# Simulating Multiple Formalisms Concurrently Based on Reference Nets

Pascale Möller, Michael Haustermann, David Mosteller, and Dennis Schmitz

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

**Abstract** Modeling complex systems nowadays requires a combination of techniques to facilitate multiple perspectives and adequate modeling. Therefore, UML and other formalisms are used to enrich the portfolio of Petri nets. Often the different models are transformed into a single formalism to simulate the resulting models within a homogeneous execution environment. For UML, the mapping is usually done via the transformation to some programming language. Anyhow, the problem with generative techniques is that the different perspectives that are provided by the applied modeling techniques can hardly be retained once the models are transformed into a single formalism.
In this contribution we elaborate on how multiple formalisms can be used together in their original representation. One of the main challenges for our approach is the provision of means for coupling mentioned formalisms so they can be executed together. We utilize the synchronization features of Reference Nets to couple multiple modeling techniques. This results in a simultaneous and concurrent execution of models featuring a combination of multiple modeling formalisms. A finite automata (FA) modeling and simulation tool is presented to showcase the principle concepts and options that are gained by our results.

**Keywords:** Petri Nets, Multi-Formalism, Model Synchronization, Reference Nets, Finite Automata

## 1 Introduction

Modeling complex systems requires a separation of concerns and demands support for taking multiple perspectives on the system including various levels of abstraction. This is achieved by combining several modeling techniques.

Looking back at the original ideas of Carl Adam Petri [27], automata are enhanced by a communication mechanism, thus introducing the modeling of concurrency. The new formalism, he calls "net", facilitates the modeling of additional concepts: locality (of time and place), asynchronism and concurrency. With his conceptual extension to the formalism comes a shift of perspective. In consequence, nets may serve for other purposes than automata.

Petri nets in their various forms have proven to be an adequate technique to model, analyze and understand concurrent systems. They provide an operational

semantics, and by utilizing high-level Petri nets, modelers have a technique at hand to cover multiple abstraction levels and perspectives. Anyhow, Petri nets are not always the optimal solution for every modeling task. We agree with Milner, who says: "I reject the idea that there can be a unique conceptual model, or one preferred formalism, for all aspects of something as large as concurrent systems modeling" [25, p. 78].

Depending on the context and the application area, various requirements to the modeling technique have to be met. Extending a formalism (or creating a new one) allows to shift the focus to other aspects, thus providing a different abstraction. Multiple techniques may be combined to facilitate various perspectives in order to cover different levels of abstraction (vertical) or for a combination of complementing views (horizontal).

Today, modeling techniques are quite elaborate, and the means for modeling are systematically developed and researched. The purposes for constructing *conceptual models* and their usage areas are manyfold. Krogstie [19, p. 3] has put them into categories: models may be used for system (forward) design, for the (computer-assisted) analysis, to support communication, for quality assurance, model deployment and activation, or just for making sense out of something.

The complexity of systems has dramatically increased since the beginning of computer science. This raises the necessity to provide several perspectives at the same time. Of course, taking different perspectives makes no sense at all if they can not be somehow related, which means they have to be linked together.

Standard modeling languages like UML (Unified Modeling Language) offer a portfolio of techniques to provide multiple modeling perspectives covering various levels of abstraction. Model driven development approaches facilitate the development of domain specific languages that can be related on the basis of meta-models. Usually, they are generated to a single target language, which comprises a loss of the desired abstraction level and an alteration of the perspective. Model simulation environments support the execution of models, but models usually do not provide the means to be used in combination.

The concurrent execution of multiple modeling techniques in their original representation, using explicit model coupling, are promising in order to combine the advantages of linking models and direct simulation. Consequently, tool support is required not only to support the modeling of multiple perspectives, but also to integrate the different views and to keep them consistent with each other.

We identify the following two main challenges. (1) For the generic coupling of multiple modeling techniques an adequate mechanism has to be found. (2) How is it possible to simulate coupled models from various modeling techniques in one environment concurrently without losing the original representation?

In this work we propose a conceptual extension of current modeling techniques by synchronous channels [7] as they are provided in the context of Reference Nets. Synchronous channels can supply synchronous communication between models in order to exchange data or control other models. Furthermore, we propose to map the constructs of the modeling techniques to Reference Net constructs. It should be noted that this mapping is not equivalent to the trans-

formation that we reject above, since the original representation of the model is preserved and even visually visitable during simulation. For modelers it should appear as if a model is being simulated in its original modeling technique. In order to achieve this, we propagate the simulation events of the underlying Reference Net back to the original model.

## 2 Proposal for Coupling through Synchronization

In this section we briefly introduce the conceptual and technical background of our work and present a simple introductory example of the coupling of multiple formalisms. First, we present the Renew environment for modeling and execution of Reference Nets and other modeling techniques. (Java) Reference Nets as our main modeling and simulation formalism are introduced afterwards. Last, with a simple example, we demonstrate our idea of multi-formalism execution, which is presented in general in Section 3.

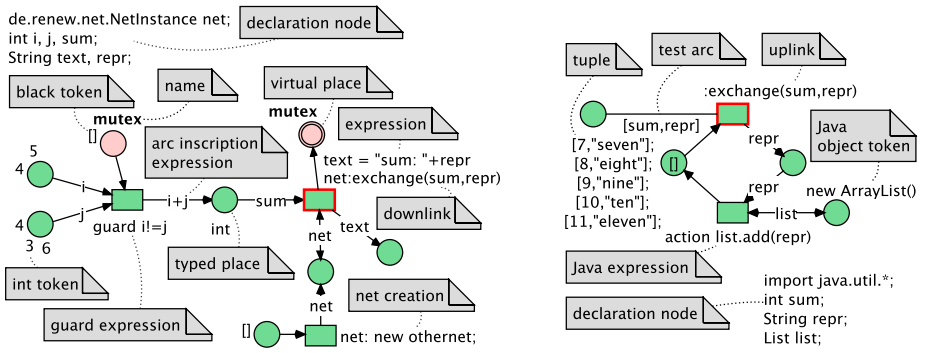
### 2.1 Renew

Renew is a continuously developed extensible modeling and simulation environment for Petri nets and other modeling techniques [21]. Due to its recent enhancements and extensions it has evolved into an IDE (integrated development environment) for the development of Petri net-based software [5]. Renew is focused on Reference Nets and has full support for that formalism in terms of modeling and execution with or without graphical feedback. Benefiting from its plugin architecture, over the past years, Renew has been extended with multiple modeling techniques and formalisms. With recent efforts regarding model-driven language engineering [26], Renew has become an environment for the development of modeling techniques and tools as well. The current development version has full support for the concurrent and coupled simulation of multiple formalisms. Renew is written in Java and available for various platforms such as Linux, Mac OS and Windows including the source code.[1]

### 2.2 Reference Nets

The (Java) Reference Nets formalism [20] is a high-level Petri net formalism that combines the nets-within-nets paradigm [29] with synchronous channels [7] 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

[1] The full multi-formalism feature will be part of Renew version 2.6 and then be available at `http://renew.de`. The current development version can be found at `http://paose.net/wiki/MultiFormalism`.




