DOMAIN SPECIFIC LANGUAGE FOR PETRI NETS

Maxim CERNETCHI[1], Corneliu NASTAS[1],
Alexandr CARA[1], Vladislav CRUCERESCU[1]

[1]Department of Software and Automation Engineering, group FAF-212, Faculty of Computers, Informatics and Microelectronics, Technical University of Moldova, Chisinau, Moldova

*Corresponding author: Cernetchi Maxim, maxim.cernetchi@isa.utm.md

Scientific coordinator: Irina Cojuhari, conf. univ., dr., DISAa

Abstract. This scientific paper presents the development of a Domain Specific Language (DSL) for Petri nets, a formalism used for modeling concurrent systems. The DSL provides an intuitive syntax for modeling Petri nets, which enables domain experts to design and analyze complex systems without the need for formal training in the underlying theory. The paper outlines the syntax and semantics of the DSL and demonstrates its applicability through a set of case studies. The results show that the DSL improves the productivity and efficiency of the modeling process and makes the analysis of Petri nets more accessible to a wider audience. The paper concludes with a discussion of the benefits and limitations of the DSL and its potential for future research and development in the field of Petri nets.

Keywords: *Petri Nets, transitions, places, token, dsl.*

Introduction

Petri Nets are a powerful mathematical tool used to model complex systems and processes [1]. They provide a graphical representation of a system's state and its transitions, making them an ideal choice for modeling a wide range of scenarios. However, creating and analyzing Petri Nets can be a time-consuming and error-prone process, especially for large and complex systems. To overcome this challenge, Domain-Specific Languages (DSLs) can be used to provide a more efficient and convenient way to work with Petri Nets. A DSL is a programming language that is tailored to a specific domain, making it easier to express domain-specific concepts and operations. In this context, creating a DSL for Petri Nets can simplify the process of modeling, simulating, and analyzing Petri Nets, improving both the accuracy and the speed of the analysis. This can have significant benefits in areas such as manufacturing, logistics, and systems engineering, where the use of Petri Nets is common. In this article, we will explore the process of creating a DSL for Petri Nets, discussing the benefits of this approach and providing practical examples to illustrate its usage.

Domain analysis

Petri Nets are a mathematical modeling language used to represent and analyze the behavior of discrete systems. Petri Nets are graphical representations of the interactions between different components in a system and provide a visual representation of the underlying relationships between states and transitions.

Key concepts and principles:

Petri Nets are a graphical modeling language used to represent and analyze the behavior of discrete systems. The key concepts and principles of Petri Nets include:

1. *Places*: places represent the components of a system, such as resources, queues, or intermediate stages of a process. They are represented by circles in a Petri net diagram.

2. *Transitions*: transitions represent events or actions that change the state of the system. They are represented by rectangles in a Petri net diagram.

3. *Tokens*: tokens are used to represent the presence of a resource or the completion of an event. Tokens are represented by dots in a Petri net diagram and are placed in places.

4. *Arcs*: arcs represent the relationships between places and transitions. They are represented by directed lines connecting places to transitions. Inbound arcs (place to transition) indicate the input required for a transition to fire, while outbound arcs (transition to place) indicate the output produced when a transition fires.

5. *Firing*: firing is the process by which a transition changes the state of the system by consuming tokens from input places and producing tokens in output places. A transition can only fire if there are enough tokens in its input places and enough free space in its output places.

6. *State*: the state of a Petri net is represented by the distribution of tokens in the places. A Petri net can be in multiple states at the same time, representing different scenarios or parallel processes.

7. *Reachability*: reachability is the property of a Petri net that allows for the analysis of the possible sequences of transitions that can lead to a particular state. This property is useful for verifying the behavior of the system and for analyzing the impact of changes to the system.

8. *Boundedness*: boundedness is the property of a Petri net that ensures that the number of tokens in a place that remains within a specified range. This property is important for ensuring the stability and reliability of the system.

These are the key concepts and principles of Petri Nets. Understanding these concepts is essential for effectively using Petri Nets to model and analyze discrete systems.

