Graphical Languages for Functional Reactive Modeling based on Petri nets

David Mosteller, Michael Haustermann, and Leonie Dreschler-Fischer

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

Abstract The ideas of functional programming have become well adopted by various fields of information science due to their useful properties regarding their application to nowadays complex and heterogeneous IT environments. The graphical modeling tools that support us in keeping up with mentioned complex environments often lack the functional perspective, especially when it comes to formal operational semantics. The Rmt approach (Renew Meta-Modeling and Transformation) provides a conceptual framework for the development of domain specific modeling languages with transformational semantics based on mappings to Petri net components. Petri nets provide a dynamic perspective of systems based on states and state transitions. With the Rmt approach this perspective is leveraged to the abstract perspective of domain specific modeling languages. In this contribution Petri net components are developed that capture functional properties for the specification of domain specific modeling languages using operational semantics. The benefits of this approach are the means to develop graphical formalisms utilizing formal operational semantics and data processing by using a functional abstraction.

Keywords: Metamodeling, Petri nets, Reference Nets, Functional Programming, Reactive Programming

1 Introduction

A popular perspective in the recent past has been the notion of functional reactive systems. With the emergence of ubiquitous systems and the *Internet of Things* (IOT) we are surrounded by software systems that constantly react to events in our environment. Petri nets are very well suited to model such types of applications as they share many characteristics with the functional reactive paradigm. They support synchronous as well as asynchronous message passing, have a precise notion of locality and they are inherently concurrent. Although they are state-based in nature, they do support taking a functional perspective.

In this contribution we provide a conceptual basis to develop executable graphical languages for modeling reactive systems based on functional system decomposition. As a result of this contribution, we provide sophisticated net components that apply the functional perspective. Our novel approach enables

the application of known concepts of functional programming languages such as higher-order functions and recursion by utilizing the nets-within-nets paradigm to ensure the referential transparency. The basis of this work is provided by the formalism of Reference Nets [10], a variant of high-level Petri nets, which will be introduced in Section 2. We will use this formalism to demonstrate its usefulness for modeling functional reactive systems and complex data structures in Section 3. Once this is achieved we can turn to the development of abstract (domain specific) modeling languages (DSML) that are more useful for end users by using the RMT approach (RENEW Meta-Modeling and Transformation approach, [13]) in Section 4. An example of such a DSML suited for modeling a home automation scenario will be presented in Section 5. Section 6 presents a comparison of our approach with related work before we summarize our results in the conclusion (Section 7).

2 Reference Nets

The Reference Net formalism is a high-level Petri net formalism with support for modeling complex data structures, remote synchronization and Java integration. Some of the core features of Reference Nets will be presented in the following as they are relevant for this contribution. The Java features will not be discussed in detail as they are of minor interest in this context. A thorough introduction to Reference Nets with Java integration is in the RENEW manual [11]. RENEW provides full support for modeling and execution of Reference Nets and other modeling languages. An example of the application to software engineering can be found in the latest research paper on RENEW [2].

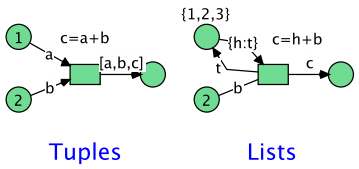


Figure 1: Using collections in Reference Nets

With high-level Petri nets, data in the form of colors is unified using unification expressions on the transitions with regard to the variable bindings on the edges. Reference Nets are not the only high-level Petri net formalism with support for collection types, however, Figure 1 shows an example of how they are realized there. On the left side the variable `c` is calculated from the inputs `a` and `b` and all of the three variables are outputted in a tuple to the place on the right side. Tuples can be hierarchically nested to perform complex operations on them by the means of unification. Lists, as depicted in Figure 1 on the right, are even more powerful as they permit iterative or recursive processing. In fact, in

the depicted example the transition may fire three times, each time computing a new value for variable `c` from its head element (`h`) and variable `b`, which is only read and not removed by using a test arc.

