Object-oriented Approach to Timed Colored Petri Net Simulation

Michał Kowalski and Wojciech Rząsa
Rzeszow University of Technology
Department of Computer and Control Engineering
al. Powstancow Warszawy 12, 35-959 Rzeszow, Poland
Email: michal.kowalski.87@gmail.com, wrzasa@prz-rzeszow.pl

Abstract—This paper presents object-oriented design of library meant for modeling and simulating Timed Colored Petri Net models. The approach is prepared to integrate TCPN models with crucial parts of larger applications implemented in object-oriented languages. The formal models can be tightly joined with applications allowing the latter to interpret states of the formal models. The formal models can be simulated, to observe behavior of modeled systems. Both approaches are used, to derive conclusions about analyzed systems [1], [11], [15], [14]. This approach, how-ever, is rarely used in business applications (e.g. in banking, trade, accounting). The first reason, certainly, being complexity and scale of this kind of software. Secondly, for the sake of special competences required for modeling and analysis and thus cost of this process.

Timed Colored Petri Net (TCPN) is a flavor of Petri Net formalism designed by K. Jensen [6]. It is suitable for modeling and analysis of distributed and concurrent systems. Since in Colored Petri Nets (CPN) tokens can carry values (colors) of specified types (colorsets) models created using CPN are more compact and thus more clear. Consequently, CPN can be conveniently used to analyze large systems. Time extension of Timed Colored Petri Net enables analysis of time relationships e.g. efficiency.

The first reason, certainly, being complexity and scale of this kind of software. Secondly, for the sake of special competences required for modeling and analysis and thus cost of this process.

Timed Colored Petri Net (TCPN) is a flavor of Petri Net formalism designed by K. Jensen [6]. It is suitable for modeling and analysis of distributed and concurrent systems. Since in Colored Petri Nets (CPN) tokens can carry values (colors) of specified types (colorsets) models created using CPN are more compact and thus more clear. Consequently, CPN can be conveniently used to analyze large systems. Time extension of Timed Colored Petri Net enables analysis of time relationships e.g. efficiency.

TCPN is based on a formal model. The formal model is a mathematical construct that captures the behavior of the system. The formal model can be used to verify the correctness of the system. TCPN is a powerful formalism for modeling and analyzing complex systems. It is used in a variety of applications, including computer systems, communication networks, and manufacturing systems.

To make this approach more pervasive it is however necessary to prepare libraries designed for programming languages commonly used in business applications and enabling convenient incorporation of formalisms. In this work we present an approach to enable TCPN-based modeling using Java – one of most frequently used object oriented programming languages. We present object-oriented design of library we have implemented to support modeling and simulation of Timed Colored Petri Nets as a part of larger systems. This first approach assumes that TCPN model should be correctly simulated. Formal analysis of TCPN models is considered as possible future work.

II. RELATED WORK

There is great variety of Petri net flavors and there is also great variety of computer tools designed to create and analyze Petri net models. An extensive list is maintained e.g. by University of Hamburg1. For Jensen’s Timed Colored Petri Nets there are two important software packages: Design/CPN [3] and newer, rewritten in Java: CPN Tools [7], [10]. Both of them have GUI to create Petri net models and both implement algorithms enabling simulation and formal analysis. Also both tools provide interfaces for external communication, that allow various levels of integration using various programming languages.

Design/CPN has interface called COMMS/CPN [5]. It allows to communicate with simulator while Petri net is being simulated. It is a message passing interface with small set of functions that allow to establish connection, send and receive messages. The functions should be called from Standard ML code being part of Petri net model (e.g. guards). There is Java/CPN interface, that facilitates establishing communication with COMMS/CPN from Java software.

The newer CPN/Tools has two kinds of interfaces forming Access/CPN framework [17]. The first one, available for Standard ML and more tightly connected with the model is designed with state space analysis in mind. It is worth mentioning, that this interface is meant to be formalism independent, not strictly connected to TCPN. The second part of Access/CPN is Java CPN Model Interface. It allows to

1http://www.informatik.uni-hamburg.de/TGI/PetriNets/tools/db.html
create or import model created using CPN Tools. It also supports TCP/IP based communication with running simulator.

The presented TCPN design and analysis tools certainly provide interesting interfaces that can be used in selected applications. The message passing communication allows to steer TCPN simulator and respond to some events. It is however a low level communication solution. It also does not allow for tight coupling of TCPN models with applications: TCPN model should be implemented using Standard ML programming language, and Petri net side of communication should also be programmed in Standard ML. It is not possible to implement guards and inscriptions directly in object-oriented language and pass objects as token values.