Tools and techniques:

There are several tools and techniques that are commonly used in the analysis and design of systems modeled with Petri Nets. Some of the most important tools and techniques include:

1. *Modeling and simulation*: Petri Nets can be used to create models of systems, which can then be simulated to visualize and analyze the behavior of the system.

2. *State space analysis*: Petri Nets can be analyzed to determine the number of reachable states, the number of transitions between states, and other properties of the system. This analysis is useful for verifying the behavior of the system and for analyzing the impact of changes to the system.

3. *Verification and validation*: Petri Nets can be used to verify and validate system models by checking the consistency and completeness of the model and by comparing the behavior of the model with the behavior of the real system.

4. *Performance analysis*: Petri Nets can be used to analyze the performance of systems by calculating various performance metrics, such as response time, throughput, and utilization.

5. *Control and optimization*: Petri Nets can be used to control and optimize systems by determining the optimal policies for controlling the transitions and by finding the optimal settings for the parameters of the system.

These are some of the most important tools and techniques for Petri Nets. The choice of tool or technique will depend on the specific requirements of the system being modeled and the goals of the analysis.

Applications and use cases:

Petri Nets have a wide range of applications and use cases in various domains, including but not limited to:

1. *Manufacturing and production*: Petri Nets can be used to model and analyze the behavior of production systems, such as assembly lines, and to optimize the production process.

2. *Computer networks*: Petri Nets can be used to model and analyze the behavior of computer networks, such as communication protocols and data flow, and to optimize the performance of the network.

3. *Software engineering*: Petri Nets can be used to model and analyze the behavior of software systems, such as workflow systems and user interfaces, and to optimize the design and implementation of the software.

4. *Business process modeling*: Petri Nets can be used to model and analyze the behavior of business processes, such as supply chain management and customer service, and to optimize the performance of the processes.

5. *Healthcare*: Petri Nets can be used to model and analyze the behavior of healthcare systems, such as patient flow and resource utilization, and to optimize the delivery of care.

6. *Transportation*: Petri Nets can be used to model and analyze the behavior of transportation systems, such as air traffic control and traffic flow, and to optimize the performance of the transportation system.

These are just a few examples of the wide range of applications and use cases for Petri Nets. The versatility of Petri Nets makes them suitable for modeling and analyzing a wide range of discrete systems in various domains.

Grammar and Language Design.

Grammar refers to the set of rules governing the structure of a language, including the way words are formed, combined, and used to convey meaning. It encompasses everything from the parts of speech and their functions to sentence structure and punctuation.

The language design started from identifying what the DSL language can and must do. We identified the basic rules for the petri Nets:

1. Petri Nets have transitions and places

The syntax for this would be simple instantiation:

```
place p;
place p1, p2, p3;
tran t;
tran t1, t2, t3;
```

Where pos, tran is the datatype of the variable. Notice that it uses semicolons to represent the end of the declaration.

2. The places' state describes the amount of token it has, it also has a token maximum capacity which can be finite or infinite.

This can be done with this simple representation:

```
place p;
p.amm = 3;
p.cap = 9;
```

The token amount of the place p is 3, whilst its maximum capacity is 9. The default quantity for the token amount is zero, whilst for the maximum capacity is infinity.

3. Transitions and places are connected through inbound and outbound arcs:

Transitions and places can connect with multiple places and transitions.

```
place p, p1, p2, p3;
tran t, t1, t2;
t.in = {p1, p2};
t.out = {p3};
p.in = {t1, t2};
p.out = {t};
```

The inbound arcs for the transition t, connect it to p1 and p2, whilst the outbound arc connects it to p3. p is connected to t1 and t2 with outbound arcs, which should not be confused with the syntax p.in. For simplicity, it means whether the arrow in the graphical representation is pointing towards the place or the transition. Notice that "p.out = {t}" connects itself to t which already has arcs. Rather than overriding t's arcs, it would add to the inbound arcs t already has, making t.in = {p, p1, p2}.