(a) somenet (b) othernet

Figure 1: Example Reference Net system showcasing selected features (image and description adapted from [6], image originally from [12, p. 77])

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 [22], the formal definition can be found in [20] (in German). In the following we give a brief overview over the features of Reference Nets based on an example.

The Reference Net system depicted in Figure 1 exhibits a large part of the constructs of Reference Nets and features those that are relevant for this contribution. It consists of two corresponding nets, *somenet* and *othernet*. The places can hold black tokens, elementary data types, or references to complex data types or embedded Reference Nets. Java types are imported and declared in the declaration node. Net instances are created from net templates using the `new` keyword as shown in the lower part of Figure 1a. Transitions have a flexible inscription language featuring guards, synchronous channels, and Java expressions. Places and arcs can also hold inscriptions. Virtual places are virtual copies of places, where the same semantic place may have multiple graphical figures in order to prevent long and crossing arcs.

The two highlighted transitions from *somenet* and *othernet* form a synchronous channel, which always consists of two parts: *downlink* and *uplink*. The downlink must hold a reference to the net containing the uplink. The net *somenet* in Figure 1a holds the `:exchange(sum,repr)` channel's downlink and a reference (`net`) to the net *othernet* in Figure 1b, which contains the uplink. Even though the invocation of the synchronous channel is directed, due to the required reference, the information exchange is bidirectional.

The unification algorithm searches for bindings of a transition that satisfy the possible assignments of variables with respect to the declared inscriptions on transitions, arcs, and places. If both transitions participating in a synchronous channel are enabled, they can fire synchronously and exchange information in both directions. In this example the `:exchange(sum,repr)` channel unifies the

sum of the values of `i` and `j` bound to the variable `sum` in *somenet* with the variable `sum` from the tuple of `sum` and `repr` from *othernet*.

The mechanism of synchronous channels provides powerful features for the coupling of multiple models. As described above, a synchronous channel consists of a pair of up- and downlink. The main reason for this is an efficient implementation (with mostly polynomial complexity instead of exponential complexity) that has been described in [20]. If multiple uplinks of one net can participate in the firing of synchronized transitions, one of the possible uplinks is chosen non-deterministically. For each downlink (and its parameters) there is exactly one uplink that will be bound when a transition is fired, and both must participate since the firing of the transitions is atomic. In the following we will briefly sketch our approach to coupling multiple modeling techniques by introducing our running example, before we generalize from this idea in order to develop our conceptual approach.

### 2.3 Coupling Finite Automata with Reference Nets

The coupling of finite automata and Reference Nets serves as a simple example for coupling multiple formalisms. We facilitate the coupling through enhancing the finite automata formalism with synchronous channels. More precisely, we develop a concept to synchronize finite automata state transitions with the firing of Reference Net transitions.

In this section we outline the coarse idea by presenting a useful example for the coupling of finite automata and Reference Nets. The general concept for the coupling and simulation of multiple formalisms is described in Section 3. An exemplary implementation that allows the simultaneous and synchronized execution of both – finite automata and Reference Nets – is described in Section 4.

One application area of multi-formalism simulation is the controlling of systems to avoid unwanted behavior such as deadlocks or security violations. To enforce orderly behavior, it is possible to use a controlling instance to restrict the possibilities of the controlled system. Ezpeleta et al. use controllers with Reference Nets in such a way [10]. Finite automata are well-suited for this purpose because they provide an intuitive description of the desired behavior.

We present a simple example to motivate a goal of controlling nets – avoidance of deadlocks and unwanted situations, as mentioned by Burkhard [4]. The Reference Net in Figure 2 depicts a simple production and consumption process. The consumption (lower right part) requires at least one preceding production (upper right part) to prevent the system from running into a deadlock.

At the left hand side there are three manually fireable transitions that each instantiate one of the controller automata shown in Figure 3. A reference token can be removed anytime by the centered transition. Producing puts a token into the storage place. Consuming is processed in two steps. In the first step the consumption process is entered. In a second step a token from the storage is consumed. If there is none, the net is stuck in a deadlock.

To avoid a deadlock, one may use a finite automaton. Three deadlock-avoidance strategies are shown in Figure 3. Strategy *avoid* (Figure 3a) constrains the

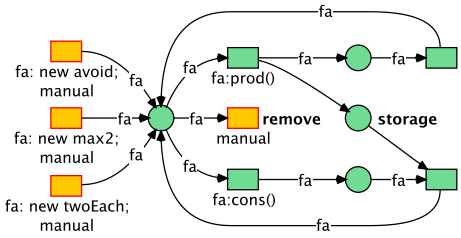
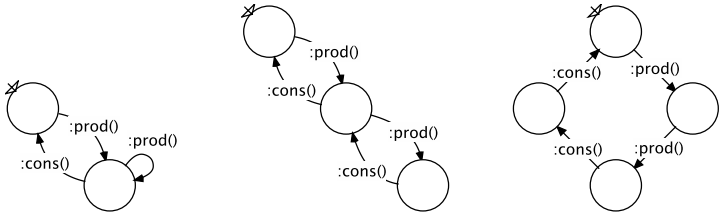



Figure 2: Reference Net with potential deadlock

net to produce at least once before each consumption. Strategy *max2* (Figure 3b) limits the number of tokens in the storage to 2, so that a consumption process takes place at least after every second production. The third strategy *twoEach* (Figure 3c) predefines the order to two productions followed by two consumptions repeatedly. With all three strategies a deadlock is avoided.




