Universiteit Leiden
Opleiding Informatica

Refitting PIPE: Extending the Petri Net tool PIPE 5

Name:	Dico Duba
Studentnr:	1455060
Date:	21-08-2016
1st supervisor:	Jetty Kleijn
2nd supervisor:	Bas van Stein

BACHELOR THESIS

Leiden Institute of Advanced Computer Science (LIACS)
Leiden University
Niels Bohrweg 1
2333 CA Leiden
The Netherlands

Refitting PIPE

Extending the Petri Net tool PIPE 5

Dico Duba

Abstract

Elementary nets (EN) are the most fundamental class of Petri nets. For this reason they are useful to teach about the basics of Petri nets before moving on to higher level Petri nets. The Platform Independent Petri net Editor (PIPE) is an open-source tool to draw and analyse Petri nets. It has been originally developed by students at Imperial College London in 2004. PIPE 5 is the latest version of this program, created during a master project in 2014.

In this thesis we report on a bachelor project to adapt PIPE 5 and make it suited for students to create and analyse EN systems. For this, three modules were created: An EN converter and validator, a state space generator and a process generator. Besides this, numerous bugs have been fixed and several improvements related to usability have been made to turn using PIPE for education purposes into a smooth experience.

Contents

List of Figures

Chapter 1

Introduction

1.1 Petri nets

Petri nets are mathematical models to describe and analyse distributed systems [13]. Petri nets can be visualized as directed graphs with places, transitions and arcs. This visual representation allows for a more intuitive representation of a particular system.

Several types of Petri nets exist. In this thesis we are mostly interested in Place/Transition (P/T) systems and Elementary Net (EN) systems. Higher level Petri nets exist, but these type of systems are usually more complex and more focussed on practical problems. For education it is preferred to start with simple Petri nets, with EN systems being the most fundamental model.

In Petri nets, active elements are represented by transitions and passive elements by places. Places contain local information. This information is displayed as so-called "tokens" in these places. The arrangement of tokens in the net is called a configuration, modelling the global state over the places. The net can be animated by activating transitions (called "firing"). Then tokens disappear and appear, creating a new configuration. Firing transitions only changes the configuration, not the structure of the net.

1.2 PIPE

PIPE (Platform Independent Petri net Editor) was developed originally by students at Imperial College in London. The earliest release was in 2004, and the latest official release was by this group in 2007 with version 2.5 [4]. The source code was then moved to Sourceforge [3]. Between 2007 and 2010 PIPE 3 was developed, and between 2010 and 2013 PIPE 4 was developed. Development of PIPE 3 and 4 happened by volunteers all

over the world. While the functionality of PIPE was improved greatly between 2007 and 2013, the quality of the code deteriorated. The code of PIPE 4 was highly tangled [15]. During this period modules were added and editing more types of Petri nets became possible. In 2013, PIPE 5 was moved to Github[1] and rebuilt using the model-view-controller (MVC) and the publisher-subscriber software design pattern [15]. At the beginning of 2016 only the basic program is implemented. This version still has numerous bugs, unfinished implementations and some questionable design choices that would make usage of this program harder.

1.3 Goals

The mathematical concepts behind Petri nets are important, but when teaching students about Petri nets it is helpful to have them visualize the Petri nets when talking about certain behaviour. In 2010, PIPE was chosen to be best fit to teach about simple Petri nets in the Theory of Concurrency course at LIACS [6], because it was open source and modular. At other universities the program is used to design and analyse General Stochastic Petri Nets [12] [8].

It has been briefly considered to use a different tool during the Theory of Concurrency course. Petri-LLD was the only tool we found that supports EN systems natively, but development has stopped on this tool in 2006 [1]. We did not find it worth the effort to update this version to work correctly on multiple platforms and extend this version to support the analysis we want to do on EN systems. None of the other tools listed in the extensive database of Petri Net World support EN systems natively [2]. Since PIPE is still maintained, and Zwijger found this tool the best fit to teach about simple Petri nets in 2010 [6], we will also work with PIPE. PIPE 2.5, and later PIPE 4 have a number of problems that make them difficult or unreliable to use. Besides this, both versions have some problems where the tool does not behave as the mathematical model dictates. We have chosen to extend version 5 over version 4, because version 4 is unmaintained since 2013, and the code base of PIPE 5 is much cleaner than of version 4.

The goal of the project described in this thesis is to create a version of PIPE 5 that can be used at LIACS to teach about Petri nets. First and foremost, PIPE 5 needs to support EN systems. To analyze EN systems, a module to create process nets and a module to create reachability graphs for EN systems have to be made. Finally the program needs to be bug free and intuitive to use for end-users. This thesis will provide the theoretical background and both an overview of newly added features and fixed bugs. The final product will be made available to LIACS, as well as to the general public via Github[2]. Changes made to PIPE by us will be proposed as so-called pull requests on the main Github projects by Tattersall.

[1] https://github.com/search?q=user%3Asarahtattersall+PIPE
[2] https://github.com/search?q=user%3Adcduba+PIPE+fork%3Atrue

1.4 Report structure

First we will establish the mathematical definitions of the models discussed in this thesis in chapter 2. In chapter 3 we will look at what was in PIPE 5 before this project began. Then, in chapter 4, we look at what modules were added to PIPE, and which bugs were fixed in this project. In chapter 5 we look at how well we accomplished the goals as well as suggest future work.

Chapter 2

Definitions

2.1 Preliminaries

$\mathbb{N}$ denotes the set of natural numbers: $\{0, 1, 2, 3, \ldots\}$ and $\mathbb{N}_+$ the set of all positive natural numbers: $\{1, 2, 3, \ldots\}$. Similarly, $\mathbb{R}_+ = \{x | x \in \mathbb{R}, x > 0\}$ denotes the set of all positive real numbers.

In the set $\mathbb{N} \cup \{\omega\}$, ω is a special symbol representing an "infinite number". By convention, $\forall n \in \mathbb{N} : n < \omega$.

We use $\mathcal{P}(x)$ to denote the power set of x.

2.2 Elementary Net systems

Definition 1. [7] A net is a triple $N = (P, T, F)$, where:

(1) P and T are finite sets with $P \cap T = \emptyset$ and

(2) $F \subseteq (P \times T) \cup (T \times P)$. □

P is the set of places of N, T the set of transitions of N and F comprises the arcs between these elements.

A net can be represented graphically, with places, transitions and the arcs between them represented by circles, rectangles and arrows. See Figure 2.1 for an example.

For a net $N = (P, T, F)$, respectively we define the input-set of $x \in P \cup T$ as ${}^\bullet x = \{y \in P \cup T \mid (y, x) \in F\}$. Similarly, we define the output-set of $x \in P \cup T$ as $x^\bullet = \{y \in P \cup T \mid (x, y) \in F\}$. From the definition it follows that the input-set and the output-set of a place contain only transitions, and the input and the output-set of a transition contain only places. In other words, N is a directed bipartite graph.

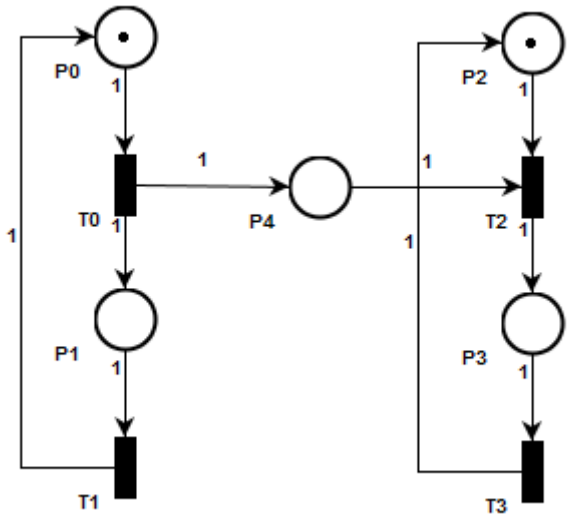


Figure 2.1: "Producer-consumer" EN system. Places are represented with circles. Transitions are represented by black rectangles. P_1 and P_2 contain a token represented by a black dot.

Definition 2. A configuration (or marking) C of a net N is a subset of P. □

A configuration denotes the current state of the net. When a place $p \in P$ belongs to C, we say that P is marked (in C) or that there is a "token" in P (in C).

A token is represented graphically by a dot in the corresponding place. See Figure 2.1 for an example of a marked place in a Petri net.

The following definition is based on [7], but adds restrictions that will be explained later.

Definition 3. An elementary net system (EN system) is a quadruple $M = (P, T, F, C_{in})$, where:

(1) (P, T, F) is a net and

(2) $C_{in} \in P$ is the initial configuration.

(3) $\forall t \in T \ : \ {}^\bullet t \cap t^\bullet = \emptyset$

(4) $\forall t \in T \ : \ {}^\bullet t \neq \emptyset \neq t^\bullet$ □

The initial configuration C_{in} of $M = (P, T, F, C_{in})$ is the set of places that initially contain a token, where $P \setminus C_{in}$ is the set of places without a token.

Definition 4. A transition $t \in T$ is enabled (to occur) in a net $N = (P, T, F)$ for configuration C if:

(1) $\forall p \in {}^\bullet t : p \in C$ and

(2) $\forall p \in t^\bullet : p \notin C$. □

To denote that t is enabled in C, we write $t \operatorname{con} C$.

Definition 5. When an enabled transition t occurs (is "fired") in an EN system in configuration C, the new configuration is $C' = (C \setminus {}^\bullet t) \cup t^\bullet$. □

We write $C\,[t\rangle\,C'$ to denote that t was enabled and fired in configuration C, and this resulted in the new configuration C'.

An EN system can be animated by firing enabled transitions. This is sometimes referred to as the "token game". From the initial configuration transitions are fired, and the tokens in the graphical representation of the system are moved to represent the new configuration.

To denote firing multiple transitions in a sequence we write $C\,[t_1, \ldots, t_n\rangle\,C'$ if $\exists C_0, \ldots, C_n \subseteq P$ such that $C_{i-1}\,[t_i\rangle\,C_i$ for $1 \leq i \leq n$. If an arbitrary sequence is fired from C_0 resulting in C_n, we can instead write $C_0\,[seq\rangle\,C_n$ where seq denotes this arbitrary sequence.