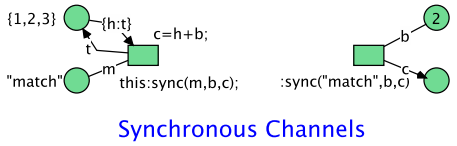


Figure 2: Synchronous channels of Reference Nets

Synchronous channels, as exemplarily depicted in Figure 2, control the firing of multiple transitions as one synchronized event. A synchronous channel consists of a pair of downlink (caller) and uplink (callee), so the reference must be known on one side of the channel. They may have arguments to transport information similar to the call of a function, but they have a slightly different notion as the unification of arguments enables a bidirectional exchange of data. In the depicted example the downlink (left side) calls the right side (uplink) in the local net instance (`this`). The first argument (`m`) on the downlink can be unified with the String (`"match"`) on the uplink. The second argument (`b`) receives its actual parameter from the right side and the calculation of variable `c` is performed on the downlink transition on the left side before it flows through the channel to the uplink.

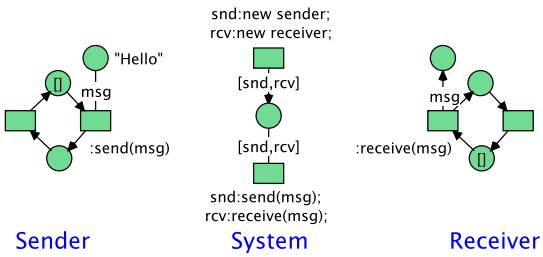


Figure 3: Dynamic hierarchies: nets-within-nets [19]

Clearly, the expressiveness of synchronous channels is quite powerful. However, they unfold their real potential in combination with dynamic hierarchies and the *nets-within-nets* paradigm [19]. Using the `new` syntax shown in the upmost part of Figure 3 Reference Nets can create instances of other net patterns. This enables dynamic hierarchies up to an arbitrary level of nesting. The depicted example is restricted to a level of two and models a message sending scenario. It shows a system of three components. In the center is a system net,

which manages the instances of senders (left) and receivers (right). The latter communicate through the synchronous channels (`send` and `receive`) of the system net. While this configuration models a small application scenario, the next section goes a step further and demonstrates how Reference Net systems may be applied to model a more comprehensive application.

3 Functional Reactive Systems

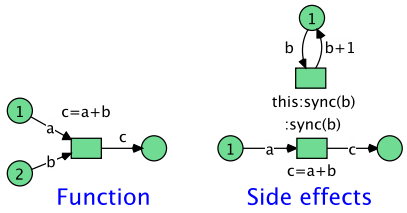


Figure 4: A functional perspective on Reference Nets

Recall the net on the left side of Figure 4, which we have discussed in the previous section. It may be considered as a function that computes an output from a number of inputs. It even pertains functional key characteristics, like referential transparency and immutability of objects. However, we already know that this is only half of the truth, so we must restrict ourselves to pertain these properties, which we will consider in the following. Synchronous channels provide us with the means to explicitly model side effects and control their impact on the computation (cf. right side of Figure 4). If we wish the data passed along the synchronous channel to be immutable, we simply design our data structure to allow read access only.

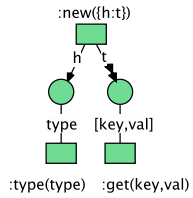


Figure 5: An immutable named key-value pair data structure

The simple Reference Net model in Figure 5 implements a key-value pair data structure comparable to JSON or other similar data types. It is created by calling the `new` channel with a list as argument containing a type identifier as

head and a list of key-value pairs as tail. It uses a flexible arc (double arrow) we have not introduced up to now, which "unpacks" all of the key-value pairs to the output place. From there the individual values can be queried through the `get`-channel but due to the test arc, read only access is permitted. An example of its usage will be demonstrated shortly.