(a) avoid (b) max2 (c) twoEach

Figure 3: Different deadlock avoidance strategies

## 3 Concept for Multi-Formalism Simulation

In the following we elaborate on our approach to multi-formalism simulation that we sketch in Section 2. We will generalize the idea of using finite automata to control and visualize parts of a system. We present a method to link multiple formalisms through synchronized actions on the basis of Reference Nets.

### 3.1 Coupling via Synchronous Channels

In order to capture the overall structure and behavior of complex software systems, several modeling techniques are applied in combination. Coupling of models supports the modeling process in general by providing modeling techniques that exactly match the requirements of the modeling purpose. Each created model covers a distinct yet partly overlapping perspective of the system. All models need to be integrated in a consistent way to cover the complete system.

In a common setting – e.g. when using UML – the models are, more or less, modeled in isolation. The relations to other modeling techniques and therefore between the created models are not shown explicitly. While at first sight this makes it easier for the modeler, this is a problem. Modelers must know how models interact with each other. Usually, this relation is established by the compiler of the models if the models can be used directly for code generation. In most modeling environments this is not the case. In consequence, the models are complemented with implementation details to facilitate their execution.

As motivated in the introduction, the transformation of models to a single target language involves a loss of perspectives when looking at the executed system. While, of course, a common execution language may work in the background, we propose to directly execute the models and to make the coupling explicit in order to retain the perspectives. The combined simulation in their original representation requires an operational semantics of the applied techniques and a mechanism for the coupling of models.

Coupling multiple techniques requires considerable conceptual and technical support. We use, in addition to the operational semantics of the modeling techniques, an execution environment that properly supports multi-formalism simulation: Renew. A drawback of the explicit coupling of multiple modeling techniques is that it involves an extension of the modeling languages.

Direct simulation is not applicable for every modeling technique. For example the execution of a solely structural diagram seems not to be useful. In this contribution we mainly consider behavioral techniques that are discrete and state-based. This covers many of the UML behavioral diagrams and process modeling languages, such as BPMN and EPC.

### 3.2 Model Coupling with Graphical Feedback

In our group we have studied the necessary and sufficient solutions to implement modeling techniques within our framework(s). To extend high-level Petri nets by synchronous channels, elaborate algorithms were needed. The specification, design and implementation was a highly complex task. However, on top of this a very powerful semantics for the description of other modeling techniques is available now. With previous contributions we have shown how a new formalism can be developed on the basis of Renew by providing operational semantics through a mapping to Petri nets [16,26]. In this contribution we propose to create formalisms that use a mapping to Reference Nets in the background in order to provide the operational semantics for the modeling technique but present the original representation to the user.

Figure 4 summarizes the general idea of our approach to multi-formalism modeling and execution. The upper part (Graphical Layer) of the figure contains the graphical models (model drawings) and model instance drawings that are visible to the user and permit user interaction. Model drawings are artifacts that are created from a graphical editor within Renew or may be imported from an external tool. The instance drawings reflect the simulation state to the user and allow control over the simulation, e.g. by triggering a certain simulation step.

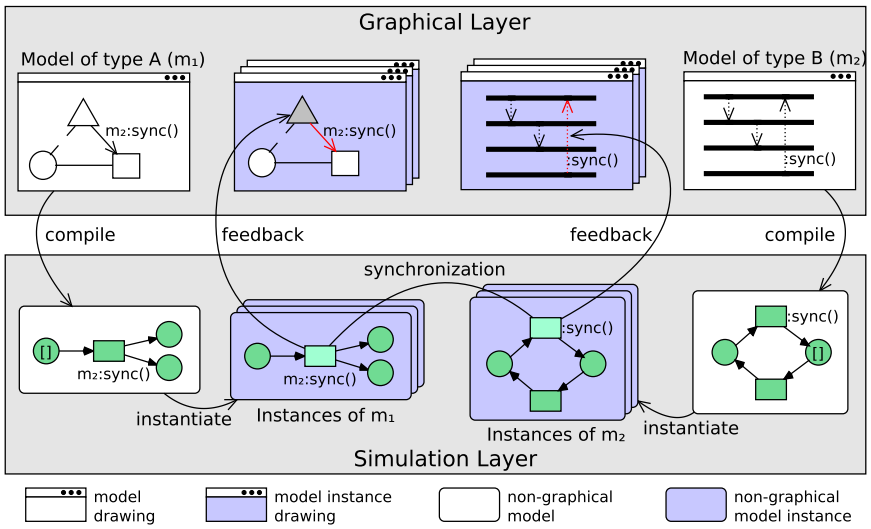



Figure 4: Conceptual model of the model synchronization

The displayed model drawings on the top-left and top-right hand side are arbitrary in a sense that they do not have a real application or semantics and serve as representative for any modeling technique a modeler may want to use. A simple example of how a concrete modeling technique is implemented is provided for communicating automata in Section 4. For a specific modeling technique a mapping from the constructs of the modeling technique to Reference Net constructs is needed in order to obtain an executable model. This may be compared to a code generation approach. The development of such a mapping is a part of the modeling effort, a developer of a modeling language is obliged to perform.

Graphical models are compiled into non-graphical Reference Nets, which can be instantiated and executed in the Renew simulator (Simulation Layer). Synchronization is performed on the level of the Reference Net instances (originating from the same or other modeling techniques). A mapping of the dynamic view of generated Reference Nets back to the modeling technique constructs even allows a direct feedback. The presented solution enables the simulator to pass simulation events from the Reference Net to the original model, e.g. to highlight a corresponding graphical figure. Introducing an additional Reference Net layer on top of the simulator has advantages, but also comes with a few restrictions.

### 3.3 Discussion

The best solution for a modeling and execution tool with respect to performance is an optimized implementation with a generic coupling mechanism. However, such an implementation from scratch is hard and time consuming. Our approach based on Reference Nets simplifies the development of executable, coupled modeling techniques with a powerful and yet easy to use mechanism for synchro-

nization. Due to reasonable hardware and an efficient implementation of the Reference Net formalism, this approach is applicable without a huge drawback in terms of performance, and the Reference Net formalism is an adequate language for the definition of an operational semantics.