2.3 Place/Transition systems

Definition 6. [7] A multi-net is a tuple $N = (P, T, F, W, K)$, where:

(1) (P, T, F) is a net,
(2) $W : F \to \mathbb{N}_+$ is the weight function, and
(3) $K : P \to \mathbb{N}_+ \cup \{\omega\}$ is the capacity function. □

The weight function for $(p, t) \in F$, $p \in P$, $t \in T$ denotes the number of tokens that are removed from p if t fires. The weight function for $(t, p) \in F$, $p \in P$, $t \in T$ denotes the number of tokens that are added to p if t fires.

The capacity function denotes the maximum number of tokens that are allowed in that place, where ω means that the capacity is unbounded.

The graphical representation is like a normal net (definition 1), with the following differences:

- If $W(f) > 1$ (the weight for arc f is more than 1) for $f \in F$, then $W(f)$ is displayed as a number on arc f.
- If $K(p) \neq \omega$ (the capacity for place p is not infinite) for $p \in P$, then $K(p)$ is displayed as a number next to p.

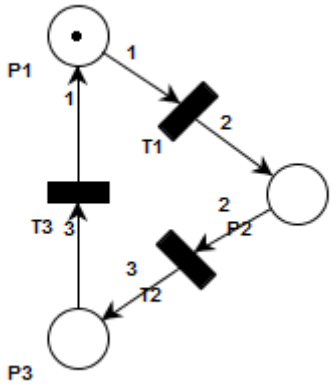


Figure 2.2: Simple P/T system

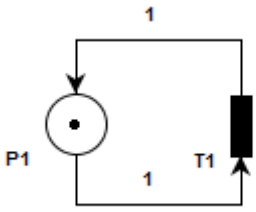


Figure 2.3: Net with a self-loop: ${P_1}^\bullet \cap {}^\bullet P_1 \neq \emptyset$

Definition 7. A configuration (or marking) of a multi-net N is a function $C : P \to \mathbb{N}$, such that $\forall p \in P : C(p) \leq K(p)$. □

A configuration denotes the current state of the net. The marking of a place is represented by the number of tokens currently in the place.

Definition 8. [7] A place/transition system (P/T system) is a tuple $M = (P, T, F, W, K, C_{in})$, where:

(1) (P, T, F, W, K) is a multi-net and
(2) $C_{in} : P \to \mathbb{N}$ is the initial configuration. □

The initial configuration gives the number of tokens initially present in each place of M. See figure 2.2 for an example of a P/T system.

By definition, places in an EN system can contain at most one token. Places in P/T systems have a capacity, and can any number of tokens up until that capacity. These places only behave like places in EN systems if the capacity is exactly 1, and at most 1 token is in the place. Similar to this, in an EN system a transition takes all tokens from inbound places, and puts tokens in every outbound place. For the net to behave like an EN system, and since we can never add more tokens to a place than its capacity, the weight on every arc to and from a transition must therefore be 1.

The transition enable rule for P/T systems as defined in definition 9 does not automatically work the same as the enable rule in EN systems as defined in definition 4. If the net contains so-called self-loops (Figure 2.3), where there is a place $p \in P$ for which $p^\bullet \cap {}^\bullet p \neq \emptyset$, the transition t in that self-loop behaves differently in a P/T system than it does in an EN system. If no token is in p, both the enable rule for P/T systems (definition 9 and EN systems (definition 4) fail on the first condition. If p would contain a token, the transition would be enabled in a P/T system. With the restrictions in the previous paragraph, the first condition holds: $W(p,t) \leq C(p) = 1 \leq 1$. The second condition, $C(p) + W(t,p) - W(p,t) \leq K(p) = 1+1-1 \leq 1 = 1 \leq 1$ also holds. In EN systems, the first condition for the enable rule holds ($p \in C = \mathit{true}$), but the second condition does not hold ($p \notin C = \mathit{false}$).

Thus, in EN systems, such a transition should not be enabled. Changing the constants we defined in the previous paragraph will not matter, because by definition $W(p,t) = W(t,p)$, so the condition $C(p) + W(t,p) - W(p,t) \leq K(p)$ will always be $C(p) \leq K(p)$, which is by definition true. In conclusion, to simulate an EN system in a P/T system, the P/T system may not contain any self-loops.

Definition 9. A transition $t \in T$ is enabled in a net $N = (P, T, F, W, K)$ for configuration C if:

(1) $\forall p \in {}^\bullet t \ : \ W(p,t) \leq C(p)$ and
(2) $\forall p \in t^\bullet \ : \ C(p) + W(t,p) - W(p,t) \leq K(p)$ □

Definition 10. When an enabled transition t occurs (is "fired") in a P/T system $N = (P, T, F, W, C_{in})$ in configuration C, the new configuration C' is $\forall p \in P \ : \ C'(p) = C(p) - W(p,t) + W(t,p)$. □

2.4 Other nets in PIPE

Only one type of Petri net exists in PIPE. The features of this Petri net in PIPE are covered by two high-level Petri nets: Coloured Petri nets and Generalised Stochastic Petri nets. In this section we will give the definition of these two types of Petri net and show how we can emulate P/T systems using these types of nets.

2.4.1 Coloured Petri net

Definition 11. [9] A Coloured Petri net (CPN) is a tuple $M = (P, T, A, \Sigma, Col, G, N, E, C_{in})$, where:

(1) $P \cap T = P \cap A = T \cap A = \emptyset$,
(2) Σ is a set of colour sets,
(3) $Col : P \to \Sigma$ is the colour function,
(4) $G : T \to \mathit{bexpr}$ is a boolean guard function,

(5) $N : A \to (P \times T) \cup (T \times P)$ is the arc mapping function,

(6) $E : A \to expr$ is the arc weight expression function and

(7) C_{in} is the initial configuration mapping places $p \in P$ to a multiset of tokens with colours corresponding to $Col(p)$.

□

In PIPE multiple arcs between the same elements are joined together into one arc.

In CPN's, multiple arcs can exist between the same two components, where in nets (definition 1) and multi-nets (definition 6) only one arc can exist between every combination of place and transition.

The numeric expression *expr* in the function E dictates the "weight" for each token colour on a particular arc. This expression can contain special variables that correspond to the number of tokens in a particular place.

The boolean expression *expr* in the function G dictates if a transition can fire. Again, this expression can contain special variables that correspond to the number of tokens in a particular place.

A CPN system behaves like a P/T system where the capacity of each place is unbounded under the following circumstances.

- Only one colour exists in a P/T system. Σ must thus be a set with only one colour in it. Similarly *Col* now maps every place into that one colour.
- Only one arc can exist between two components in a P/T system. When only one such arc exists, (P, T, A, N) can be reduced to (P, T, F) as defined in definition 1.
- In P/T systems, a weight is in $\mathbb{N}_+$. This is a valid mathematical expression, so if $E : A \to \mathbb{N}_+$ the weight function E for CPN systems behaves like the weight function W for P/T systems.
- P/T systems have no guard function, so if we take *true* as the special value of the boolean expression for N, the guard function will never prevent a transition from being enabled that would otherwise be enabled.

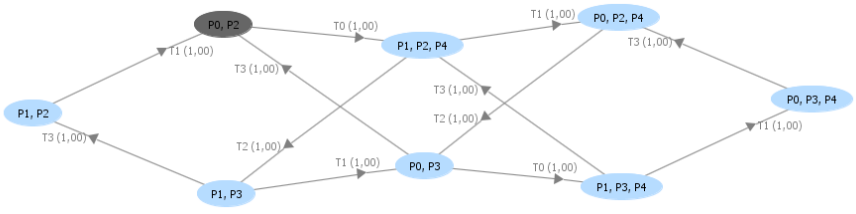


Figure 2.4: Reachability graph for "producer-consumer" EN system in figure 2.1.

2.4.2 Generalized Stochastic Petri net

Definition 12. [5] A Generalised Stochastic Petri net (GSPN) is a tuple $G = (P, T, \Pi, I, O, H, \Delta, C_{in})$, where:

(1) $P \cap T = \emptyset$,

(2) $\Pi : T \to \mathbb{N}_+$ is the priority function,

(3) $I, O, H : T \to P^*$ are the input, output and inhibition functions mapping transitions to multisets of places,

(4) $\Delta : T \to \mathbb{R}_+$ is the transition rate and weight function and

(5) $C_{in} : P \to \mathbb{N}$ is the initial configuration. □

T in this definition consists of immediate transitions (T_i) and timed transitions (T_t). When an (enabled) immediate transition fires, no time passes. When an (enabled) timed transition fires, some time passes based on its associated transition rate. A transition can never be both an immediate and a timed transition ($T_i \cap T_t = \emptyset$).

The priority function dictates which transition should fire first. By convention, all immediate transitions fire before any timed transition fires. Immediate transitions have a weight that can be used to calculate the likelyhood of an immediate transition firing when it is in conflict with an other immediate transition, where an equal weight means that both are equally likely to fire.

A GSPN system behaves like a P/T system where the capacity of each place is unbounded under the following circumstances.

- A P/T system does not have inhibitor arcs. When $H = \emptyset$, we can reduce (P, T, I, O, H) to (P, T, F) as defined in definition 1.
- In P/T systems only immediate transitions exist, and each of these transitions is equally likely to fire when they are enabled. This means that for a GSPN system to behave like a P/T system, each transition must have the same priority and the same weight. We can set both to the constant 1.

2.5 Analysis of Petri nets

Several ways of analysing Petri nets exist. In this section we define what a reachability graph and a coverability graph is for P/T systems. The reachability graph and coverability graph for GSPN systems are similar to those of P/T systems, but due to the timed transitions in GSPN systems they are usually constructed differently. Their definitions are not given here, but instead the differences are highlighted in the relevant parts of chapter 4. We also define what process nets are, but we only do this for contact-free EN systems.