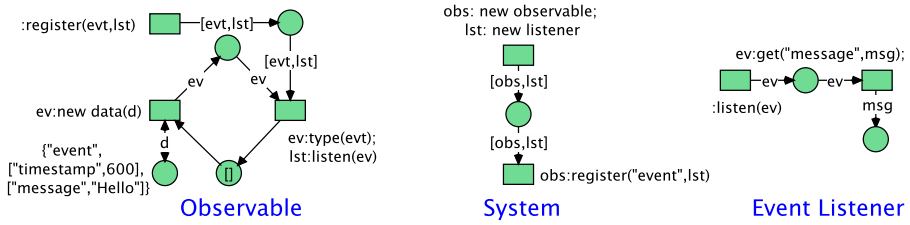


Figure 6: A reactive system

In order to form a reactive system it must gain the ability to react upon events that are triggered by its environment. The example of sender and receiver from Figure 3 in Section 2 contained patterns of this behavior. It has been revisited in Figure 6 and extended by an event passing mechanism. The system component in the center remains mainly the same, only the listener is now registered with the observable to listen for occurring events. The observable stores the event listener together with the respective event type. As a first step in the cyclic process an instance of the key-value data net is created with a new event from the pattern at the leftmost inscription. It contains a type identifier ("event"), a timestamp (600) and a message ("Hello"). Each time this event occurs the listener is notified by the synchronous transition (`listen`). The actual reactive system is modeled as depicted by the rightmost part (event listener). All it does is store the message text to the rightmost place. Note that multiple events can be processed concurrently at a time by the event listener.

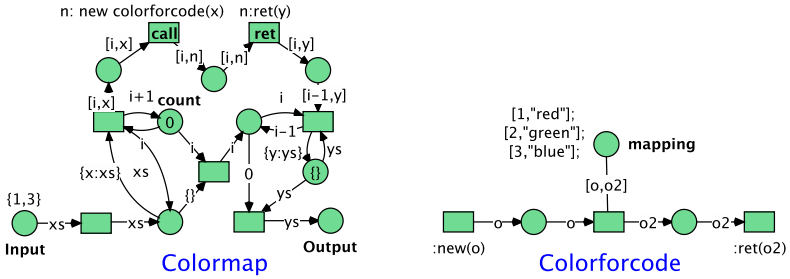


Figure 7: Higher-order functions

When nets are interpreted as functions, the nets-within-nets paradigm with the possibility to instantiate new nets enables the creation of higher-order functions. Figure 7 shows an example of a higher-order function implemented as a Reference Net. The left side shows a net that maps integer values to colors using the net on the right side. The creation of a new instance of the `colorforcode` net ensures the referential transparency for every element in the list.

The map component has an input/output behavior just like the function that was depicted in Figure 4. Note the input and output elements on the map component, one consuming a list (`xs`) as an input to the computation, the other producing a list (`ys`) as a result. The input list is processed successively, taking into account the position of the elements in the list, so that the output order corresponds to the input order. The unpacking is done by separating the head element (`{x:xs}`) while counting the elements on Place `count` at the same time. While the (un)packing of the list is sequentialized, the actual function calls execute in parallel. The function is applied by firing the transition `call`, which creates a net instance of the net `colorforcode`. The result is retrieved from the net instance by the subsequent transition `ret`. The right part of the map component symmetrically collects the results while respecting the initial order. The `colorforcode` net returns names of colors for numbers according to the tuples in Place `mapping`.

We will see a more sophisticated implementation of a reactive system using higher-order functions in Section 5. First, we introduce the means to develop graphical modeling languages as abstractions of Reference Net systems in the next section.

4 Functional Reactive DSML

In this section we will demonstrate the development of domain-specific languages with the Rmt framework specifically with respect to modeling functional reactive systems. The development of an executable modeling language with the Rmt framework consists mainly of two steps, the development of constructs for the graphical editor and the provision of operational semantics in the form of Petri net components [1, Chapter 5]. The following part gives a brief summary of creating a graphical editor with the Rmt framework. It is taken from our previous work [14], where we developed an executable formalism of BPMN. It was adopted to this context for demonstration purposes.

The Rmt framework (Renew Meta-Modeling and Transformation) [13] is a model-driven framework for the agile development of DSML. It follows concepts from software language engineering (SLE, [9]) and enables a short development cycle to be appropriately applied in prototyping environments. With the Rmt framework, the specification of a language and a corresponding modeling tool may be derived from a set of models, defined by the developer of a modeling technique. A meta-model defines the structure (abstract syntax) of the language, the concepts of its application domain, and their relations. The visual instances (concrete syntax) of the defined concepts and relations are provided