Petri Net Kernel [16] is a Java library designed to support implementation of various types of Petri nets. It can also be adapted for Timed Colored Petri Nets and in fact was exploited in our research done so far [12], [13], [14]. This general purpose tool has an important disadvantage: token values in Colored Petri Nets are represented by String objects. This approach has certain justification: in Java String objects are immutable, as tokens in Petri nets. Consequently, design of the programming language ensures one of important assumptions of the formalism. This has, however, two important disadvantages. First, in applications where tokens should carry values more complex than strings, it is not convenient to develop String representation of each possible value to pass it to the model and then parse it to an object when its being collected. Second, the performance loss resulting from necessity of parsing the strings is significant.

III. CHALLENGES OF OBJECT-ORIENTED DESIGN FOR TCPN

Library designed in this work should enable modeling and correct simulation of TCPN models. The simulator should be able to run automatically without requiring user interaction and efficiency of simulation must be sufficient for practical applications.

Tight integration of a model as a part of a larger application should be possible, in order to allow subsequent states of a model to be automatically interpreted by an application in its domain of responsibility. In order to limit the effort required by integration, model of Petri nets should be created using classes and objects of Java. Additionally, the library should put possibly small restrictions on objects that are traversing TCPN models in the form of tokens and on implementation of TCPN guards and inscriptions.

Petri net formalism, however, puts restrictions on behavior of its models. The one that strongly affects the design of the library and presents significant challenge, concerns behavior of tokens. Tokens in TCPN are immutable similarly to symbols in functional languages. Presented solution must correspond to this demand, to ensure correct behavior of TCPN model. This requirement must be reconciled with the need to put almost arbitrary object as token value, as mentioned above.

IV. OBJECT-ORIENTED REPRESENTATION OF TCPN

This section presents object-oriented design of the library together with solutions to major problems.

A. Elements of the TCPN Graph

The bipartite graph of Timed Colored Petri Nets consisting of two types of vertexes: Places and Transitions and of Arcs joining the vertexes is described by the natural entities used in object-oriented design.

Places are represented by objects of class Place holding String attribute describing name. Class attribute called type and describing type of the place and marking attribute described by a class implementing IMarking interface.

Transitions are described by two attributes: name of class String and by guard used to determine if the transition can be fired. Guard is implemented as a reference to an object implementing IGuard interface which defines only one method called guard. The method gets binding as an argument and returns a Boolean value to indicate the guard result.

In this work we distinguished input arcs (from a place to a transition) and output arcs (from a transition to a place) and described them separately. For simplicity, we assumed that input arcs will be used only to define number of tokens used in firing a transition, more complex operations can be performed on output arcs using not only basic expressions but also functions. This assumption does not limit expressiveness power of TCPN and considerably facilitates simulation process.

Input arcs are objects of class InputArc and hold references to the places being their sources. The InputArc class objects have also inscription attribute describing tokens that should flow through this arc while firing transitions. The inscription is a String in the format analogous to the one defined by K. Jensen [6] and describing count of tokens and variables.

Output arcs are represented by objects of class OutputArc and containing references to their destination places and inscriptions. The inscriptions are represented by objects implementing the IExpression interface that defines method called process. The method receives binding of the transition being fired as an argument, and returns collection of tokens (class Token) that should be put in the output place as a result of transition firing. The process method can perform any desired operations on tokens and their contents. BasicOutputArcExpression class provides means to define simple expressions analogous to the ones used for input arcs.

The references between subsequent elements of TCPN join transitions with their input and output arcs, while the arcs store references to their source or destination places. This solution allows to conveniently access required information while firing a transition by simulator. Parser for arc inscriptions provided as strings is an implementation of finite automata.
B. Tokens

Tokens indicating state of a Petri net model are represented by generic class Token. Type of the class parameter corresponds to the type of the token used in Petri nets. It also determines type for the value field that holds the token value. This way any class can be used as token type, according to the requirements. Token timestamp is represented by timestamp field of the Token class.

Consistent simulation of a Petri net requires token values to be controlled by the net only. In Java this is not easy to achieve, since objects are passed to and from methods by reference, not by value. Consequently, it might happen that one object is referred by more than one token as its value or a user could modify value of an object during simulation interfering with the simulation algorithm.

The TCPN designed by K. Jensen are described using functional programing language. This solves the previously mentioned problems by natural means of this paradigm: the tokens are immutable, as are symbols in functional programing. To emulate this behavior in Java, we decided that the following requirements are crucial: (1) TCPN simulator can be trusted to deal with the tokens respecting Petri net principles. (2) By design we must ensure that the user will not be able to breach the rules of TCPN simulation, despite his or her lack of knowledge concerning these principles. (3) For efficiency reasons objects should be copied as rarely as possible.

In order to meet the above requirements, we clone token values (using deep cloning described in [4]) when they enter simulator structures and when they are about to leave these structures. Thus, the compromise ensures correct simulation, regardless of how the token values are processed outside of the TCPN structures, concurrently preserving reasonable efficiency overhead.