2.5.1 State space

Definition 13. [7] An edge-labelled directed graph is a tuple $G = (V, \Gamma, \Sigma, v_{in})$, where:

(1) V is a finite set of nodes,

(2) Σ is a finite set of edge labels,

(3) $\Gamma \subseteq V \times \mathcal{P}(\Sigma) \times V$ is a set of labelled edges and

(4) v_{in} is the starting node. □

Each triple $(v_{start}, label, v_{end}) \in \Gamma$ is an edge from v_{start} to v_{end} with label *label*.

Definition 14. A sequential configuration graph (SCG) or reachability graph for a P/T system $N = (P, T, F, W, K, C_{in})$ with initial configuration C_{in} is a edge-labelled graph $G = (V, \Gamma, \Sigma, v_{in})$ with all reachable configurations from a Petri net, where:

(1) $V = \{C' \mid seq \in T^*, C_{in}\ [seq\rangle\ C'\}$,

(2) $\Sigma \subseteq T$,

(3) $\Gamma = \{(v_{start}, t, v_{end}) \mid v_{start}, v_{end} \in V, t \in T, t \operatorname{con} v_{start}, v_{start}\ [t\rangle\ v_{end}\}$ and

(4) $v_{in} = C_{in}$. □

In other words, the graph contains an edge labelled by t between two reachable configurations if a transition t is enabled in the first configuration, and firing t in that configuration yields the second configuration. In the definition above v_{start} and v_{end} nodes are configurations. Thus nodes and configurations are in a one to one correspondence.

The SCG for an EN system is always finite. A SCG for a P/T system may be infinite. See figure 2.4 for an example of a SCG for an EN system.

Definition 15. A marking C covers configuration C' if $\forall p \in P\ :\ C(p) \geq C'(p)$. □

When C covers C', we can say $C \geq C'$.

Definition 16. [10] A coverability set S of a P/T system $N = (P, T, F, W, K, C_{in})$, where the capacity of every place is unbounded, is a subset of $(\mathbb{N} \cup \{\omega\})^P$ such that the two following conditions hold:

(1) for every reachable configuration $C \in C_N$, there is a marking $C' \in S$ such that $C' \geq C$,

(2) for every configuration $C' \in S \setminus C_N$, there is an infinite strictly increasing sequence of reachable configurations converging to C'. □

A coverability set S is minimal if and only if no proper subset of S is a coverability set of N. The minimal coverability set is used to describe the state space of a P/T system with a finite number of states. Every configuration covers itself by definition. $S \setminus C_N$ is a set of states where the marking in one or more places is set to the arbitrary large symbol ω. An infinite strictly increasing sequence of reachable configurations means that for all configurations that are not in S ($D = C_N \setminus S$), $\forall D_0, D_1 \in D \ : \ D_0 \, [seq\rangle \, D_1, D_1 \geq D_0$. These configurations will eventually converge to the configurations where the number of tokens in one or more places are infinite, represented by ω.

The (minimal) coverability graph has some nodes with reachable configurations from a Petri net and special nodes for configurations that cover previous configurations. For configurations that are not strictly greater than an other configuration the arcs are the same as in sequential configuration graphs. For nodes that have an infinite strictly increasing sequence of reachable configuration converging to it, there is one arc for each configuration that eventually leads to that configuration, labelled with each transition that would lead to a precursor to that configuration.

Definition 17. [14] A minimal coverability graph of a P/T system $N = (P, T, F, W, C_{in})$, where the capacity of every place is unbounded, is a finite edge-labelled graph $G = (V, \Gamma, \Sigma, v_{in})$, where:

(1) V is the minimal coverability set,

(2) $\Sigma \subseteq T$,

(3) $\Gamma = \{(v_{start}, t, v_{end}) \mid v_{start}, v_{end} \in V, t \in T, t \, \mathrm{con} \, v_{start}, v_{start} \, [t\rangle \, v_{end}\}$ and

(4) $v_{in} = C_{in}$ □

In this definition, each node mentioned in definition 16 part 2 represents all configurations that coverge to that node. This means that if we have a configuration C_{start} represented by node v_{start}, a configuration C_{middle} and a configuration C_{end} represented by node v_{end}, we can say $v_{start} \, [t\rangle \, v_{end}$ if and only if $C_{start} \, [t\rangle \, C_{middle}$ and an infinite increasing sequence of reachable configurations converging exists between C_{middle} and C_{end}. Both C_{middle} and C_{end} are represented by v_{end}.

2.5.2 Process nets

Definition 18. [11] An EN system has contact in configuration C if $\exists t \in T$, where:

(1) ${}^\bullet t \subseteq C$ and

(2) $t^\bullet \cap C \neq \emptyset$. □

"Contact" is a situation where a transition t would be enabled based on ${}^\bullet t$, but is not enabled because one or more of its output places is filled. See figure 2.5 for an example.

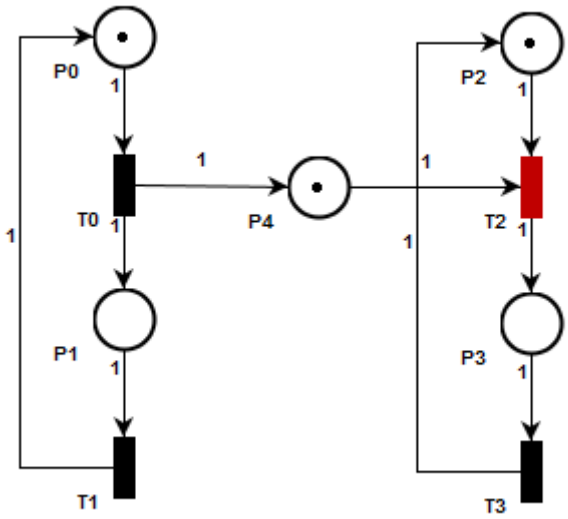


Figure 2.5: T_0 is not enabled, because it has contact in P_4.

Definition 19. [11] A concurrent run for an EN system $N = (P_N, T_N, F_N, C_{in_N})$ is a tuple $Co = (P_{Co}, T_{Co}, F_{Co}, L)$, where:

(1) $\forall p \in P_{Co} \ : \ |{}^\bullet p| \leq 1 \wedge |p^\bullet| \leq 1$,

(2) Co is an acyclic net,

(3) $L \ : \ Co \rightarrow N$ is a labelling function,

(4) $L(P_{Co}) \subseteq P_N$ and $L(T_{Co}) \subseteq T_N$,

(5) $\forall p \in P_{Co}$, if ${}^\bullet p = \emptyset$, then $L(p) \in C_{in_N}$ and

(6) $\forall t \in T_{Co} \ : \ L({}^\bullet t) = {}^\bullet(L(t)) \wedge L(t^\bullet) = (L(t))^\bullet$. □

Concurrent runs are used to describe the concurrent flow and the causal dependencies of an EN system. This definition does not enforce that elements cannot occur concurrently to themselves. However, contact-free EN systems are guaranteed to never have this problem [11].

Definition 20. A process net $Proc = (P_{Proc}, T_{Proc}, F_{Proc}, L)$ of a contact-free EN system $N = (P, T, F, C_{in})$ is a concurrent run of that system. □

All states in this process net can be mapped to states in N with L. If a transition t in $Proc$ can be fired from a configuration C_{Proc} resulting in C'_{Proc}, there exists a corresponding configuration C_N for N where $L(t)$ can be fired resulting in C'_N, where $L(C_{Proc}) = C_N$ and $L(C'_{Proc}) = C'_N$.

Chapter 3

Overview of PIPE

PIPE is an open-source program written in java to build and analyse Petri nets. It provides the means to build several types of high-level Petri nets, such as GSPN and CPN systems (definition 11 and 12), as well as regular P/T systems (definition 8). When PIPE is started, it is automatically in "Drawing mode", shown in Figure 3.1. The user can add elements by clicking the corresponding button in the toolbar and next clicking in the drawing area in the bottom-left. Places, transitions and annotations are added by clicking the button and clicking in an empty space. Arcs are added by clicking the source element, then optionally clicking in an empty area to create a bend, and finally clicking on the target element. PIPE only allows creation of proper arcs; that is both an arc between two elements of the same type and also an arc between two elements that already have a similar arc connecting them is not allowed.

Right-clicking on Petri net elements allows the user to open a dialog box to edit the properties of that element, and gives an option to delete that element. Deletion will always leave the net in a consistent state, meaning that if a place or transition is deleted, all arcs that are connected to it are deleted as well.

Places in PIPE have a unique identifier (name), a (possibly unbounded) capacity and a number of tokens for each token colour (e.g. 2 default tokens, 3 blue tokens). The number of tokens in each place defines what the starting configuration is.

Transitions in PIPE have a unique identifier (name), and can be marked as timed transitions or immediate transitions. Timed transitions reserve input tokens and output them after a certain delay based on a rate parameter. Immediate transitions behave as a timed transition, but with no delay. Transitions can be marked as single servers or infinite servers. A single server can only reserve one batch of tokens before it must output the tokens, even if it is still enabled after reserving the tokens. An infinite server can reserve as many batches of tokens als it can, as long as the transition is enabled. Timed transitions are usually only used in normalized Petri nets where each place has an unbounded capacity, where reserved batches of tokens can never overflow