A benefit of our concept is that the means for synchronization are made utilizable. This makes it easier to develop formalisms and to concentrate on the modeling language engineering. It supports a prototypical approach because language constructs may be designed in their Reference Net representation to be later on replaced with the constructs of the formalism in development. Synchronous channels provide a common conceptual basis for combining multiple formalisms. All formalisms share the same mechanism for communication and are consequently designed to be used in combination. Users directly profit from the *native* feedback in comparison to generative approaches.

Reference Nets are well-suited to cover multiple properties of modeling techniques, especially those of discrete, state-based behavioral techniques that we consider in this contribution. In order to simulate an arbitrary number of models concurrently, the execution language has to provide an intuitive mechanism to support the implementation of concurrency. The requirement for atomicity of activities and resource allocation emerges from the possibility of concurrent activities in a set of models. The simulated models should not effect each other, unless it is explicitly intended via synchronization. Thus, the execution language has to support strict data encapsulation for models in simulation. For process modeling languages often dynamic hierarchization is required (e.g. for subprocesses in BPMN 2.0). Reference Nets as Petri net formalism provide a powerful mechanism to implement concurrent behavior, resource allocation and atomicity of actions. Due to the nets-within-nets concept, encapsulation of simulated models and dynamic hierarchization is provided naturally by Reference Nets.

The utilization of synchronous channels as a coupling mechanism implies that the coupling always has to be synchronous, but the implementation of asynchronous coupling is possible as well. Two channels with an intermediate buffer-place result in asynchronous behavior.

Another property that is inherent in many modeling techniques is time. An extension of Reference Nets with timed expressions exists [22]. The presented concept is thus applicable for timed modeling techniques using this formalism as basis. Renew supports the execution of these nets. However, the formalism can only be simulated sequentially.

Generating models into a single formalism also has the advantage of being able to verify within a single formalism. For the intermediate format of Reference Nets, however, there are not many verification tools available yet. In general the usage of a single formalism seems to be valuable. With some of our tools, that are currently under development, we can generate a reachability graph for some restricted net formalisms.

The use of Reference Nets as coupling mechanism comes with some limitations. As described in Section 2.2, one synchronous channel, comprising up- and downlink, synchronizes exactly two transitions. There exist no means for global

synchronization (due to the basic idea of the locality principle of a transition) and consequently, there is no broadcast communication synchronizing all transitions by default. A synchronous channel only provides the synchronized action of two elements. The synchronization of more than two transitions is accomplished by combining multiple channel inscriptions on one transition (having at most one uplink, but several downlinks). In many cases, these restrictions do not constrain the general possibilities of modeling or easy alternatives exist: e.g. when a model instance can/shall not hold references to other models, communication may be provided by a system net that holds references to all participants.

The limited scope of variables to one transition and the connected arcs is a sensible property for Petri net formalisms, because it fits the general concept of locality, which is inherent in Petri nets. For other modeling techniques this may be a limitation. In many cases, Reference Nets are still adequate to provide operational semantics to modeling techniques with global variables. The use of a single place for each variable and an arc connection to these places for every transition can be used to simulate global variables. However, the variables in Reference Nets are immutable in a sense that there is unification but no assignment of variables. Therefore, the use of additional variables is required sometimes. This said, we move on to the development of a formalism that profits from the introduced concept for multi-formalism simulation.

## 4 Development of a Finite Automata Plugin

In order to demonstrate the principle concepts that are applied to couple models we describe the development of the Finite Automata plugin (FA plugin). This plugin extends Renew with the capabilities for modeling and simulating finite automata that have the ability to synchronize with Reference Nets.

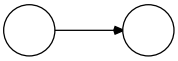
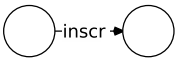
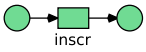
At first sight it may seem odd to use finite automata next to Petri nets because they actually provide a similar perspective. Simple Petri nets already, and Reference Nets a fortiori, are more expressive than finite automata. Still, finite automata offer an intuitive reflection of a systems state, which is especially helpful when multiple levels of abstraction of a complex system shall be covered.

### 4.1 Requirements

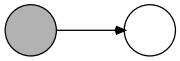
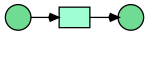
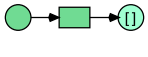
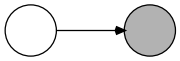
We derive the following requirements from our goal of extending Renew with support for modeling finite automata and for simulation on the basis of Reference Nets. A concept of finite automata for simulation and synchronization with Reference Nets is necessary. For this purpose, the finite automata modeling language is to be extended. Also, during execution, an automaton's state shall be *visually* inspectable. Furthermore, the editor has to provide usability features to support efficient modeling.

Table 1: Mapping of finite automata and Reference Net constructs

(a) Static view

| Automata | Reference Net |
|---|---|
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

(b) Dynamic view

| Reference Net | Automata |
|---|---|
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

## 4.2 Concept and Design

The modeling of finite automata is provided by the FA plugin, which was available prior to this work. In its previous version it was restricted to offer some capabilities for drawing finite automata, without the possibility of simulation. As Renew is natively capable of simulating Petri and Reference Nets, we extend the existing FA plugin with simulation and synchronization capabilities.

*Reference Net Based Semantics* As already mentioned in Section 3, we propose to provide semantics via a mapping to Reference Nets in order to exploit their synchronization features. Additionally, we have to manually implement the visual representation of the simulation state so that the user has feedback in the original representation of the modeling technique (i.e. the state of the simulation of the Reference Net at runtime has to be mapped back on the automaton).

The mapping of finite automata to Petri nets is straightforward as displayed in Table 1a. A finite automaton's state is mapped to a Reference Net's place. We do not distinguish between regular and end states, because we are more interested in the dynamic behavior than in the investigation of the formal language an automaton generates. Start states are mapped to places that are initially marked with a black token (represented as `[]`). A state transition between two states is mapped to a transition that is connected to the places representing the corresponding states. This also holds for the special case of a self-loop. The inscriptions of the state transitions are mapped to inscriptions of net transitions.

In addition to the static mapping, we need to provide a visualization of the current simulation state by mapping the dynamics of the Reference Net back to the automaton. This mapping is depicted in Table 1b. An empty place is

mapped to a non-activated state. A place that contains a token is mapped to an activated state (i.e. the current state in a DFA), which we highlight by a gray filling. The firing of a transition (represented by a highlighted transition) is mapped to a red highlighted state transition.