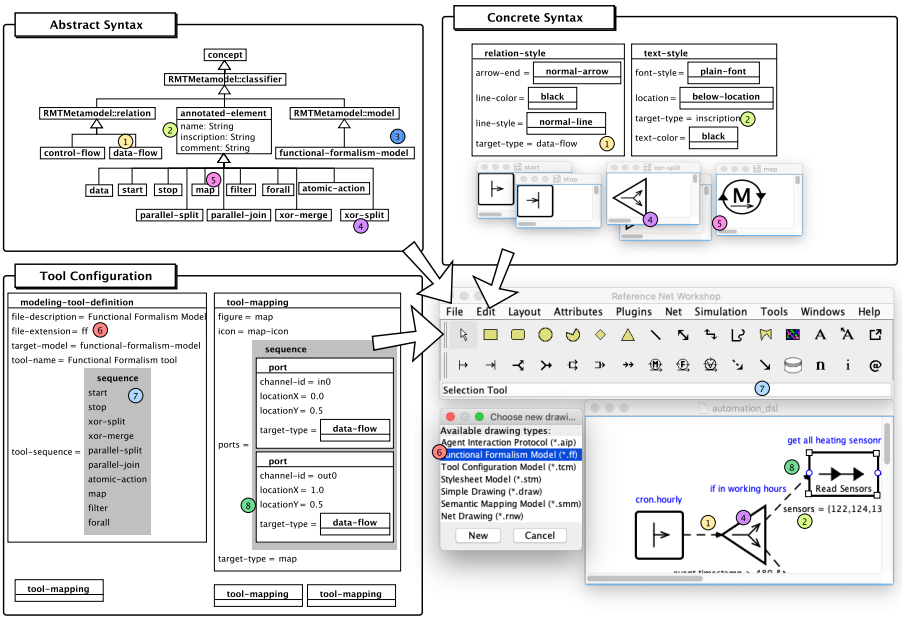


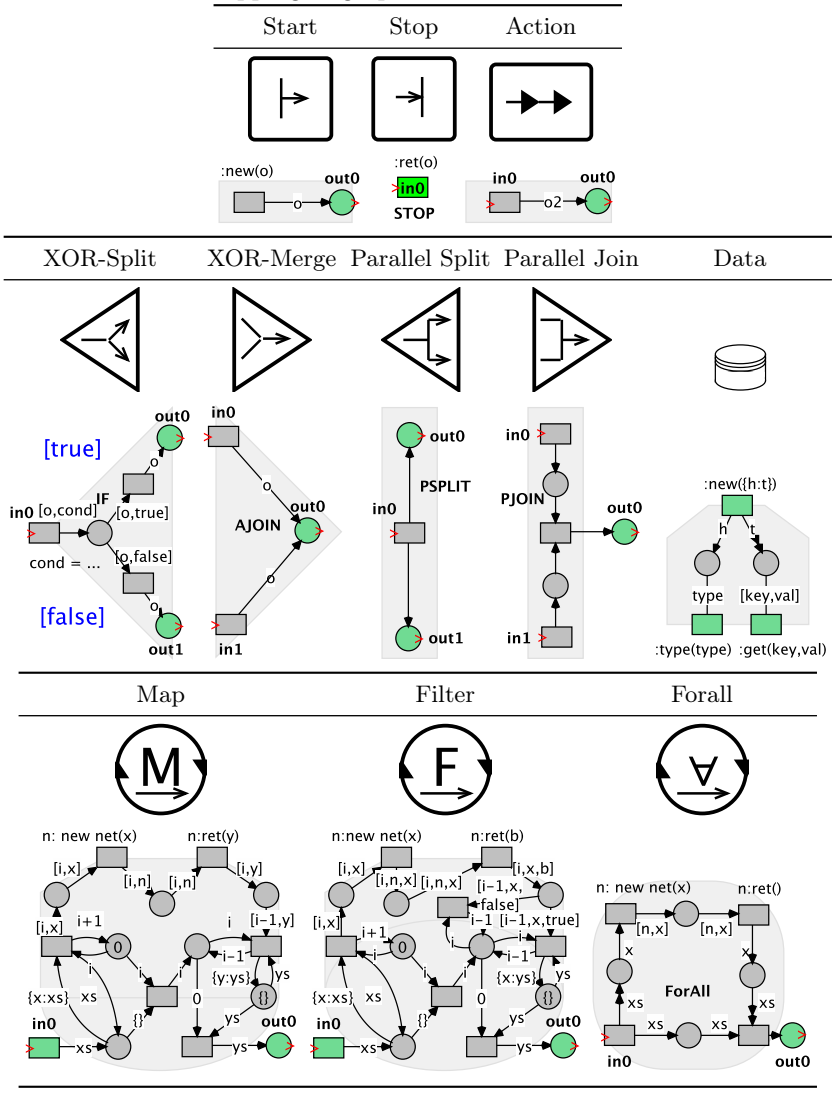
Figure 8: Excerpt from the RMT models for a home automation modeling tool