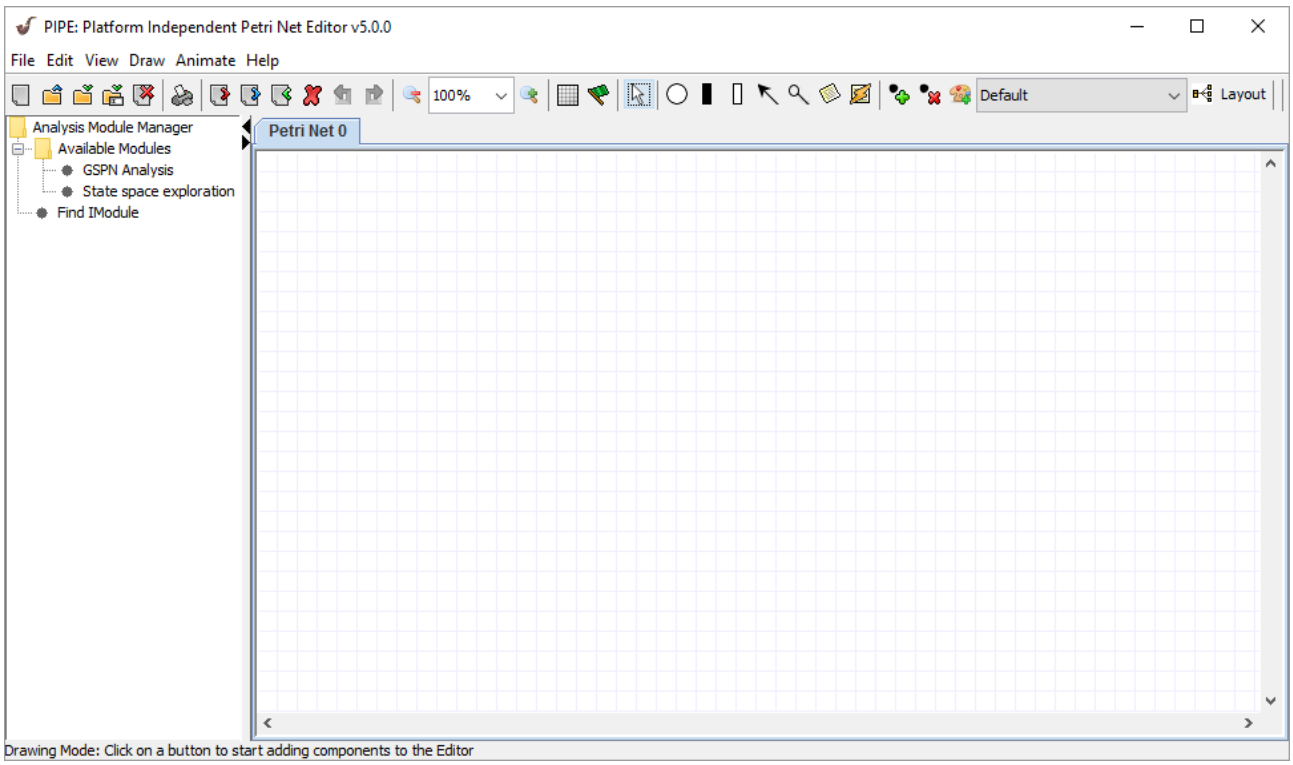


Figure 3.1: PIPE 5 in drawing mode. Top: menu bar and tool bar. Bottom left: Module manager. Bottom right: Tabbed view with drawing area for "Petri Net 0".

the output places. Immediate transitions can be given a priority. Enabled transitions with a higher priority must be fired before other enabled transitions can be fired.

There are two types of arcs in PIPE, regular arcs and inhibitor arcs. Regular arcs are directed arcs between a place and a transition, or between a transition and a place. Regular arcs can have a weight for each token type. This weight determines how many of each token is removed from, or added to, a place if a transition is fired. An inhibitor arc is a directed arc between a place and a transition, and disables the transition if one or more tokens are in the connected place. Per type, only one arc can exist between a place and a transition and between a transition and a place.

3.1 Animation

PIPE allows the user to enter animation mode to fire transitions and change states, based on the initial configuration defined in the drawing phase. This mode can be started by clicking the flag button in the toolbar, or by selecting the option in the menu bar. The interface now shows an additional panel "Animation History" in the bottom left showing previous fired transitions. Enabled transitions are now displayed in red. (Figure 3.2)

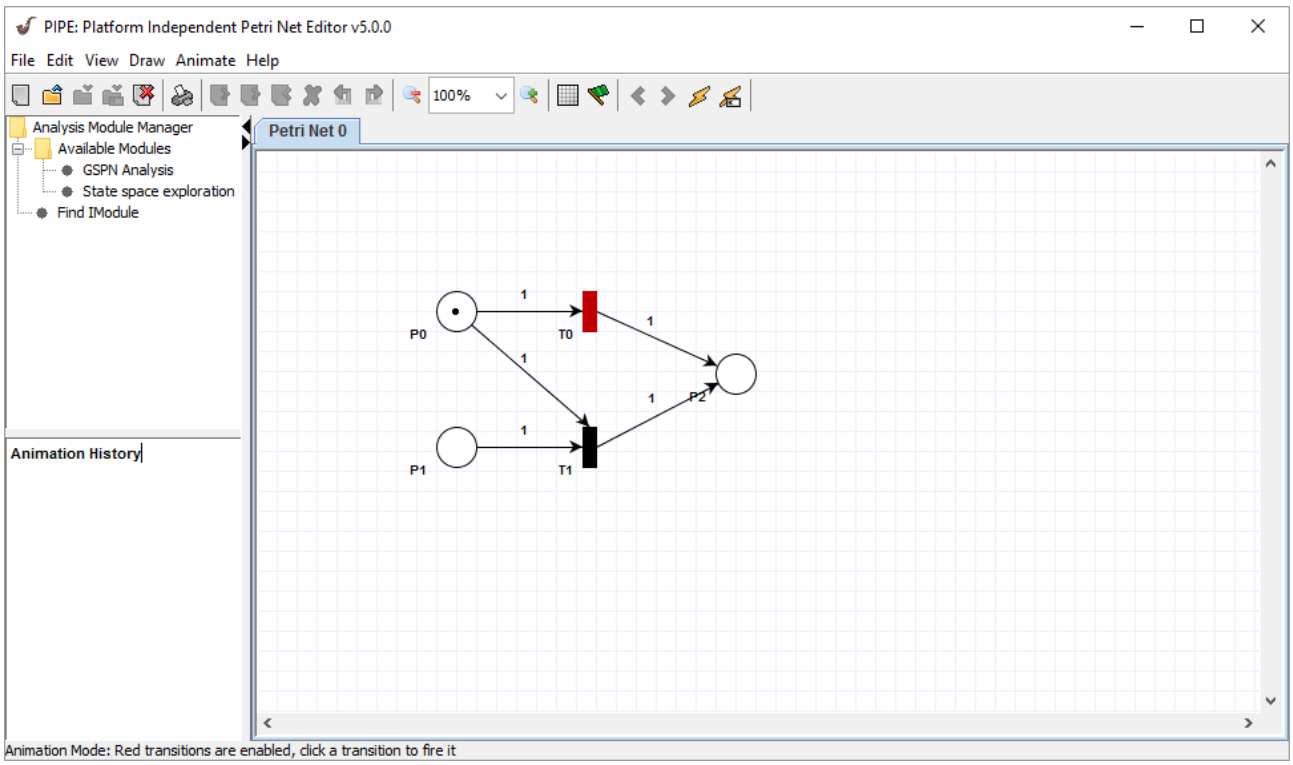


Figure 3.2: PIPE 5 in animation mode. Bottom left: Record of previously fired transitions (Animation History). Bottom right: Animated net "Petri Net 0" with enabled transition T_0.

There are three ways a user can fire transitions from a given configuration. The first method is to click a transition they want to fire. Enabled transitions are marked in red to make it easier for the user to identify such a transition. The user can also click a button in the toolbar to select a random enabled transition to fire. Lastly the user can click a button in the toolbar to fire a number of transitions with a specified delay.

The animation history in the bottom left shows which transitions have been fired. Buttons in the toolbar allow the user to go back a step or go forward a step in the fire sequence. Animation does not change the underlying net, or the initial marking. Once the user exits animation mode the net with its original, initial marking will be displayed again.

3.2 Modules

PIPE has a system in place which allows for easy integration of custom modules in PIPE, even without recompiling the program. These modules work on the underlying Petri net and can both alter the net or analyse it. At the start of this project two modules exist: A state space explorer and a GSPN analysis module. The state space explorer generates reachability graphs and coverability graphs. The GSPN analysis module calculates some characteristics of GSPN's such as the steady state distribution, the average token counts in each place and the transition throughputs. The modules can be started by double clicking the name of the

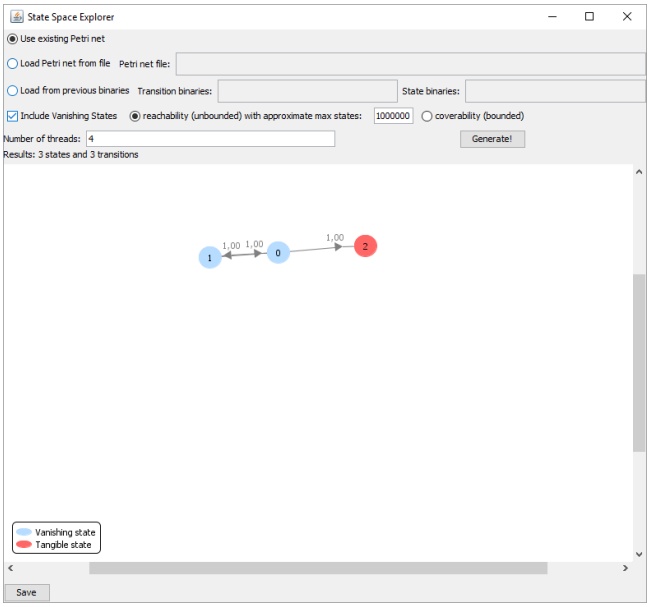


Figure 3.3: Old State Space Explorer module.

module in the Module Manager on the left side of the program as shown in Figure 3.1. Modules can be run as stand-alone programs by loading the jar file and then instead of starting PIPE, starting the module.

3.2.1 State Space Explorer

The state space explorer (Figure 3.3) generates reachability graphs and coverability graphs. The module can generate a graph for the currently displayed Petri net, generate a graph for a Petri net loaded from a file or load a previously generated graph from a binary file. In addition, the user can choose if they want to display vanishing states in the graph. In context of GSPN systems with timed and immediate transitions, vanishing states are those states in which an immediate transition can be fired. By convention immediate transitions are fired before timed transitions, so by resolving all immediate transitions invisibly, a cleaner reachability and coverability graph is created. If a Petri net contains a loop with immediate transitions, generating a coverability graph without vanishing states will result in a so-called "timeless trap". This means that the explorer can not find the next steady state (where all immediate transitions are resolved), and thus can not generate the graph. For P/T and EN systems where only immediate transitions exist, vanishing states are states that have outbound arcs. To create correct reachability and coverability graphs for P/T and EN systems, vanishing states should always be included in the graph.