*Synchronization of Finite Automata and Reference Nets* We extend finite automata by synchronous channels to couple them with Reference Nets (see Section 2.3). The direct mapping of inscriptions from state transitions to inscriptions on Reference Net transitions makes the use of synchronous channels and other Reference Net inscription types possible. This is due to the compiler, which is handling the inscriptions as if they were attached to a Reference Net. Consequently, Reference Nets or other modeling techniques that provide compatible synchronous channels can synchronize with finite automata in the same way.

### 4.3 Modeling Capabilities

In this section we focus on the support for modeling finite automata with the FA plugin. This comprises the drawing capabilities of the editor and the means for the annotation of FA constructs with Java inscriptions.

*Basic Modeling* The FA plugin can be used to model finite automata using the constructs that are depicted in Table 1a: start states, end states, start-end states, neutral states and state transitions (between states). Because the FA plugin is implemented as a Renew plugin it inherits modeling capabilities and usability features. These features are complemented with some useful improvements such as arc drawing handles or interchangeable state representations. The FA plugin supports modeling of nondeterministic finite automata as well as deterministic finite automata. We limit the quantity of start states to one, so that the semantics remains simple. Drawings may be ex- and imported to the JFLAP[2] format and to a textual representation in addition to the standard formats of Renew.

*Reference Net Inscriptions* The inscriptions of our finite automata are inherited from Reference Nets. Consequently, the syntax and the semantics are similar. The inscription types, that are relevant for the simulation of finite automata along with Reference Nets, are depicted in Figure 5. In general, more inscription types are available. These are described in detail in [22, p. 48 ff.].

Nevertheless, modelers of our extended finite automata have to consider some differences to the modeling of Reference Nets. Reference Net inscriptions can only be – in a sensible way – attached to state transitions. This means, Reference Net inscriptions that are attached to states are not handled by the compiler and thus these are *not* executed by the Renew simulator in contrast to Reference Nets where Java inscriptions to places are possible. Another important consideration is that variables in finite automata can not be stored and only accessed arc-locally (see Section 3.3). We prototypically implemented the feature to provide

[2] JFLAP [18] is a modeling tool that is mainly concerned with teaching basics of theoretical computer science.

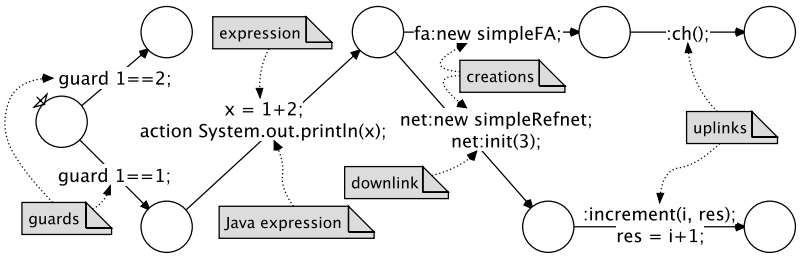



Figure 5: Available inscription types for finite automata

modelers with the capability to initialize the finite automata with diagram-global variables already. So far, this implementation is experimental.

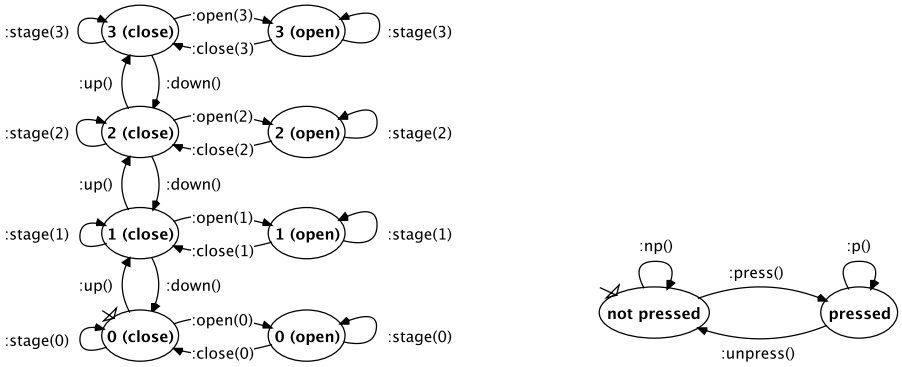
We forgo the description of each inscription that is illustrated in Figure 5 because most of them are inherited from Reference Nets. The synchronous channel inscriptions are of specific relevance in this context. These can be used analogously to the Reference Nets (see Section 2.2). Besides of calling uplinks of other models, our extended finite automata are also capable of providing uplinks.

### 4.4 Formalism Implementation

The implementation of the formalism touches multiple layers of Renew: (1) graphical layer, (2) shadow layer and (3) compilation layer. We identify the following four main tasks: (a) implement a compiler that translates the graphical representation into a non-graphical data structure (called *shadow net*), (b) develop a mapping from finite automata concepts to Reference Net concepts, (c) implement a second compiler that compiles the finite automata constructs according to the previously developed mapping to Reference Net constructs (called *compiled nets*) and (d) implement the return of feedback from simulation events to a graphical representation. With the completion of these four tasks, the FA plugin for Renew provides the concurrent simulation and synchronization of finite automata and Reference Nets. The FA plugin as described in this section is part of the Renew package referenced in Section 2.1.

## 5 Lift Application

In this section we present a slightly larger example for the use of multi-formalism execution. The presented system is a model of a lift, which is partly created as automaton and partly as Reference Net. The lift can move up- and downwards between four floors and open its door on each of the floors. It is possible to call the lift on a floor by pressing a button on the respective floor. For reasons of simplicity, we assume that pressing the button on one floor has the same effect as pressing the button for the destination floor from inside the lift. Thus, we do not model the touch panel inside the lift explicitly. In this scenario the status of




(a) Lift automaton (b) Floor status automaton

Figure 6: Automata models of the lift example

the lift and the status of the button on each floor is simple. These are systems without any concurrency and with a defined state. Therefore, these components are modeled as finite automata. The complex part in this scenario is the control mechanism that ensures a reasonable serving strategy. Consequently, it is realized as a Reference Net. Thereby, we use adequate modeling techniques for all parts. The control mechanism as independent component can be exchanged by another version that uses a different serving strategy.

*Automata Models* The automata models as depicted in Figure 6 have multiple state transitions attached with uplink inscriptions in order to be synchronized with the Reference Net. With the loop at each state, it is possible to implement state dependent behavior (e.g. opening the door is only possible when the button on the respective floor is pressed).

