cabac, L., Duvigneau, M., Wagner, T., Wester-Ebbinghaus, M.: Software engineering with Petri nets: a Web service and agent perspective. Transactions on Petri Nets and Other Models of Concurrency (ToPNoC) pp. 41–61 (2014), http://link.springer.com/chapter/10.1007/978-3-662-45730-6_3
2. Blunk, A., Fischer, J.: Prototyping domain specific languages as extensions of a general purpose language. In: Haugen, O., Reed, R., Gotzhein, R. (eds.) System Analysis and Modeling: Theory and Practice, Lecture Notes in Computer Science, vol. 7744, pp. 72–87. Springer Berlin Heidelberg (2013), http://dx.doi.org/10.1007/978-3-642-36757-1_5
3. Cabac, L.: Modeling Petri Net-Based Multi-Agent Applications, Agent Technology – Theory and Applications, vol. 5. Logos Verlag, Berlin (2010)
4. Cabac, L., Mosteller, D., Wester-Ebbinghaus, M.: Modeling organizational structures and agent knowledge for Mulan applications. Transactions on Petri Nets and Other Models of Concurrency (ToPNoC) pp. 62–82 (2014), http://link.springer.com/chapter/10.1007/978-3-662-45730-6_4
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 (2008), http://dx.doi.org/10.1016/j.infsof.2008.02.006
6. Harel, D., Rumpe, B.: Meaningful modeling: what's the semantics of "semantics"? Computer 37(10), 64–72 (Oct 2004)
7. Haustermann, M.: BPMN-Modelle für petrinetzbasierte agentenorientierte Softwaresysteme auf Basis von Mulan/Capa. Master thesis, University of Hamburg, Department of Informatics, Vogt-Kölln Str. 30, D-22527 Hamburg (Sep 2014)
8. Kindler, E.: Coordinating Interactions: The Event Coordination Notation. Tech. Rep. 05, DTU Compute - Department of Applied Mathematics and Computer Science, Technical University of Denmark (2014)
9. Kleppe, A.: Software language engineering: creating domain-specific languages using metamodels. Pearson Education (2008)
10. Kummer, O.: Referenznetze. Logos Verlag, Berlin (2002), http://www.logos-verlag.de/cgi-local/buch?isbn=0035
11. Kummer, O., Wienberg, F., Duvigneau, M., Cabac, L.: Renew – User Guide (Release 2.4.2). University of Hamburg, Faculty of Informatics, Theoretical Foundations Group, Hamburg (Jan 2015), http://www.renew.de/

12. Lohmann, N., Verbeek, E., Dijkman, R.: Petri net transformations for business processes - a survey. In: Jensen, K., Aalst, W. (eds.) Transactions on Petri Nets and Other Models of Concurrency II, Lecture Notes in Computer Science, vol. 5460, pp. 46–63. Springer Berlin Heidelberg (2009)
13. Moldt, D.: Petrinetze als Denkzeug. In: Farwer, B., Moldt, D. (eds.) Object Petri Nets, Processes, and Object Calculi. pp. 51–70. No. FBI-HH-B-265/05 in Report of the Department of Informatics, University of Hamburg, Department of Computer Science, Vogt-Kölln Str. 30, D-22527 Hamburg (Aug 2005)
14. Möllers, K.S.M.: Ein Ansatz zur Dynamisierung von Verteilungsdiagrammen anhand der Entwicklung eines Mulan-Werkzeugs. Bachelor thesis, University of Hamburg, Department of Informatics, Vogt-Kölln Str. 30, D-22527 Hamburg (2014)
15. Nytun, J.P., Prinz, A., Tveit, M.: Automatic Generation of Modelling Tools. In: Rensink, A., Warmer, J. (eds.) Model Driven Architecture - Foundations and Applications, Lecture Notes in Computer Science, vol. 4066, pp. 268–283. Springer Berlin Heidelberg (2006), http://dx.doi.org/10.1007/11787044_21
16. OMG, Object Management Group: Business Process Model and Notation (BPMN) – Version 2.0.2 (2013), http://www.omg.org/spec/BPMN/2.0.2
17. Paose-Website: Petri Net-Based, Agent- and Organization-oriented Software Engineering. University of Hamburg, Department of Informatics, Theoretical Foundations Group: http://www.paose.net (2015)
18. Petrasch, R., Meimberg, O.: Model Driven Architecture: eine praxisorientierte Einführung in die MDA. Heidelberg: dpunkt-Verlag (2006)
19. Rouvoy, R., Merle, P.: Rapid Prototyping of Domain-Specific Architecture Languages. In: Larsson, M., Medvidovic, N. (eds.) International ACM SIGSOFT Symposium on Component-Based Software Engineering (CBSE'12). pp. 13–22. ACM, Bertinoro, Italie (Jun 2012), http://hal.inria.fr/hal-00690607
20. Sadilek, D., Wachsmuth, G.: Prototyping visual interpreters and debuggers for domain-specific modelling languages. In: Schieferdecker, I., Hartman, A. (eds.) Model Driven Architecture - Foundations and Applications, Lecture Notes in Computer Science, vol. 5095, pp. 63–78. Springer Berlin Heidelberg (2008), http://dx.doi.org/10.1007/978-3-540-69100-6_5
21. Valk, R.: Object Petri Nets – Using the Nets-within-Nets Paradigm. In: Desel, J., Reisig, W., Rozenberg, G. (eds.) Advances in Petri Nets: Lectures on Concurrency and Petri Nets, Lecture Notes in Computer Science, vol. 3098, pp. 819–848. Springer-Verlag, Berlin Heidelberg New York (2004), http://www.springerlink.com/openurl.asp?genre=article&issn=0302-9743&volume=3098&spage=819