The user can also select if they want to generate a reachability graph or a coverability graph. Since reachability graphs are potentially unbounded, an upper bound for the number of states has to be given. The program will not generate states anymore when this number is reached. Moreover the number of threads in which the state space is calculated can be given. This can speed up generation of graphs for larger Petri nets [15].

When the button "Generate!" is clicked, the reachability or coverability graph is generated on the canvas below the button. States are numbered based on the order they have been derived, and arcs show the throughput of tokens between two states. When hovering the mouse over a state, the marking in that state should be shown.

In the bottom left is a button to save the graph as a binary file. This binary file contains the states and arcs as displayed on the canvas, and can be used to load the reachability or coverability graph again without needing to generate it again from the Petri net.

3.2.2 GSPN Analysis

The GSPN Analysis module analyses various properties of the current Generic Stochastic Petri net. For each of the tangible states it calculates the marking of that state. It also displays the steady state distribution, the average token count for each place in these tangible states and the average timed throughput through timed transitions. Since this thesis is not about these type of nets and this module has not been changed we will not go into detail about these properties.

3.3 Code base of PIPE 5

PIPE 5 is developed as several "independent" Github projects with several software design patterns kept in mind. Design patterns are used in software engineering to better structure and create easier to maintain code. We identified the following design patterns in the existing code base of PIPE.

Model-view-controller pattern

The model-view-controller design pattern splits up code into three parts: Code that strictly handles what the user sees (the view), code that strictly handles how data is stored (the model), and code that handles interaction between the user and the program (the controller).

Publisher-subscription pattern

The publisher-subscription pattern decouples code by letting parts of code subscribe to events that signify changes in other pieces of code. For example, when the user creates a place and transition, then connects these two elements with an arc, the arc will "listen" to changes of the position of the place and transition. When the place is moved over the canvas, it will generate events that its x or y coordinate has changed. The arc that listened to these events can use this to update the coordinates of its end points so that the arc is still visually connected to the place.

Visitor pattern

The visitor pattern allows to perform an action on an instance of a class, without extending that particular class. This is used in various places, such as when adding or removing an element from the Petri net. This way a component can be removed as an abstract component rather than a specific component.

3.4 Github projects

The code is split up in four Github projects containing five parts, namely PIPEGui, PIPEModuleGui, PIPECore, PIPEMarkovChain and PIPEAnalysis. This is a summary of what the code of each part contains.

PIPEGui

PIPEGui is the main project of PIPE. It is the graphical interface of the program. This project contains the views and controllers of PIPE. The application builder creates the top view "PipeApplicationView" and creates for each of the buttons, menu items and comboboxes in the toolbar and menus an "Action". These actions denote what should happen when the user clicks a button or menu item.

The "PipeApplicationController", the controller for the "PipeApplicationView" is in charge of the tabs containing Petri nets and the "PetriNetController" instances that correspond to those tabs. Besides that, it contains functionality to save Petri nets. Drawable components each have their own controllers to provide an easy way to change their properties.

As described earlier, right-clicking components of a Petri net allows users to open a dialog where properties of that component can be edited. The gui description of these dialogs can be found in this Github project. Besides this, this Github project contains mouse and keyboard input handlers for anything that happens on the canvas inside a tab. The handlers will behave differently based on whether the application is currently in drawing mode or in animation mode. To undo and redo actions an entire group of classes dedicated to "history actions" is included in this project. These classes dictate how various actions have to be undone, or redone. The module manager is also included in this project and for each module that is compiled with this project, a stub is included that calls the actual class located in PIPEModuleGui.

PIPEModuleGui

PIPEModuleGui contains the classes that are called by the module manager in PIPEGui. The classes create a dialog, and execute the logic behind that dialog. Each of these classes could theoretically be executed as stand-alone program, as they all include a "main" method (default called method on startup).

PIPECore and PIPEMarkovChain

PIPECore contains the model part of the model-view-controller design pattern of the PIPE application. The model for most of the application is, not surprisingly, a Petri net. For each element of a Petri net there is at least one class in the model. The logical model behind how to animate a net is also included here.

To save a Petri net, it needs to be serialized. The input-output classes are included in PIPECore. GSPN systems are built directly on top of Markov chains. To decouple the code further, that part of the code is split out to its own project PIPEMarkovChain. PIPEMarkovChain also included a reader and writer to serialize states as found in the State Space Explorer.

In PIPECore we can also find the data models for naming a Petri net, visitors for Petri net components in the visitor pattern, possible exceptions that can be thrown and parsers for functional weights and the grammars it can contain.

PIPEAnalysis

PIPEAnalysis contains the implementation of a multithreaded MapReduce-style state explorer and a parallel Gauss-Seidel steady state solver. This project is only referenced in the two modules that were included by Tattersall in her rewrite of PIPE [15].

Chapter 4

Contributions

The goal of this project is to create a version of PIPE that works better than the existing version of PIPE 4. While the code of this version of PIPE is a lot cleaner than that of version 4, the program is very bare-bones. Many features from PIPE 4, including zoom functionality and export as an image are missing. The code is incomplete in certain areas, and some unit tests seem to be written to pass for the current implementation rather than to describe logical behaviour. Some examples of problems include:

- Places that can not be modified.
- A wide range of problems related to improper handling of renaming Petri net elements such as being unable to delete elements when they are renamed and being able to create multiple arcs between two elements.
- An animation mode that still does take in account already deleted components.
- Undo and redo buttons that potentially leave the Petri net model in an inconsistent state.
- Modules that work for the GSPN systems they were intended for, but break when used on other types of Petri nets implemented in PIPE, or when the Petri net is modified.

In this section we will give a brief description of the modules that will be created to be able to use PIPE 5 during the Theory of Concurrency course at LIACS. Besides creating these modules, most issues mentioned earlier will be fixed. The code for the added modules is available in Appendix B.

Support of EN systems

Although EN systems in itself are specific P/T systems (definition 8), it is a laborious process to create such a system in PIPE. Besides this, the program is to be used by students who are not yet very familiar with Petri nets, or the specific differences between EN systems and the nets simulated in PIPE. For this

reason, a module is to be created that views a given Petri net as a net that behaves in every way like an EN system.

Configuration graph and coverability graph

There already exists a module for a configuration graph and a coverability graph, but these graph are not useful for the purposes we want to use them. The graphs display too little information and nodes are often displayed on top of each other. Moreover, the markings in the tooltips when hovering over a state are unreadable, and sometimes even missing. To be more useful, the nodes should instead show the markings and the tooltip should include the markings in a more human-readable format. An effort has to be made to prevent overlapping nodes as much as possible. Edges should include transition names that resulted in that edge.

The coverability graph uses an arbitrary large number instead of ω to replace infinitely growing states. The graph was found to generate graphs that were incorrect, because the generator made assumptions about capacity of places where it should not. This has to be fixed too.

Support of process generation

There currently does not exist a way to create process nets (definition 20) from a source net. An "unfolding" button exists, but this functionality has been disabled since the implementation was flawed in PIPE 4. Even then, unfolding takes into account every possible fire sequence while we are only interested in creating a net for a single fire sequence. The goal here is to create a module that takes a fire sequence as input, then checks if that fire sequence is even possible, and then creates a process net from this fire sequence.

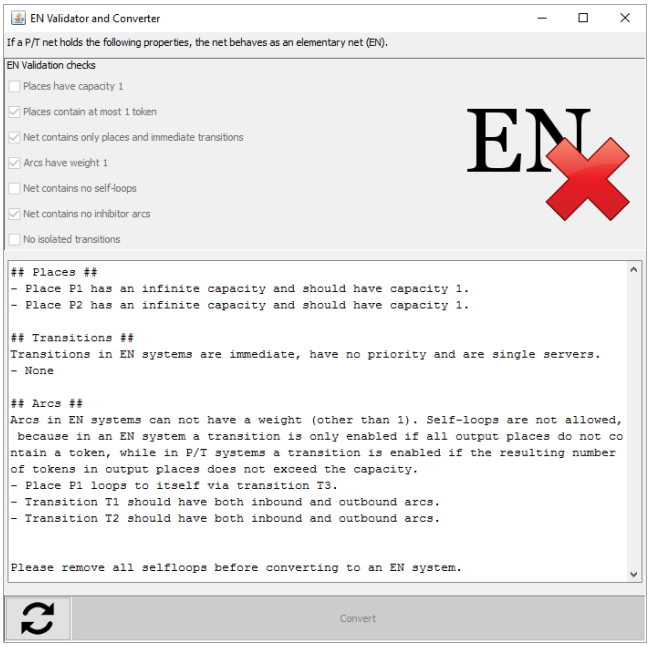


Figure 4.1: Dialog of EN Validator and Converter for net in figure 2.3.

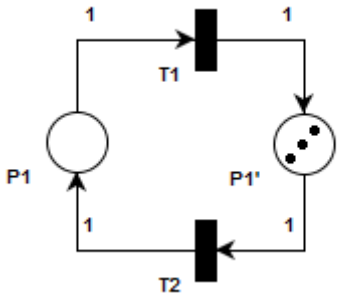


Figure 4.2: Place p with complement place p', simulating a place with capacity 3.

4.1 P/T systems in PIPE

The nets simulated in PIPE are a combination of GSPN nets (definition 12) and CPN nets (definition 11). We have already shown how P/T systems where the capacity of places is unbounded can be simulated in these types of nets. Places with finite capacity can be simulated in these nets by complementing a place (figure 4.2). Complementing a place p is done by adding a place p', and setting ${}^\bullet p' = p^\bullet$ and $p'^\bullet = {}^\bullet p$. The combined number of tokens in p and p' is the capacity of p. However, PIPE already provides us with a feature that allows us to set the capacity of a place, so no complement places are required. This allows us to fully simulate P/T systems, as long as we take specific values for the functionality we don't need.