*Reference Net Control* The Reference Net in Figure 7 implements a lift control that serves a floor when the lift is requested by a pressed button and prefers to keep the movement direction. On the top left hand side of the net, instances of the automaton models are created (one instance of lift, four instances of floor). On the top right hand side there are four transitions that trigger a push of the button in one of the floors. For technical reasons it is not possible to fire state transitions in automaton models, yet. Hence, these transitions in the control net are a simple solution to overcome this technical restriction.

The actual lift control is divided vertically into the four floors. Each floor consists of three to four control places (green places) representing the lift on the respective floor on its way downwards (leftmost place), on its way upwards (rightmost place) or with an open door (places in the center). Changes to the lift and floor models or state queries to these models require access to the lift and floors places. This is achieved by using virtual places.

The lift is initialized with closed doors in the ground floor. With Transition *o* in the bottom, the door is opened initially. Transition *cu0* closes the door of the

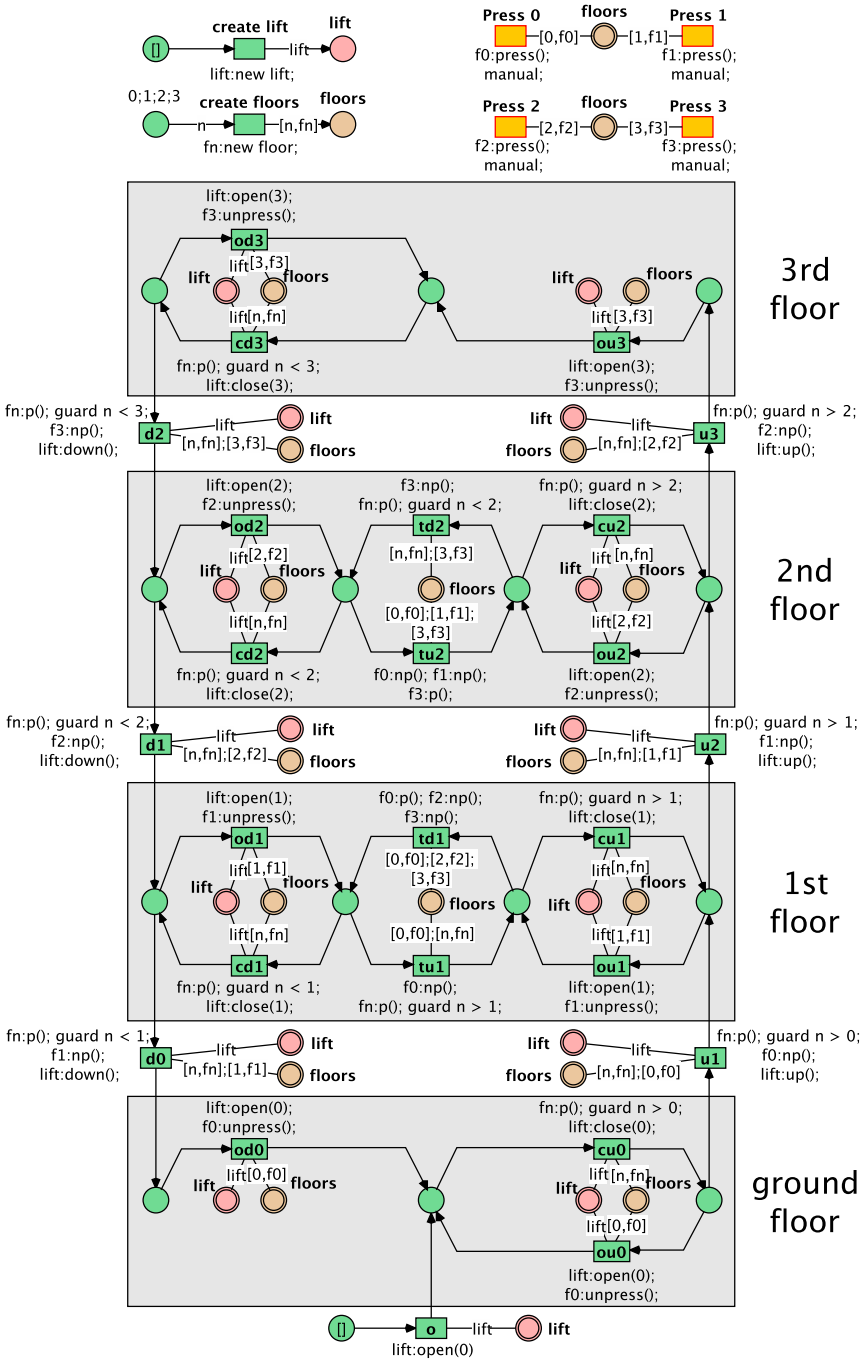



Figure 7: Reference Net for controlling the lift

lift (`lift:close()`), which can then start its way upwards. This is only possible, when the lift was requested on one of the higher floors (i.e. the button is pressed on one of the higher floors, `fn:p(); guard n > 0`). If the door is closed the lift may move upwards one floor with Transition *u1* (`lift:up()`) if the lift was requested from a higher floor (`fn:p(); guard n > 0`) and it was not requested on the same floor (e.g. somebody wants to hop on, `f0:np()`).

With this mechanism, the lift can move upwards until it reaches a floor where the button is pressed (e.g. on the second floor). There, it has to open the door (`lift:open(2)`) and the request for the floor is reset (`f2:unpress()`).

On the first and second floor the lift has two places representing an open door to distinguish the two directions. The lift is allowed to change its direction (with Transitions *td1*, *tu1*, *td2*, *tu2*) if there is no request in the current direction (`f3:np()`) and a request in the other direction (`fn:p(); guard n < 2`).

Analogously to the right hand side of the lift control, the left side implements the controlling of the lift moving downwards.

## 6 Related Work

In this section we briefly relate our proposal to work in the area of multi-formalism modeling and execution.

Zeigler [30] proposes the multifacetted modeling methodology, which is an approach to simulation modeling by integrating multiple models. The hereby used *Discrete Event System Specification* (DEVS) formalism is capable of constructing coupled models that are composed of atomic DEVS models. In his work an abstract simulation concept for the DEVS formalism was developed. The concept claims to couple serveral simulators for each component in a system of systems to facilitate simulation using a global coordinator for synchronization. Unlike our approach, Zeigler's demonstrations are rather abstract. Our approach is also different from his as we attach importance to concurrent simulation, in particular to true concurrency of Renew. We propose to work with Reference Nets instead of DEVS.

