

Petri Nets for Object-Oriented Modeling

Stefan Witt

Abstract

Ensuring the correctness of concurrent programs is difficult, since common approaches for program design do not provide appropriate methods. This paper gives a brief introduction to Object Colored Petri Nets, which can be used for modeling the control flow of concurrent programs, for their simulation, and for deriving and proving specific properties.

Emphasis will not rely upon the precise formal definitions, but rather on the concepts and applications. The paper is intended to give computer scientists who are not familiar with Petri nets a short overview of them, especially of object-oriented nets.

1 Introduction

Object-oriented modeling requires formal techniques to handle the structure of complex systems. There are many different approaches such as the Object Modeling Technique, OMT, or the Unified Modeling Language, UML. These are extensively used nowadays, and they provide very good means for modeling the static structure of object-oriented systems, that means the classes and their relationship. They also propose methods for describing dynamic aspects like dynamically generated objects in Object Interaction Diagrams or object states in State Diagrams. But these techniques for object-oriented modeling lack formal methods for describing concurrency. Some techniques have been proposed, but they do not provide as much flexibility as Petri nets.

This paper introduces Petri nets for object-oriented modeling. Petri nets are a formal approach for modeling actions, which are carried out by concurrent processes on a computer. Elementary Petri Nets are defined in section 2. They are introduced formally in order to give an impression of the necessary mathematics that is used to prove specific properties of Petri nets. The principles of extending them to Higher Petri Nets are described in the third part. In the main section, section 4, Object Colored Petri Nets are briefly introduced, which are a formal technique for modeling concurrent object-oriented systems. Finally, a summary is given in section 5.

2 Elementary Petri Nets

A Petri net describes the static relationship between concurrent actions of processes as well as their dynamic behavior during their execution. The static structure of a Petri net consists of *transitions*, which depict the actions, *places*, which are preconditions and postconditions of the actions, and a *flow relation* that models the interdependence of the transitions and places. Formally, these can be defined as follows:

Definition. A *net* is a tuple $N = (P, T, F)$. P is the set of places, T is the set of transitions, and $F \subseteq (P \times T) \cup (T \times P)$ is the flow relation. Furthermore, $T \neq \emptyset$ and $P \cap T = \emptyset$ must be fulfilled.

Petri nets can be visualized in graphs. Places are depicted as circles, transitions are squares, and elements of the flow relation are drawn as arrows. An example is shown in figure 1, which presents the graph of a net N with $P = \{p_1, p_2, p_3, p_4\}$, $T = \{t_1, t_2\}$, and $F = \{(p_1, t_1), (t_1, p_2), (t_1, p_3), (p_2, t_2), (p_3, t_2), (t_2, p_4)\}$. The actions, which are modeled by transitions t_1 and t_2 , can be carried out concurrently, because the precondition of one transition is not a postcondition of the other one.

Since nets describe only the static structure of actions, they are extended to place/transition nets that model the dynamic behavior. This is done using tokens, which lie in places if the corresponding condition is fulfilled. A transition switches by taking tokens from the precondition places and putting new tokens in the postcondition places. A complete and formal definition will be given now, supplied from [4] and [5].

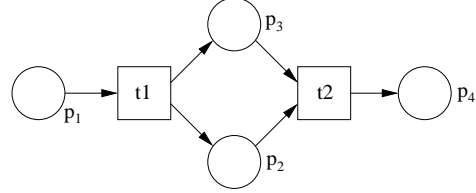


Figure 1: A Petri net

Definition. A *place/transition net* is a tuple $N = (P, T, F, K, W, m_0)$. Its ordered elements are defined as follows:

1. P, T , and F are a finite net, i.e. a net with a finite number of places and transitions.
2. K is a capacity function for each place, defined as $K : P \rightarrow \mathbb{N} \cup \{\omega\}$, where ω is an infinite capacity.
3. $W : (P \times T) \cup (T \times P) \rightarrow \mathbb{N} \setminus \{0\}$ is a weight function that attaches a weight to each arc of the flow relation. If and only if $(x, y) \notin F$, then $W(x, y) = 0$.
4. $m_0 : P \rightarrow \mathbb{N} \cup \{\omega\}$ is an initial marking, which indicates the number of tokens in each place. It respects the capacities, i.e. $\forall p \in P : m_0(p) \leq K(p)$.

The following definition gives the switching rule, also called firing rule, for place/transition net (see also [4]):

Definition. Let $N = (P, T, F, K, W, m_0)$ be a net.

1. The *precondition set* of a transition $t \in T$ is defined as $\bullet t := \{p \mid (p, t) \in F\}$. Similarly, the *postcondition set* of t is defined as $t^\bullet := \{p \mid (t, p) \in F\}$.
2. A function $M : P \rightarrow \mathbb{N} \cup \{\omega\}$ is called a *marking of N* , if $\forall p \in P : M(p) \leq K(p)$.
3. A transition $t \in T$ is *enabled in M* , if $\forall p \in \bullet t : M(p) \geq W(p, t)$ and $\forall p \in t^\bullet : M(p) + W(t, p) \leq K(p)$. This is denoted as $M \xrightarrow{t}$.
4. A transition $t \in T$ *switches or fires* to a follower marking M' of M , if $M \xrightarrow{t}$ and $\forall p \in P : M'(p) = M(p) - W(p, t) + W(t, p)$. This is denoted as $M \xrightarrow{t} M'$.