The rule that tests if a transition is enabled in PIPE is complicated, because it has to work for any net that is drawn. When all components and features that are not needed in P/T systems are trivialized, the enable rule is indeed equal to the rule defined in definition 9.

4.2 EN validator and converter

In PIPE it is not easy to change the underlying Petri net model on the fly. One could change this model, but this means that one instance of PIPE has to be compiled for EN systems and one instance has to be compiled for GSPN systems. This approach is impractical. Instead EN systems are defined as a GSPN/CPN system as already implemented, but with certain restrictions. In section 4.1 we showed how to view P/T systems as a combined GSPN/CPN system. In definition 8 we showed that we can only simulate EN systems in P/T systems if the Petri net does not contain any self-loops, and all weights and capacities are set to 1.

4.2.1 Dialog

The new EN validator and converter module opens a dialog as shown in figure 4.1. In the top is a field that shows which checks were passed and which have not. To the right of that field is an image that allows the

user to verify if a net is a valid EN system with just a glance. Below that a detailed breakdown of the issues is given. At the very bottom is a refresh icon that runs the module again on the current net. The majority of the space is taken up by the convert button that changes certain things in a net so that it will behave as an EN system.

4.2.2 Validation and conversion

As a general guideline, the module only converts a net if it visually looks like the EN system we want to emulate in PIPE. All checks that did not pass are reported to the user. First the module will check if all places have a capacity of 1. When the net is converted, the capacity of all places is set to 1, regardless of what their capacity used to be. Then the module will check if all places contain at most 1 token, because by definition $\forall p \in P \ : \ C(p) \leq K(p)$. In definition 11 we mentioned that to simulate a P/T system in a CPN net, we had to define a colour set with one token type. We choose this token type to be a black token named "default", as this is the token type every new net begins with. If a place has tokens, they are replaced by a single "default" token. After this, no coloured tokens will be left in the net.

The third check ensures that only immediate transitions are in the current net as described in section 4.1. On conversion, all timed transitions are converted to immediate transitions. PIPE allows the user to mark transitions as "single server" or "infinite server". A "single server" can only "reserve" a single batch of tokens, while an infinite server can reserve an infinite amount of batches of tokens. However, this only has an observable effect on timed transitions. Since those type of transitions do not exist in P/T or EN systems, we normalize these on conversion to single servers, as this graphically better represent transitions found in EN systems. The fourth check will ensure that all arc weights are 1. On conversion all arc weights are set to a single "default" token, regardless of what it was.

The last three checks require the user to change the net manually before conversion is possible. Self-loops as discussed earlier can have transitions that are enabled in P/T systems, but should not be enabled in EN systems. There is no elegant way of converting such loops. The following conversions were considered, but ultimately rejected:

- Since a transition t in a self-loop should never be enabled in an EN system, we could add an empty place p to ${}^\bullet t$. Since ${}^\bullet p = \emptyset$, no tokens will ever appear in p, and thus t will never be enabled. There are however two problems with this approach. First of all we don't know if t would ever be enabled. To find out if this is the case, the entire state space must be generated, and subsequently analysed for occurrence of t on an edge. This potentially an expensive operation for large nets. If we would not check this, and simply add an empty input place to t, we would add a place every time the convert button is clicked. This adds a lot of clutter. The second problem is that this would introduce a new

place in the net, while that place can not be easily removed with the undo button. Since we don't know why this self-loop was introduced, we don't know if this self-loop was intentional, and if the user wants us to correct it.

- Similar to the first option, we could remove this transition and all arcs connected to it. This has the same problem as the first option, as the removed transition and arcs can not easily be put back by clicking the undo button.

- If we would assume the self-loop was introduced on purpose, we could expand the self-loop to a sequence of 2 places and 2 transitions. This would significantly alter the sequential configuration graph, and we can't be sure the user added the self-loop on purpose. This change can also not be easily undone.

The other check ensures that the net contains no inhibitor arcs. We can not guarantee that between a place and transition there is only one normal arc, or one inhibitor arc. While it makes no sense that both exist between a place and transition, as it would always disable that transition, we would need to remove the inhibitor arc if a normal arc already existed. This has the same problem as already mentioned for self-loops: We can not easily undo this.

Lastly we check if no isolated transitions (${}^\bullet t = \emptyset$ or $t^\bullet = \emptyset$) exist. Such transitions cause problems when generating the SCG of an EN system, because they can nearly always occur. They are not automatically removed because of the same reason as why we do not make structural changes for the last two checks: The changes can not be easily undone. Instead, the user should remove the transition themselves, or complete the EN system they were creating.

If the net contains self-loops, inhibitor arcs, isolated transitions or already is a valid EN system, the convert button is greyed out and thus prevents the user from converting the net.

4.3 State space explorer

PIPE 5 already has a state space explorer as described in section 3.2.1, which can draw the sequential configuration graph.

To make the graph usable without hovering over places we first created an easier way to show the configuration in a specific node. When the Petri net has only places with capacity 1, the configuration can be displayed as a set of places that contain a token as shown in figure 2.4. This is by far the easiest way to interpret the information displayed on the node. When the capacity of one of the places is more than one, we have to display how many tokens are in each place. This is done by creating a tuple that displays the number of

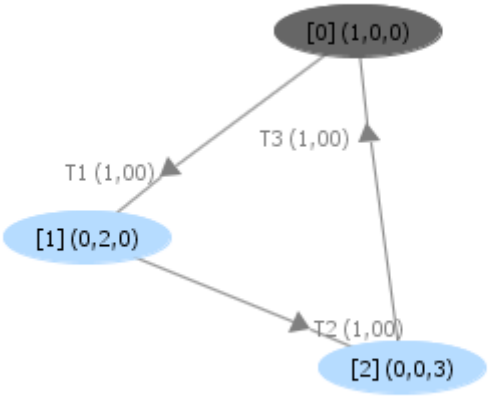


Figure 4.3: Reachability graph for the P/T system in figure 2.2.

tokens for each place in a fixed order as shown in figure 4.3. This allows for easy comparison between two states. When multiple types (or colours) of tokens exist within the Petri net, the number of tokens in each place is displayed as a tuple, with a fixed order for the type of tokens. A representation of a state with 2 token types and 3 places could be something like "$((0,1),(1,2),(3,0))$".

When the user hovers over a place, a box with a detailed view is displayed. Since there is a lot more room in this box each place is displayed on a new line, with the number and type of tokens behind it. To have an easier time figuring out what the initial state of a configuration graph is, the first node is now displayed in a different colour and labelled differently.

Previously, only a throughput rate was displayed on edges between states. For EN and P/T systems we are more interested in which transitions are fired. This information was not easily available in the old code base, and has been implemented from the ground up. Now, the transitions that could have been fired to go from state to state are displayed on the edge too.

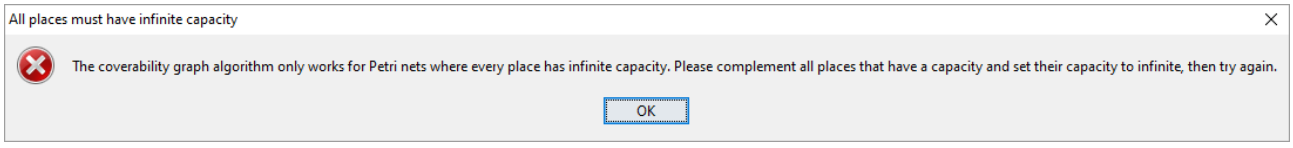


Figure 4.4: Error message shown when generating coverability graph when a place doesn't have infinite capacity.

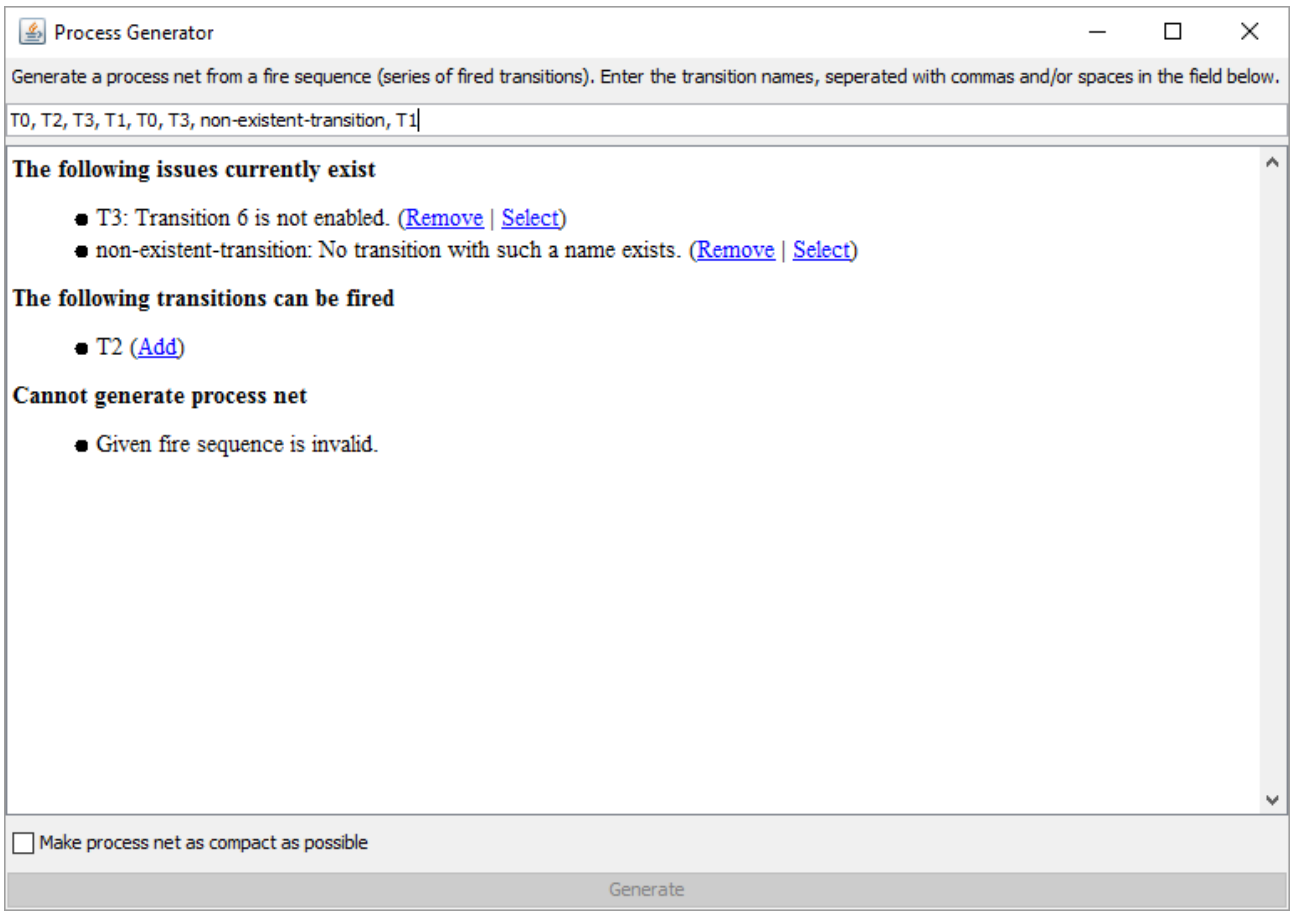


Figure 4.5: Process generator dialog for invalid fire sequence.