Lara et al. [23] introduce a tool that supports the combined use of multi-formalism modeling and meta-modeling, called AToM$^3$. Due to the definition of *graph grammar models*, formalisms can be transformed into an appropriate formalism for which simulation is already supported. They suggest the DEVS formalism as the central modeling formalism that can be universally used for simulation purposes. AToM$^3$ also supports code generation, a meta-modeling layer that can be used to model formalisms in a graphical manner, and the possibility to transform models by preserving the behavior [1]. In contrast to our approach, Lara et al. focus on providing meta-modeling and model-transformation features. The transformed models have to be simulated in an external environment.

Möbius is a tool for modeling and simulating systems composed of multiple formalisms. The project focuses on extensibility by new formalisms and solvers, which is demonstrated in [8]. An *abstract functional interface* is implemented,

which transforms models to framework components to allow for addition of formalisms and interaction between models. Their approach differs from ours in the way of having an overall system state. The Möbius tool enables selective sharing of model states, so that solvers (i.e. simulators) are able to access them.

One practical implementation of a multi-formalism modeling and simulation concept was done by The GEMOC Initiative [13]. This initiative's vision is to advance the research on the coordinated use of modeling languages. They recognized a problem in the unavailability of a generic runtime service for multiple modeling languages. GEMOC Studio is proposed as a tool to create meta-models for both the representation and the operational semantics of modeling languages. Created models of multiple languages can then be executed in coordination, while being debugged and graphically visualized. They use the *Behavioral Coordination Operator Language* (BCOoL [24]) to explicitly specify the coordination patterns between heterogeneous languages. The used execution engine operates as a coordinator of multiple language-specific engines.

Frank [11] proposes a method for multi-perspective modeling that extends the classical approach (of e.g. UML) to conceptual modeling by including the organizational environment. He is also interested in tool support and states the following requirement we share: "A tool environment for enterprise modeling should allow for creating multi-language diagrams, i.e., diagrams that integrate diagrams of models that were created with different DSML" [11, S. 946]. The linking of techniques he proposes is established on the level of a meta-model, but the method lacks an environment to properly support the execution. "However, there are other paradigms that come with specific advantages, too. [...] Languages used for creating simulation models would allow for supplementing enterprise models with simulation features. Petri Nets provide mature support for process analysis and automation" [11, S. 960]. We find the combination of such approaches to modeling language engineering with the provision of operational semantics for a dedicated execution environment most promising.

Jeusfeld [17] proposes the linking of multiple perspectives through declarative constraints in the context of meta-modeling domain-specific languages for the ADOxx platform. He distinguishes the relation between model and external environment from the internal model validity and focusses on the latter. The constraint language is used to define a Petri net firing rule and to sketch a firing rule of BPMN constructs. However, he does not show the combined execution of these formalisms.

There are other researchers who have tried to combine various formalisms with Petri nets. The set of all firings of a Petri net can be considered to be a language. The research of automata and Petri nets as language descriptions has led to the control of Petri nets by (finite) automata [4]. The results suggest that controlling Petri nets through finite automata can be beneficial. A combination of finite automata and high-level Petri nets can be seen as a vertical composition, as both techniques basically provide the behavioral modeling perspective according to Krogstie [19], just on another level of abstraction.

While first the idea of language intersections were of interest, the idea of a *controller* was used in the context of application modeling [28]. A lift system was modeled, however, just the control was addressed and no other modeling techniques were applied. In other research, flexible manufacturing systems (FMS) were used to demonstrate central aspects of control theory (cf. [14] for discrete event system discussion). Most of the authors' work concentrates on techniques that provide one specific modeling perspective. A production system can be controlled by a simple model to ensure that no deadlocks can occur [9,15].

Overall our approach presented here allows to reduce the cognitive load of modelers. The complexity can be reduced by using an appropriate modeling technique, like finite automata, workflows or other modeling techniques like BPMN, eEPCs, Activity Diagrams, etc. However, each formalism needs to be connected via the synchronous channels to be executed within our environment. Systems complying to our interfaces could also mimic our approach if sufficient support for the execution of modeling techniques is available, for example with CO-OPN/2 [2] or the Zero-safe net formalism [3]. Our illustrations of the principle usage in Sections 4 and 5 can be seen as proof of concepts.

## 7 Conclusion

In this contribution we conceptually present how various formalisms can be used together not only for modeling, but also for simulation while preserving their original representation.

In our opinion, the benefit of using multiple formalisms for the modeling of complex systems can be increased by providing a solution for the simulation of these formalisms. That does not mean that we just intend to transform the models of multiple formalisms into one single formalism, but rather provide simulation feedback for the original representation of the models. Therefore, the various formalisms are mapped to Reference Nets and the simulation events are returned to the models original representation (see challenge (2) in Section 1).

We set up the thesis that the synchronization and data exchange between models of various formalisms should not be achieved through the development of several individual coupling mechanisms. Instead, we represent the opinion, that this should be achieved through *one* coupling mechanism.

We propose the generic coupling of models using the synchronous channels on the basis of Reference Nets (see challenge (1) in Section 1). Regardless of whether the models are of various or the same formalism. Synchronous channels allow for the synchronization of several models, and data exchange in all directions.

As a proof-of-concept we practically demonstrate how this approach can be implemented for the coupling of finite automata and Reference Nets.

In the close future we plan to improve the various concepts and implementations of our approach to support modeling techniques with variables for formalisms that are not capable of storing references by themselves. Furthermore, we investigate how to support the development of techniques through meta-modeling and to provide a close adaption to the simulation environment of Re-

NEW, e.g. to model the techniques themselves, the drawing tools and their operational semantics [26]. In the context of the above mentioned investigation, among others we will develop a RENEW plugin that provides the modeling and simulation of statecharts.