An example is given in figure 2, which shows the Petri net $N = (P, T, F, K, W, m_0)$ with

$$\begin{aligned}
 P &= \{p_1, p_2, p_3\}, \\
 T &= \{t_1\}, \\
 F &= \{(p_1, t_1), (t_1, p_2), (t_1, p_3)\}, \\
 K &: p_1 \mapsto \omega, p_2 \mapsto \omega, p_3 \mapsto \omega, \\
 W &: (p_1, t_1) \mapsto 2, (t_1, p_2) \mapsto 3, (t_1, p_3) \mapsto 1, \text{ and} \\
 m_0 &: p_1 \mapsto 3, p_2 \mapsto 1, p_3 \mapsto 2.
 \end{aligned}$$

The left side of figure 2 shows the initial marking m_0 , the right side shows the marking m_1 after switching of transition 1. According to the arc weights given by W , the transition takes 2 tokens out of p_1 , places 3 tokens into p_2 , and one into p_3 . The new marking is thus $m_1 : p_1 \mapsto 1, p_2 \mapsto 4, p_3 \mapsto 3$. Note that the number of input tokens may differ from the number of output tokens, i.e. a transition can absorb and generate tokens.

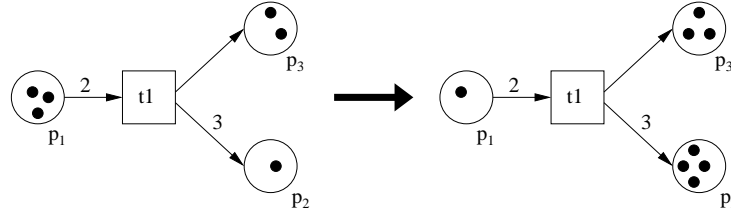


Figure 2: Switching a transition of an Elementary Petri Net. On the left the marking before the switching is shown, and on the right the marking afterwards.

3 Extensions of Elementary Petri Nets

Since the first introduction of Petri Nets by C.A. Petri in 1962, different extensions of Elementary Petri Nets have been defined, so-called *Higher Petri Nets*. The basic idea of these extensions is the usage of individual tokens instead of elementary ones. In *Colored Petri Nets* each token is of a certain type, and each arc of the net is inscribed with a type. A transition can be enabled in a marking only if the tokens in the precondition set are of the types the arcs require.

A step further is the idea of using nets as tokens instead of elementary tokens. This results in a two-level hierarchy of nets, consisting of one system net and many token nets. These nets are called *higher order Petri nets*. However, they are not further regarded in this paper.

For object-oriented modeling different Petri nets have been defined. *Elementary Object Nets*, as described in [5], use Elementary Petri Nets as tokens, and a switching rule, which regards both the state of the *System Net* and the *Object Net* (the tokens). There is only one object net, and thus the flexibility of Elementary Object Nets is restricted and not suitable for object-oriented modeling. Therefore, they are mainly used in workflow process modeling.

Newer approaches introduce Higher Petri Nets for object-oriented modeling, which are used for formal modeling of concurrent processes in practically used object-oriented languages. There are two different approaches: *Hierarchical Object-Oriented Petri Nets*, [1], and *Object Colored Petri Nets*, [2]. The latter will be described in more detail in the following section, and examples will be given. Hierarchical Object-Oriented Petri Nets are not followed in this paper, since their basic ideas are similar to Object Colored Petri Nets, for instance they also use communication transitions (called “abstract transitions”) for invoking methods of other nets.

4 Object Colored Petri Nets

Object Colored Petri Nets are a compound of static class nets, which describe the behavior of an object class, and dynamically generated object nets, which are instances of class nets. They have been defined in [2] especially for object-oriented design of concurrent program systems. Therefore, they reflect the aspects of object-oriented programming, such as dynamic generation of objects, communication between objects, and inheritance.

A class net, which is formally defined in [2], is a Petri net with the following extensions compared to Elementary Petri Nets:

1. It is a Colored Petri Net, i.e. the tokens are of different types, and the transitions are enabled in a marking if the precondition sets contain markings of the type specified by the arcs from the precondition places to the transition.

Predefined types are *UNIT* for anonymous tokens and *OID* for object identifiers. Object identifiers are used to reference to a specific object net.

2. There are different transition types: Normal transitions and communication transitions (IN, OUT, INV, and REC), which are used for method invocations (see below). A transition cannot be of more than one type.
3. Each place is inscribed with a type, and only tokens of that specific type can be put into the place.