In PIPE 5 the coverability graph did not display ω when the number of tokens in a place could grow infinitely. This has been fixed in the current implementation. Furthermore, the algorithm that was used assumed that the capacity on all places was infinite, but this was never verified. Since this resulted in wrong coverability graphs for Petri nets with limited capacity, an error has been added when this is tried instead of showing a potentially wrong coverability graph. This error suggests complementing places with a capacity, then setting all capacities to infinite. (Figure 4.4)

4.4 Process generator

PIPE originally only supported modules that only work on a single Petri net. The process generator module must create new Petri nets, and to support this a new type of module had to be created. This new module base class is intended to be used for modules that modify Petri nets instead of analysing them. It provides an event listener (section 3.3) that the module can use to send custom events to the application. The application then acts appropriately to the custom event. This way we don't need to trust the module to leave the application in a consistent state, especially if something in the application is changed. This way there is also only one extra

module type required for any module that needs to change anything in the application, where otherwise controllers would need to be passed to the module to do it manually.

4.4.1 Dialog

When the user starts the module, they are presented with a dialog as shown in figure 4.5. In the top is an input field where the user can input a fire sequence manually, either by typing or by pasting text. Below that there is a field that shows information to the user. If the current fire sequence is invalid, the problems are displayed first. Below that suggestions for the next fired transition are given. Lastly an error message is shown with the exact reason why no process net can be generated in the current situation. At the bottom is a checkbox that allows the user to let the process net be more compact. The expansive version shows each transition in the fire sequence in its own column. This way it is easy to follow how the given fire sequence results in the process net that is displayed. The compact version moves each transition as far left as possible. This makes it easier to identify transitions that are sequential or concurrent. Finally there is a button that generated a process net in a new tab in the application.

Behind an error line two links are displayed. The first link allows the user to select the wrong transition, so they can more easily find it, and potentially correct it. The second link allows the user to remove the wrong transition. Behind a suggestion a link is displayed to automatically add the transition to the fire sequence. This way a fire sequence can quickly be built without needing to type.

4.4.2 Algorithm

The algorithm that is used simply simulates the given fire sequence on a copy of the source net. It assumes that when a transition is fired, all tokens are removed from all input places, and put in the output places. This way only one place corresponding to the place in the original net can contain tokens. Because of this, the module is restricted to nets where all places have capacity 1, such as EN systems.

"Contact" is a situation as shown in figure 2.5, when in a certain configuration C there is a transition t for which ${}^\bullet t \subseteq C$ (all input places contain tokens), but $t^\bullet \cap C \neq \emptyset$ (there are tokens in output places). The transition is thus not enabled, because its output places are not empty. Such a situation does not translate well to a process, because the transition is not necessarily disabled in a process in the same situation. This means that if a process net would be generated when there is contact in a net, one can find other sequences of transitions that can be fired in sequence in the process net, but not in the source net.

The module will generate the state space for a source net and make a collection of all possible configurations. For each configuration it will check for all transitions if there is contact. If it detects such a situation the

module will not generate a process net, but instead give an error message containing the configuration in which contact occurs. The solution is to complement the place in which contact occurs (Figure 4.2).

When we find that there is no contact in the given Petri net, and the given fire sequence is valid for that net, the process net is generated. The algorithm keeps track of the last place instance for each original place in the source net. Initially, it only contains the places that contained a token in the original net in the starting configuration. Then it will go through the fire sequence. For each transition it will find the transition with that name in the source net, and add a transition with an unique name based on the original name to the process net. For each arc in the original net pointing into that transition, the source place is looked up in the list of places that currently do not have outbound arcs in the process net, and an arc is added to the place that corresponds to the original place. If no such place exists, an error is shown, but this should not happen unless the source net is inconsistent. Then the same is done for outbound arcs from the transition. Again, if there is already a place in the process net with that name as an end point, an error is shown. This should not happen in a consistent net. Each of the places and transitions is positioned in the right column (horizontally), but not yet vertically.

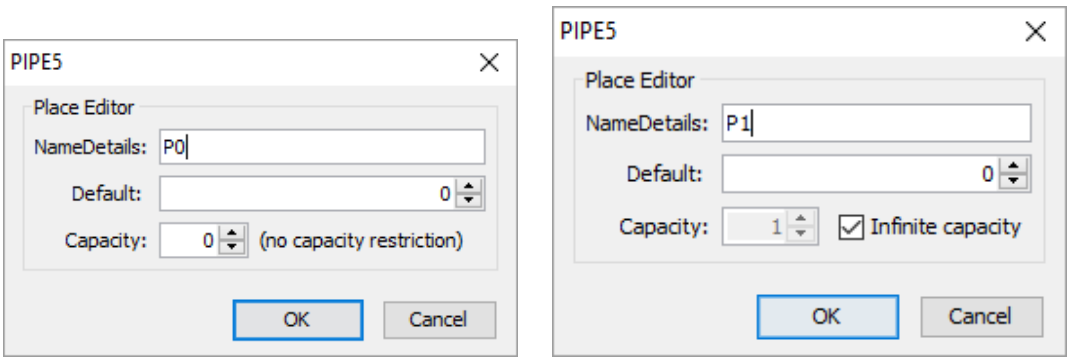
Then places and transitions are formatted. Per column, places and transitions are sorted alphabetically and layed out with a minimum distance between each other. On the second pass, each transition is checked. All inbound places are sorted vertically so that arcs don't cross over other places pointing into the same transition. Any places below that, who now would be too close to a moved place will be moved further down.

4.5 Bugs

4.5.1 Fatal bug

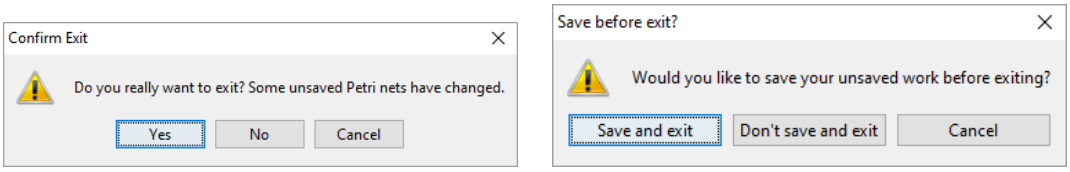
On Windows, PIPE did not compile or start, while there was no such issue on Linux or Mac. In this case, the pipe icon displayed in the title bar could not be loaded and crashed the application.

The problem was that a call to `getResource(..)` was done with a constant that relied on the system defined path separator. On Linux and Mac, this path separator is a forward slash (/). On Windows however, the path separator is a backslash (\). The `getResource(..)` function expects a relative path, or an absolute path beginning with a forward slash. This caused Java to search for a relative path instead of an absolute path on Windows, resulting in this crash. Simply replacing the prefix of the path fixed this issue.



(a) Old place dialog. (b) New place dialog.

Figure 4.6: Side-by-side comparison of the place edit dialog.



(a) Old exit dialog. (b) New exit dialog.

Figure 4.7: Side-by-side comparison of the application exit dialog.

4.5.2 Compilation errors

PIPE did not work correctly in Java 8. Compilation with the JDK for Java 8 would produce several errors. This was caused by two issues. One issue was with a test that tried to cast an `Object null` to a `T null`. Explicitly defining `T` in the template so this conversion was not made those tests pass. A bigger issue was that for previous versions of Java an outdated XML parser was loaded. This XML parser was already available natively in Java 7 and 8, so removing the dependency for this outdated XML parser fixed the issue.

4.5.3 Usability

Place editor dialog

When creating a Petri net, the user could right-click on a place to open an place edit dialog as shown in figure 4.6a. There were several things wrong with this dialog:

- The spinner modal behind "Capacity" didn't allow the user to click the arrows. Doing so would result in an exception in the background. Manually changing the capacity by selecting the number and typing did work.
- If the number of tokens in the starting configuration exceeded the capacity, an exception would be thrown in the background. However, part of the dialog was still saved.

- If a place had a capacity that was not infinite, there was no way to edit the place anymore. Any change would result in an exception being thrown in the background.
- The capacity spinner has a magic value "0", which encoded "infinite capacity". This in itself is counter-intuitive, but a result of how infinite capacity is stored in the underlying code. What was worse is that the hint behind the capacity spinner would not update when the value was updated, and would disappear if the capacity was not infinite.

To address these issues, the place edit dialog was reworked. The new dialog, as shown in figure 4.6b, replaced the capacity spinner with a combination of a spinner and a checkbox. When the checkbox for "infinite capacity" is checked, the capacity spinner is greyed out, as its value is irrelevant in that case. The spinner now has a minimum value of "1" and when the place does not have infinite capacity, that value is used instead. When the sum of tokens in the starting configuration of this place exceeds the capacity, an error is shown and the dialog is not saved before this issue is resolved.