using graphical components from RENEW's modeling constructs repertoire. They are configurable by style sheets and complemented with icons and a tool configuration model to facilitate the generation of a modeling tool that nicely integrates into the RENEW development environment.

Figure 8 displays a selected excerpt from the models required for the home automation DSML, together with the tool that is generated from these models. The parts of the figure are linked with colored circles and numbers. The meta-model (upper left) defines the concepts for classifiers (4, 5) and relations (1) of the modeling language and the corresponding model (3). Annotations are realized as attributes of these concepts (2). The concrete syntax (upper right) is provided using graphical components, which are created with RENEW, as it is depicted for the XOR-split and the map component (4, 5). Icon images for the toolbar can be generated from these graphical components. The representation of the inscription annotation (2) and the data-flow relation (1) is configured with style sheets.

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

Table 1: Mapping of graphical and Petri net constructs



With these models, a tool is generated as a RENEW plug-in, as shown at the bottom right side of Figure 8. A complete example of the RMT models and additional information about the tools can be found in our article on the RMT framework [13].

Table 1 shows the *semantic mapping*, a mapping of graphical components to semantic Reference Net components. In combination with the previously described steps, this is all it needs in order to build models that are ready to be executed within the RENEW simulation environment. With the integrated sim-

ulation plugin [14] feedback from the simulation may be visualized directly in the representation of the original DSML. The first row of Table 1 shows the graphical components (above) and the semantic components (below) that initiate and terminate the execution of a DSML and also an (atomic) action. The second row contains elements that organize control flow and the data component (cf. Figure 5) and the third row contains the higher-order functions. The start component takes a variable as an argument on the `new` channel and the stop component terminates by returning a value. The values may be of arbitrary type, the variable (`o`) is inserted as a placeholder. The XOR-split component accepts a conditional statement (`cond`) that may be evaluated to a boolean value. The XOR-split component must be complemented by a closing XOR-merge component to merge the execution paths. The same holds for the parallel split and the parallel join components respectively, only, they do not branch conditionally but concurrently. The key-value data Reference Net, which we have introduced in the previous section, is included as a data storage component. Map, filter and forall are Reference Net implementations well known to the users of functional languages. They operate on lists and each of them applies a Reference Net (`net(x)`) in order to perform a certain task. The map component receives elements (`x`) as input and maps them to elements (`y`) as output. How this mapping is achieved, is subject to the applied net, the inputs and outputs of this component however are always of type list. This also holds for the filter component, however, the applied net maps arbitrary elements to boolean outputs in order to perform the filter operation. The forall component in contrast does not necessarily return any values at all, it merely guarantees that the computation on each input is completed by call of the `ret` channel with no arguments.

Net components were originally developed by Cabac [1, Chapter 5] and they are adopted from his work. The basic and control flow components, were used by Cabac to implement agent interactions (including a variant of the forall component), the data component and the components modeling higher-order functions are novel results.

5 Home Automation DSML

Once the abstract and concrete syntax are at hand the graphical editor may already be employed to draw basic models using the home automation DSML. The elements of the home automation DSML can be annotated with inscriptions. The semantic components may hold variables where the generator replaces a place holder with the value of the respective inscription.

Figure 9 shows the model of a home automation process. It models a simplified heating control unit for an office building. The process is triggered by an hourly cron job. During working hours the level of the heating system is set for each floor of the building according to the values of sensors, which are installed throughout the building. First, the sensor temperatures are read and then the values are filtered by office rooms (`sensor.roomtype`). The temperatures are subsequently mapped to a heating configuration and applied to the heating sys-