## References


1. AToM$^3$ website. `http://atom3.cs.mcgill.ca`, accessed: 2017-01-19
2. Biberstein, O., Buchs, D., Guelfi, N.: Object-oriented nets with algebraic specifications: The CO-OPN/2 formalism. In: Agha, G., de Cindio, F., Rozenberg, G. (eds.) Concurrent Object-Oriented Programming and Petri Nets, Advances in Petri Nets. Lecture Notes in Computer Science, vol. 2001, pp. 73–130. Springer (2001)
3. Bruni, R., Montanari, U.: Zero-safe nets: The individual token approach. In: Parisi-Presicce, F. (ed.) Recent Trends in Algebraic Development Techniques, 12th International Workshop, WADT'97, Tarquinia, Italy, June 1997, Selected Papers. Lecture Notes in Computer Science, vol. 1376, pp. 122–140. Springer (1997)
4. Burkhard, H.D.: Control of Petri nets by finite automata. Annales Societatis Mathematicae Polonae Series IV: Fundamenta Informaticae VI.2, 185–215 (1983)
5. 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)
6. Cabac, L., Haustermann, M., Mosteller, D.: Software development with Petri nets and agents: Approach, frameworks and tool set. Science of Computer Programming (2017), under review
7. Christensen, S., Hansen, N.: Coloured Petri nets extended with channels for synchronous communication. In: Robert, V. (ed.) ATPN. Lecture Notes in Computer Science, vol. 815, pp. 159–178. Springer-Verlag (1994)
8. Courtney, T., Gaonkar, S., Keefe, K., Rozier, E.W.D., Sanders, W.H.: Möbius 2.3: An extensible tool for dependability, security, and performance evaluation of large and complex system models. In: 2009 IEEE/IFIP International Conference on Dependable Systems Networks. pp. 353–358 (June 2009)
9. Ezpeleta, J., Colom, J.M., Martínez, J.: A Petri net based deadlock prevention policy for flexible manufacturing systems. IEEE Trans. Robotics and Automation 11(2), 173–184 (1995)
10. Ezpeleta, J., Moldt, D.: A proposal for flexible testing of deadlock control strategies in resource allocation systems. In: Pahlavani, Z. (ed.) Proceedings of International Conference on Computational Intelligence for Modelling Control and Automation, in Vienna, Austria, 12–14 February (2003)
11. Frank, U.: Multi-perspective enterprise modeling: foundational concepts, prospects and future research challenges. Software & Systems Modeling 13(3), 941–962 (2014)
12. Friedrich, M., Moldt, D.: Introducing refactoring for reference nets. In: Cabac, L., Kristensen, L.M., Rölke, H. (eds.) Petri Nets and Software Engineering. International Workshop, PNSE'16, Toruń, Poland, June 20-21, 2016. Proceedings. CEUR Workshop Proceedings, vol. 1591, pp. 76–92. CEUR-WS.org (2016)
13. GEMOC Initiative website. `http://gemoc.org/index.html`, accessed: 2017-04-02

14. Giua, A., Seatzu, C.: Petri nets for the control of discrete event systems. Software and System Modeling 14(2), 693–701 (2015)
15. Hu, H., Liu, Y., Zhou, M.: Maximally permissive distributed control of large scale automated manufacturing systems modeled with Petri nets. IEEE Trans. Contr. Sys. Techn. 23(5), 2026–2034 (2015)
16. Jacob, T., Kummer, O., Moldt, D., Ultes-Nitsche, U.: Implementation of workflow systems using reference nets – security and operability aspects. In: Jensen, K. (ed.) Fourth Workshop and Tutorial on Practical Use of Coloured Petri Nets and the CPN Tools. University of Aarhus, Department of Computer Science, Ny Munkegade, Bldg. 540, DK-8000 Aarhus C, Denmark (Aug 2002), dAIMI PB: Aarhus, Denmark, August 28–30, number 560
17. Jeusfeld, M.A.: Semcheck: Checking constraints for multi-perspective modeling languages. In: Karagiannis, D., Mayr, H.C., Mylopoulos, J. (eds.) Domain-Specific Conceptual Modeling, Concepts, Methods and Tools, pp. 31–53. Springer (2016)
18. JFLAP website. `http://www.jflap.org`, accessed: 2017-01-18
19. Krogstie, J.: Perspectives to Process Modeling, Studies in Computational Intelligence, vol. 444, pp. 1–39. Springer Berlin Heidelberg, Berlin, Heidelberg (2013)
20. Kummer, O.: Referenznetze. Logos Verlag, Berlin (2002)
21. 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
22. 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 (Jun 2016), `http://www.renew.de/`
23. de Lara, J., Vangheluwe, H.: AToM$^3$: A tool for multi-formalism and metamodelling. In: Kutsche, R., Weber, H. (eds.) Fundamental Approaches to Software Engineering, 5th International Conference, FASE 2002, held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2002, Grenoble, France, April 8-12, 2002, Proceedings. Lecture Notes in Computer Science, vol. 2306, pp. 174–188. Springer (2002)
24. Larsen, V., Ezequiel, M.: BCOol: the Behavioral Coordination Operator Language. Theses, Université de Nice Sophia Antipolis (Apr 2016)
25. Milner, R.: Elements of interaction - turing award lecture. Commun. ACM 36(1), 78–89 (1993)
26. Mosteller, D., Cabac, L., Haustermann, M.: Integrating Petri net semantics in a model-driven approach: The Renew meta-modeling and transformation framework. T. Petri Nets and Other Models of Concurrency 11, 92–113 (2016)
27. Petri, C.A.: Kommunikation mit Automaten. Dissertation, Schriften des IIM 2, Rheinisch-Westfälisches Institut für Instrumentelle Mathematik an der Universität Bonn, Bonn (1962)
28. Reisig, W.: Embedded system description using Petri nets. In: Kündig, A.T., Bührer, R.E., Dähler, J. (eds.) Embedded Systems: New Approaches to Their Formal Description and Design, An Advances Course, Zürich, Switzerland, March 5-7, 1986. Lecture Notes in Computer Science, vol. 284, pp. 18–62. Springer (1986)
29. Valk, R.: Petri nets as token objects - an introduction to elementary object nets. In: Desel, J., Silva, M. (eds.) 19th International Conference on Application and Theory of Petri nets, Lisbon, Portugal. pp. 1–25. No. 1420 in Lecture Notes in Computer Science, Springer-Verlag, Berlin Heidelberg New York (1998)
30. Zeigler, B.P.: Multifacetted Modelling and Discrete Event Simulation. Academic Press Professional, Inc., San Diego, CA, USA (1984)