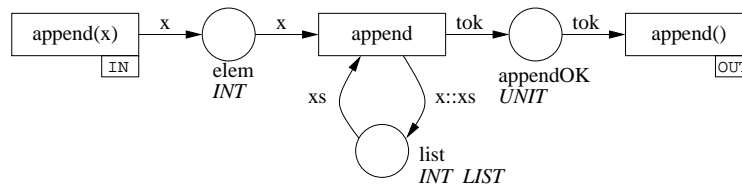


Figure 3: Function `append` modeled as a service of a class net

An example of a class net is given in figure 3, similar to an example from [2]. It shows the model of an `append` method in a list class that can store integer values. The method parameter x is an integer, which is to be appended to the list. The model

consists of an input transition $append(x)$, an output transition $append()$, and a normal transition $append$. The input transition switches every time the method is invoked with the parameter x (see below) and places one token of type INT into the place $elem$. Note that the place $elem$ can only contain tokens of INT . The $append$ transition performs the append operation by putting the integer value x and the old list xs into the $list$ place and taking the new list xs from it. It also puts an anonym token, i.e. an untyped token, into the place $appendOK$. Finally, the output transition $append()$ fires, when the append operation is done.

A call of the append method is shown in figure 4. This class net invokes the append method by firing the $id.appendCALL()$ transition, which is marked as an invocation. It requires an INT token, i.e. an integer value, from place $elem$ and an object identifier of the type OID that identifies the object instance of the called

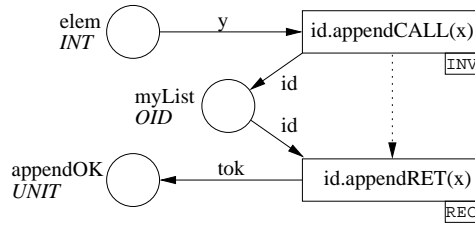


Figure 4: Class net, calling the service “append”

method. The control flow is given to the called method, and when the call returns, the $id.appendRET(x)$ transition, which is of the receive type, switches and puts an anonymous token into the $appendOK$ place. This is indicated by the dotted arrow in figure 4.

The dynamic aspect of Object Colored Petri Nets is the creation and deletion of object nets. They are created by the function new , which is a reserved function that is globally defined. It assigns a unique identifier to the calling class net and adds the new object net to the global set of nets. An example is shown in figure 5: The class net A creates a new instance of class net B whenever transition at_2 switches. An object identifier for the new object net is put into place ap_3 . The created object net of class B looks like the class net B, and is addressed by a unique identifier.

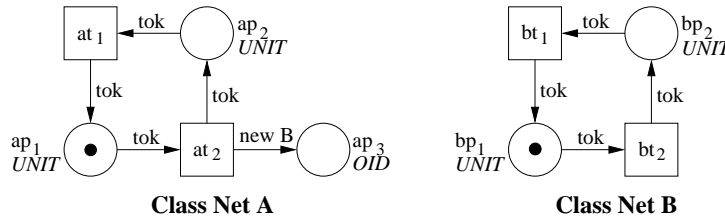


Figure 5: Creation of Object Nets. The left side shows the class net A, which generates a new instance of class net B, shown on the right side.

Apart from dynamic generation of objects and method invocation, another object-oriented concept can be modeled using Object Colored Petri Nets, the inheritance relation between classes. This is done by defining a subtype relation for the method signatures of a class. For details on this, refer to [2].

5 Summary

In this paper an approach for object-oriented modeling with Petri nets has been introduced, Object Colored Petri Nets. They provide a formal method for describing both static and dynamic aspects of a system. The static structure consists of the class definitions along with their methods, and the class hierarchy. The dynamic aspects are dynamically generated object nets, and method invocation.

Object Colored Petri Nets can be used for simulating the control flow of program systems, and for proving its correctness. They are especially good for object-oriented programs that use concurrency, because they are mathematically well understood and thus program properties can be derived from them. An example is examination if deadlocks can occur in a particular concurrent program, see [4].

Using a Petri net simulator like “Renew”, [3], the modeling can be done computer aided, and furthermore, properties of modeled nets can be derived automatically.

References

- [1] Jang-Eui Hong, Doo-Hwan Bae. *Software modeling and analysis using a hierarchical object-oriented Petri net*. In *Information Science Vol. 130 no. 1–4*, pages 133–164. Elsevier Science, 2000.
- [2] Christoph Maier, Daniel Moldt. *Object Coloured Petri Nets – A Formal Technique for Object Oriented Modelling*. In Gul A. Agha, Fiorella De Cindio, Grzegorz Rozenberg, editors, *Concurrent Object-Oriented Programming and Petri Nets*, pages 406–426. Springer-Verlag, Berlin Heidelberg New York, 2001.
- [3] Renew – The Reference Net Workshop. Available at <http://www.renew.de>.
- [4] Wolfgang Reisig. *Petri Nets – An Introduction*. Springer-Verlag, Berlin Heidelberg, 1985.
- [5] Rüdiger Valk. *Petri Nets as Token Objects – An Introduction to Elementary Object Nets*. In Jörg Desel, Manuel Silva, editors, *Application and Theory of Petri Nets 1998*, pages 1–25. Springer-Verlag, Berlin Heidelberg New York, 1998.