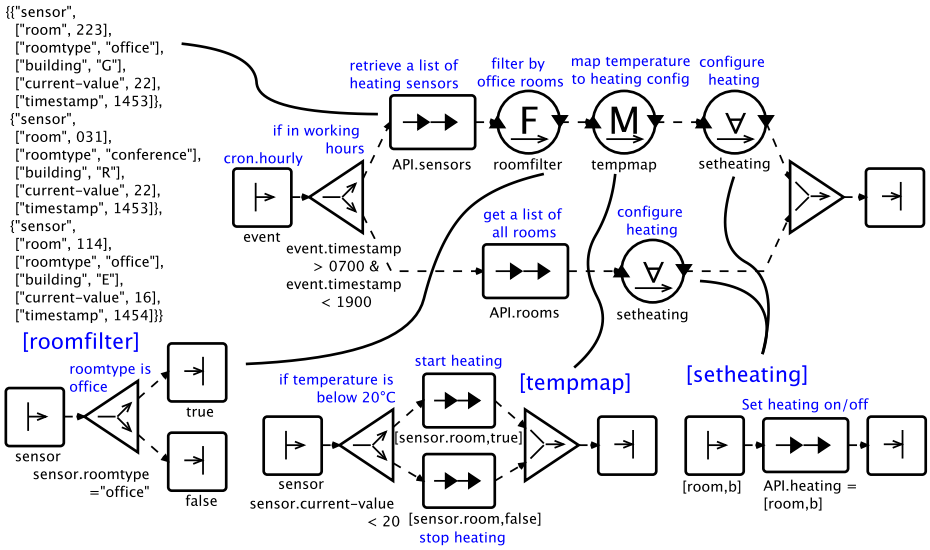


Figure 9: An example of a heating control modeled with a DSML for home automation

tem. In between office hours (`timestamp`) the heating system is shutdown for saving energy.

Together with the semantic mapping we developed in the previous section, the home automation DSML models can be executed within the Renew simulation environment. Using the net generator from the RMT framework the Reference Net depicted in Figure 10 was generated from the home automation model in Figure 9. It was extended with some inscriptions to become executable within the Renew simulation environment. The state of the home automation system becomes interactively inspectable and with the adaption to an appropriate home automation API the model could be used to control a home automation system.

6 Related Works

Graphical modeling techniques that apply a functional decomposition have a long tradition in computer science. The basis of functional block diagrams and its many variants are the interpretation of elements that are commonly depicted by rectangles as black boxes that transform inputs to outputs (cf. structured analysis and design technique [16]). Standard languages exist to simulate mature instances of such diagrams, for instance using Modelica [18] or Matlab Simulink [21], and even approaches to verification [22]. However, they are usually self-contained and do not account for the environment as in functional reactive programming.

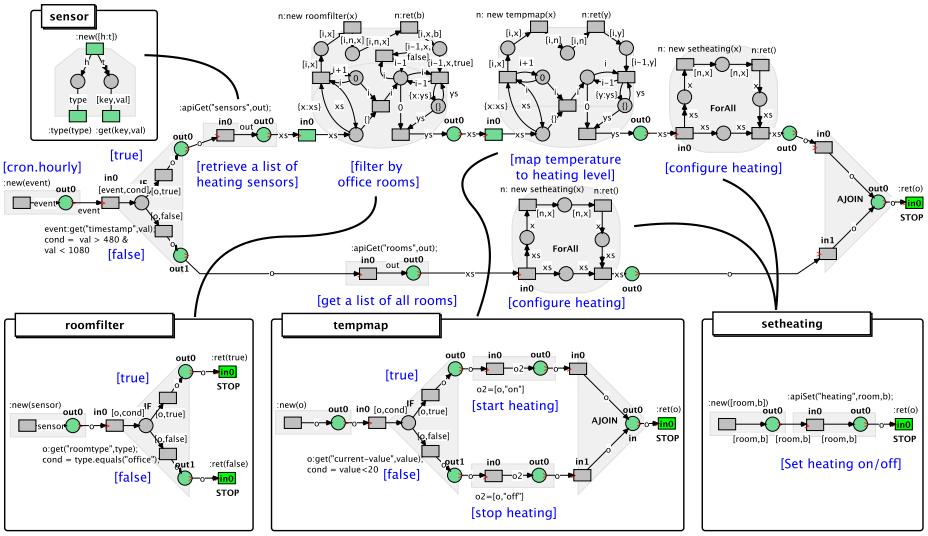


Figure 10: The Reference Net generated from the home automation DSML

There are many programming libraries that support functional reactive programming and some of them even provide a graphical notation, like the GraphDSL using Akka streams [4]. As an advancement of the map-reduce approach to big data processing, the Apache Spark project gains support for graphical flow-based programming [12]. They also target IOT and cyber-physical systems, however, the approach taken with this contribution is somewhat different as it focuses more on the operational semantics than on the processing of huge amounts of data.