C. Marking

Simulation of TCPN consists on subtracting and adding tokens from and to the markings of Petri net places, therefore proper implementation of marking is crucial for efficiency of the simulator. The structure used for implementation of marking must enable efficient search and verification of existence of tokens of known values. In case of Jensen's timed nets the structure should also enable efficient consideration of tokens with timestamps indicating that they can be used later are placed in waitingTokens field. These fields refer to objects of different classes, implementing different data structures. The activeTokens data structure is an implementation of hash table with support for storing multiple values for a single key. The hash function in this structure uses token values as keys. Thus tokens with given values can be found efficiently while firing a transition, this being a key of efficient TCPN simulation. When simulation clock is advanced selected tokens from waitingTokens in each place must be moved to the activeTokens structure. To perform this operation efficiently, the waitingTokens structure is a priority queue sorted by tokens timestamps (TokenComparator class is responsible for proper comparison of the tokens in the queue).

The BasicMarking class implements IMarking interface and the BasicMarking can be easily substituted by different implementation of marking implementing the same interface, which ensures all required methods are provided by the implementation. Each marking must implement getDistinctTokens returning list of tokens from the marking used to generate bindings while simulation. Boolean method containsToken allows to verify if the given token exists in the marking. Method getToken allows to verify if the given token exists in the marking. Method putToken and putTokens enable adding tokens to the marking. During simulation getNextTime method is used to determine the least value of the clock that would release new tokens from waitingTokens to activeTokens in this marking. Finally, setTime method is called while each change of simulation clock.

D. Petri Net Structure

The structure of TCPN model is stored in an object of class CPNet. The user is supposed to create TCPN model by creating Place, Transition, InputArc and OutputArc objects together with guards and inscriptions implementing IGuard and IExpression interfaces. For efficiency of simulation it is however vital, to provide various information about the net as quickly as possible, without the need to repeatedly exploit computationally intensive graph algorithms. Therefore, after user provided the model, the CPNet class performs additional processing in order to cache data concerning model structure and efficiently provide required information for the simulator.

On the basis of the list of transitions provided by the user the CPNet class creates list of all places in the model (as Java ArrayList). Additionally it caches Petri net structure in an array analogous to graph incidence matrix. The internal structures are filled by method init of class CPNet that should be called after the CPNet object is provided with the list of objects representing TCPN transition.

The proposed solution allows to reconcile requirements concerning efficient access to different aspects of TCPN model while simulation, with the need to ensure consistency between different representations. Obviously the structure of the Petri net cannot be changed after the call to the init method.
V. Simulation Data Structures

Variables are represented in different ways, depending on the context. In the arc structures they are represented as objects of class Var containing String attribute name and Integer attribute denoting quantity of tokens to be removed from a place. In a binding the variables are represented as their names of class String. Similarly to [6] variable's range is limited to arcs connected with one transition and their type must be consistent with type of places their arcs are related to.

Bindings are entities assigning variables to their values during simulation. Subsequent bindings are stored in HashMap objects with the key being the variable name and value of class Token. Thus, during simulation access to variable values connected with a binding is both convenient and efficient. The HashMap is encapsulated in the Binding class that exposes two methods: get to obtain tokens assigned to a variable and insert to set binding between a variable and a token.

VI. Integration Interface

The goal of this work was to enable integration of Petri nets with software implemented using imperative programming language. Designed approach enables this integration in two possible ways.

Firstly, crucial elements that create Petri net model and that are part of its behavior, can be objects used in the other parts of application. Tokens can carry objects of arbitrary classes as their values. Transition guards and arc inscriptions can be implemented to depend on application logic.

Secondly, the presented solution is equipped with event listener interface. The listeners can be registered to be called before and after events concerning changes in markings, transition firing, and changes of simulation clock.

VII. Summary

In this paper we presented object-oriented approach to modeling and simulation using Timed Colored Petri Nets. Petri net model with all its components, including token values, guards and inscriptions, should be implemented in object-oriented programming language. The concept was implemented as library in the pervasively used Java.

The approach allows one to integrate Petri net model in a larger application and let it benefit form the formalism-based simulation. The integration can be done while the model is implemented, by joining its components with components of the larger system. It can also be tightened by the use listener interface that allow to implement code responding to specific events occurring in Petri net model. Consequently, the enclosing software can influence behavior of the formal model. Concurrently, the Petri net can steer behavior of the software that can interpret events from the model and apply them in the software’s domain of responsibility.

The design of the approach ensures not only tight and convenient integration with object-oriented software. It also ensures that the rules of TCPN simulation are preserved. Thus, the software can benefit from the formal rules, governing behavior of modeled processes and from their correctness.

The presented solution includes TCPN simulator, that allows execution of created model. Care was taken, to ensure efficiency of TCPN simulation that allows practical applications. Necessary optimizations were designed and implemented and if required, additional solutions can be developed.

References