4. Arcs have weights:

Inbound arcs' weight attribute describes the amount of tokens it will consume during firing. Outbound arcs' attribute describes the amount of tokens it will produce during firing.

```
place p1, p2, p3, p4;
tran t1, t2;
t1.in = {p1 : 2, p2};
t2.out = {p3, p4};
p3.in = {t1 : 3};
p3.out = {t2};
```

The inbound arc "p1->t1" has the weight of 2. For the variables where there is no ": <number>" notation, the weight is set to the default value 1.

With this we designed the following grammar for the DSL:

```
V_N = {<program>, <declarationlist>, <declaration>, <instantiation>, <placefield>,
<connection>, <type>, <varlist>, <var>, <number>, <arcing>, <arclist>, <arc>,
<nonzero>, <digits>, <digit>, <alpha>, <string>, <char>}
V_T =
{a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s,t,u,v,w,x,y,z,A,B,C,D,E,F,G,H,I,J,K,L,M,N,O,P,
Q,R,S,T,U,V,W,X,Y,Z,_,0,1,2,3,4,5,6,7,8,9,.,,,:,{,},=,;}
P = {
<program> -> <declarationlist>,
<declarationlist> -> <declaration> ; <declarationlist> | ε,
<declaration> -> <instantiation> | <placefield> | <arcing>,

<instantiation> -> <type> <varlist>,
<type> -> place | tran,
<placefield> -> <var>.amm = <number> | <var>.cap = <number>,
<arcing> -> <var>.in = { <arclist> | <var>.out = { <arclist>,
<arclist> -> <arc> , <arclist> | <arc> },
<arc> -> <var> : <number> | <var>,

<number> -> <nonzero><digits>,
<digits> -> <digit><digits> | ε,
<digit> -> 0|<nonzero>,
<nonzero> -> 1|2|3|4|5|6|7|8|9,

<varlist> -> <var> , <varlist> | <var>,
<var> -> <alpha><string>,
<string> -> <char><string> | ε,
<char> -> _|<digit>|<alpha>,
<alpha> -> a|b|c|d|e|f|g|h|i|j|k|l|m|n|o|p|q|r|s|t|u|v|w|x|y|z|A|B|C|D|E|F|G|H|I|
J|K|L|M|N|O|P|Q|R|S|T|U|V|W|X|Y|Z
}
```

Figure 1. Grammar Definition of DSL syntax

Example:

```
place p1, p2, p3;
tran r1, r2;
p1.amm = 4;
p2.amm = 3;
p3.cap = 5;
r.in = {p1 : 2, O2};
p3.in = {r1 : 2};
p3.out = {r2};
```

Would generate something like this:

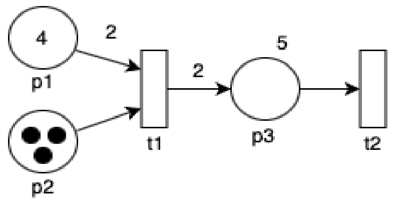

Figure 2. Petri Net representation

Conclusion.

Petri Nets are a valuable tool for modeling complex systems and processes, providing a graphical representation of a system's state and its transitions. However, working with Petri Nets can be a time-consuming and error-prone process, especially for large and complex systems. Creating a Domain-Specific Language for Petri Nets can simplify the process of modeling, simulating, and analyzing Petri Nets, improving both the accuracy and the speed of the analysis. By tailoring a programming language to the specific needs of Petri Nets, a DSL can make it easier to express domain-specific concepts and operations, allowing users to focus on the problem at hand instead of the technical details of the modeling process. The benefits of using a DSL for Petri Nets are significant, and can have a positive impact on fields such as manufacturing, logistics, and systems engineering. As such, studying Petri Nets and exploring the creation of DSLs for them is an important area of research, with the potential to improve the efficiency and effectiveness of many real-world applications.

References:

1. Păstrăvanu O., Matcovschi M., Mahulea C. *Aplicații ale rețelelor Petri în studierea sistemelor cu evenimente discrete*. Iaşi: Editura Gh. ASACHI , 2002.