Several frameworks exist that support developers in creating their own (domain specific) modeling languages based on metamodels (ADOxx [5]) and some of them even provide means for simulation or interactive execution (GEMOC Studio [3]). The execution semantics is usually coded within the modeling environment, while with the RMT approach it is provided by transformations to Petri nets. The Viatra eclipse project provides an event driven reactive framework for model transformations [20], which is an interesting approach for Petri net transformations.

Petri nets in general already offer a functional perspective due to their locality principle. For complex applications a powerful inscription language is necessary. Petri net formalisms often choose functional languages for this purpose to maintain the perspective. The Coloured Petri Nets formalism [8] for example uses ML as inscription language. To address the issue that simulators still have problems with side effects, the Curry-Coloured Petri Nets formalism uses a purely functional language to prevent side-effect related problems and logic program evaluation for the transition binding search [17]. In comparison to these formalisms, Reference nets allow the creation of dynamic hierarchies using the

nets-within-nets paradigm. This makes the implementation of advanced concepts from functional languages possible, such as recursion and higher-order functions while maintaining referential transparency.

Concerning the transformation based approach one may criticize that it is not actually functional, because it does not allow the combination of functions in the sense of currying or function composition. This would require a mechanism to modify the net structures at run-time, like higher-order Petri nets [7,6] or reconfigurable nets [15]. However, no such thing is currently supported by the Reference Net formalism and it would involve quite an effort to bring these ideas together with the high-level concepts of Reference Nets.

7 Conclusion

Software models created from a composition of functional components can be employed to support the undestanding of complex reactive environments. They may be executed within the Renew simulator for inspection and used as control mechanism, similar to a workflow execution system. In Section 2 we have laid the foundation by the means of Reference Nets. It was demonstrated in Section 3 how Reference Nets can be employed to model functional reactive systems and complex data structures. We introduced the means for creating a modeling language that supports this paradigm and developed functional net components in Section 4 to formalize their semantics. This was used to build a simple example of a home automation DSML that may be used to generate a Reference Net model to be executed within the Renew simulation environment. It can be extended with further constructs to extend the applications of the home automation DSML.

In the future we would like to advance the approach to perform formal analysis most probably using state space analysis, which is interesting because only parts of the environment are visible. Another interesting idea is the extension of the Reference Net formalism to support real higher-order functions and the combination of functions.

References

1. Cabac, L.: Modeling Petri Net-Based Multi-Agent Applications, Agent Technology – Theory and Applications, vol. 5. Logos Verlag, Berlin (2010), `http://www.logos-verlag.de/cgi-bin/engbuchmid?isbn=2673&lng=eng&id=`, http://www.sub.uni-hamburg.de/opus/volltexte/2010/4666/
2. Cabac, L., Haustermann, M., Mosteller, D.: Software development with Petri nets and agents: Approach, frameworks and tool set. Sci. Comput. Program. **157**, 56–70 (2018). https://doi.org/10.1016/j.scico.2017.12.003
3. Combemale, B., Barais, O., Wortmann, A.: Language engineering with the GEMOC Studio. In: 2017 IEEE International Conference on Software Architecture Workshops (ICSAW). pp. 189–191 (April 2017). https://doi.org/10.1109/ICSAW.2017.61

4. Davis, A.L.: Akka Streams, pp. 57–70. Apress, Berkeley, CA (2019). https://doi.org/10.1007/978-1-4842-4176-9_6
5. Fill, H.G., Karagiannis, D.: On the conceptualisation of modelling methods using the ADOxx meta modelling platform. Enterprise Modelling and Information Systems Architectures (EMISAJ) **8**(1), 4–25 (2013). https://doi.org/10.18417/emisa.8.1.1
6. Hoffmann, K., Mossakowski, T.: Algebraic higher-order nets: Graphs and petri nets as tokens. In: Wirsing, M., Pattinson, D., Hennicker, R. (eds.) Recent Trends in Algebraic Development Techniques, 16th International Workshop, WADT 2002, Frauenchiemsee, Germany, September 24-27, 2002, Revised Selected Papers. Lecture Notes in Computer Science, vol. 2755, pp. 253–267. Springer (2003). https://doi.org/10.1007/978-3-540-40020-2_14
7. Janneck, J.W., Esser, R.: Higher-order petri net modelling: Techniques and applications. In: Proceedings of the Conference on Application and Theory of Petri Nets: Formal Methods in Software Engineering and Defence Systems - Volume 12. p. 17–25. CRPIT '02, Australian Computer Society, Inc., AUS (2002), https://dl.acm.org/doi/10.5555/846335.846338
8. Jensen, K., Kristensen, L.M.: Coloured Petri Nets - Modelling and Validation of Concurrent Systems. Springer (2009). https://doi.org/10.1007/b95112
9. Kleppe, A.: Software Language Engineering: Creating Domain-Specific Languages Using Metamodels. Pearson Education (Dec 2008)
10. Kummer, O.: Referenznetze. Logos Verlag, Berlin (2002), http://www.logos-verlag.de/cgi-bin/engbuchmid?isbn=0035&lng=eng&id=
11. 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/
12. Mahapatra, T., Prehofer, C.: Graphical flow-based spark programming. Journal of Big Data **7**(1), 4 (2020). https://doi.org/10.1186/s40537-019-0273-5
13. Mosteller, D., Cabac, L., Haustermann, M.: Integrating Petri net semantics in a model-driven approach: The Renew meta-modeling and transformation framework. Transaction on Petri Nets and Other Models of Concurrency XI **11**, 92–113 (2016). https://doi.org/10.1007/978-3-662-53401-4_5
14. Mosteller, D., Haustermann, M., Moldt, D., Schmitz, D.: Integrated simulation of domain-specific modeling languages with Petri net-based transformational semantics. T. Petri Nets and Other Models of Concurrency **14**, 101–125 (2019). https://doi.org/10.1007/978-3-662-60651-3_4
15. Padberg, J., Kahloul, L.: Overview of Reconfigurable Petri Nets, pp. 201–222. Springer International Publishing, Cham (2018). https://doi.org/10.1007/978-3-319-75396-6_11
16. Ross, D.T.: Structured analysis (sa): A language for communicating ideas. IEEE Transactions on Software Engineering **SE-3**(1), 16–34 (Jan 1977). https://doi.org/10.1109/TSE.1977.229900
17. Simon, M., Moldt, D., Schmitz, D., Haustermann, M.: Tools for Curry-Coloured Petri nets. In: Donatelli, S., Haar, S. (eds.) Application and Theory of Petri Nets and Concurrency - 40th International Conference, PETRI NETS 2019, Aachen, Germany, June 23-28, 2019, Proceedings. Lecture Notes in Computer Science, vol. 11522, pp. 101–110. Springer (2019). https://doi.org/10.1007/978-3-030-21571-2_7
18. Tiller, M.: Introduction to Physical Modeling with Modelica. Kluwer Academic Publishers, USA (2001). https://doi.org/10.1007/978-1-4615-1561-6

19. 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). https://doi.org/10.1007/978-3-540-27793-4_29
20. Varró, D., Bergmann, G., Hegedüs, Á., Horváth, Á., Ráth, I., Ujhelyi, Z.: Road to a reactive and incremental model transformation platform: three generations of the VIATRA framework. Software & Systems Modeling **15**(3), 609–629 (2016). https://doi.org/10.1007/s10270-016-0530-4
21. Xue, D., Chen, Y.: System simulation techniques with MATLAB and Simulink. John Wiley & Sons (2013)
22. Zou, L., Zhan, N., Wang, S., Fränzle, M.: Formal verification of simulink/stateflow diagrams. In: Finkbeiner, B., Pu, G., Zhang, L. (eds.) Automated Technology for Verification and Analysis - 13th International Symposium, ATVA 2015, Shanghai, China, October 12-15, 2015, Proceedings. Lecture Notes in Computer Science, vol. 9364, pp. 464–481. Springer (2015). https://doi.org/10.1007/978-3-319-24953-7_33