Application exit dialog

The dialog boxes that were shown when closing a Petri net tab with unsaved changes, or the application while one or more tabs had unsaved changes, were inconsistent. The dialog box that was shown when closing the application (Figure 4.7a) had options "Yes" , "No" and "Cancel", while there were only two possible outcomes of the dialog: The application would close or it would not. The first and default option was the unsafe option, as it would close the application without saving. To understand what the buttons did, the user had to read the dialog very carefully. Dialogs like these are shown to prevent accidental lost work when the user is not paying attention, or distracted by something else, and such users are unlikely to carefully read the content of the dialog box.

The dialog was redesigned as shown in figure 4.7b. The new dialog has three clear interaction paths when this dialog is shown. The button "Save and exit" will save each unsaved Petri net using the filename it was already saved as, or shows a save dialog when that Petri net was never saved. To visualize to the user which Petri net is being saved, the application will change to the tab containing that Petri net. The application only exits after each Petri net has been saved, so the user does not lose work when aborting a save action by clicking cancel. The button "Don't save and exit" simply exits the application. The button "Cancel" closes the dialog and lets the user do what they want to do.

A similar issue existed with the standard template for error messages. This template had a "Yes" and a "No" button, but both options did nothing. The error dialog is primarily used to notify the user of something that is not possible, and should be manually fixed. Since these two buttons did not make any sense in that context, they have been replaced by a single "OK" button.

Load and save dialog

The load and save dialog used to be the most plain version the operating system offered. When the user entered a filename that didn't have an extension, correction code would be run later to add the extension. The plain version of a load/save dialog has an empty filename field on Windows, and "Untitled" in the filename field on linux/mac. This was confusing for users that did not know what the proper extension for their saved file was.

The new load and save dialogs give a hint in the filename field which extension is expected by displaying "*.ext", where ext is the actual extension. Operating systems that support this also limit the displayed files to only those files with the extension the dialog expects.

Disabled functionality

PIPE 5 contains several buttons that could also be found in the previous version of PIPE, but the implementation of the functionality that was provided by those buttons is missing from PIPE. Some of these buttons just did nothing when clicked, other buttons did display an error dialog encouraging the user to open a ticket on Github if they felt that this functionality is important. In either case, having a clickable button gives the user the expectation that this button does something. To prevent this, buttons of functionality that is not implemented have been disabled, so they appear grey and not clickable.

While PIPE still works with coloured Petri nets, and will open previously saved coloured Petri nets, the button that allowed users to add or remove tokens has been disabled. The reason for this is that this dialog, and the code that should add or remove token types from Petri nets, contains several bugs. When adding token types, the Petri net does not update weights on arcs to reflect this. When removing token types, the net can become inconsistent without a way to recover from this. When filling in a field to add a token type, the user has to press enter before closing the dialog, or the change will not take effect. Since this thesis is not focussed on these type of Petri nets, and the program will not be used at LIACS to analyse or build coloured Petri nets, the safest way to provide a robust and fool-proof build of PIPE is by disabling the functionality until it is fixed.

Numbering of components

PIPE 5 used to number places and transitions starting with P_0 and T_0. This is not incorrect, but it does not match the literature that is used at LIACS [7]. To be able to more easily recreate Petri nets found in this work numbering now starts with 1.

Export to image

While not really a bug, the functionality to export a Petri net to an image was not implemented. This functionality is useful, especially when using the Petri net outside of PIPE. This functionality was implemented by using the `Painter` that usually paints the Petri net on the canvas in the Petri net tab

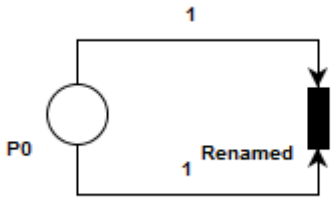


Figure 4.8: Inconsistent Petri net after renaming the transition and drawing an extra arc.

on a `Graphics2D` instance. First the bounding borders of the Petri net components were calculated to create the `Graphics2D` object, then the painter was offset so the top-left component would be painted in the top-left corner. Finally all components would be drawn on the `Graphics2D` object, and exported using Java's built-in image export function to a png file.

Added examples

PIPE is distributed with some examples that can be accessed via a menu item "Examples". The existing examples would not be displayed, because the placeholder text was never replaced with the actual internal directory where the examples were located. Some new examples were added based on the Petri nets used for exercises during the Theory of Concurrency course. These examples can be found at Appendix A.

4.5.4 Drawing

Renaming

Renaming items was not handled correctly in both the controller and the model. Arcs in the model have a hidden id that must be unique, which is based on the two components that arc is connected to. This id used to be only set when the arc was created. Whenever a new arc is created, it checks its generated id against all the existing ids in the model. The problem was that if a component was renamed after an arc was already attached to that component, a new arc could be connected. The The Petri net in figure 4.8 shows a net where originally the transition was named "T_0" and an arc was drawn between P_0 and that transition. The arc would get a hidden id "P_0 to T_0". After renaming T_0 to *Renamed*, the name of the arc would stay the same. The new drawn arc would have hidden id "P_0 to *Renamed*", which does not clash with the previously drawn arc. If a new arc would have been drawn without renaming, the arc would not have been accepted by PIPE.

The controller maintains a map that maps component names to the model. When a component is added, this map is updated with that component's name. Similar to the problem with the model, when a component was renamed, this map was not updated. This resulted in components not being able to

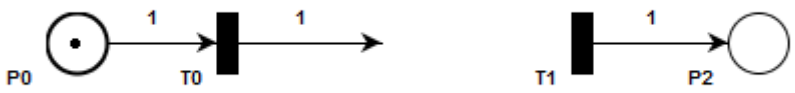


Figure 4.9: Inconsistent Petri net after deleting the middle place.

delete when they were renamed. When they were renamed back to their original name, they could be deleted again as expected.

When a component is renamed, it sends out a change event that other classes can listen to. The fix to both problems was to use the publisher-subscriber pattern to update the model and the controller when such a change event happened.

Deletion

Deleting components resulted in inconsistent Petri nets. Furthermore, undoing deletions did not result in getting the original Petri net back. As a side-effect of deleting components, other components could only be moved horizontally. This was caused by several issues, mostly relating to unfinished code.

The most visible problem was that deleting a component would only result in deleting that component, and deleting all outbound arcs of that component. All inbound arcs would remain, as shown in figure 4.9. The result is not a valid Petri net. The cause of this problem was that there was no code related to deleting inbound arcs at all. Simply extending the existing deletion code to also take into account inbound arcs solved that problem.

Undoing deletion was not possible at all. There were two problems causing this. The first problem was that the undoable events that were generated by deletion were not registered with the undoable event listener that is used by the undo and redo actions. Since those events were lost, they could not be undone. The other problem was that the deletion event assumed only one component was removed, even if that component was a place or transition with arcs attached to it. The solution to this problem was to pass on the right event listener, and to use the `MultipleEdit` class to group together deletions of multiple components.

`MultipleEdit` is a sequence of undoable events, in the order that they are performed. This sequence always leaves the Petri net in a consistent state. However, when undoing such a sequence, it did sometimes throw an exception before the whole `MultipleEdit` was undone. The problem here was that the events in `MultipleEdit` were undone and redone in the same arbitrary order.

Figure 4.10: T_0 appears enabled in animation mode, while the current structure of the Petri net does not allow T_0 to fire.

This proves to be problematic when, for example, an arc is added to the Petri net, but the components the arc should be connected to are not yet in the net. This problem was fixed by making the sequence an ordered list, where undoing the edit will traverse the list in reverse order. This will ensure the net is always in a consistent state, even when undoing an undoable event.

4.5.5 Animation

Caching

When animating a Petri net, the animated net could behave wrongly if it was animated before. In particular, transitions could appear enabled in animation mode while they were not. Firing transitions would not always result in the correct configuration.

The controller responsible for animating the Petri net has a cache for state changes. When a transition is fired that is not already in this cache, it is added to the cache. When in the same configuration at a later point in time, the cache is used to display the next state. Since the structure of a Petri net does not change during animation this works fine during a single animation run. However, when places are kept the same, but transitions or arcs are changed, the cached configuration changes of the old net are used if they were calculated then. This is visible in figure 4.10, where during the first animation mode the arcs were pointing from P_0 to T_0 and from T_0 to P_1. After firing T_0, the structure of the net was changed by reversing the arcs. T_0 now shows up as enabled, because the cache says it is, but from the structure of the net we know that this is not the case. The solution to this problem was just clearing the cache every time animation mode is started.

Multi-coloured nets

When a Petri net contained more than one type of token, only transitions where all input places contained all types of tokens were enabled, even if not all types of tokens were required. In figure 4.11 all three transitions should be enabled. Only T_0 shows up as enabled, because P_0 contains both types of tokens.

Arcs in PIPE may have an expression as weight instead of a natural number. It seems an attempt was done to prevent transitions from being able to fire if this expression evaluated to 0. The code that did

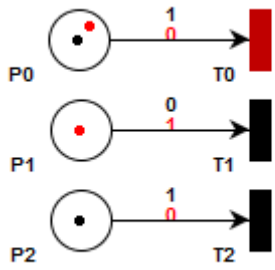


Figure 4.11: T_1 and T_2 do not show up as enabled.

this originally tested if there was a token type where the number of tokens was -1, likely a signal value of some kind. Later this test was rewritten to not enable if there was any token type that had a 0 token count. The code now tests if at least 1 token of any type is removed from the place, which works better. It is unknown if this is the intended enable rule, but the unit tests on which this is tested now pass again. In the end it does not matter much for this thesis, as coloured Petri nets are disabled for this build.

4.5.6 Graphical

Mouse events

When drawing arcs in PIPE, the end point of an edge would not update when hovering over other elements (places, transitions, labels, other edges), causing stuttering movement of the end point of the arc. Normally when mouse events are passed to an element on a canvas, the mouse event "bubbles" up through parents of that element until it reaches the canvas container. To perform a certain task, one binds an event handler to the most appropriate component, which will be called when the event bubbles up far enough.

In this case, the event never bubbled up, causing the event to never reach the event handler when the mouse hovered over anything else than the background of the canvas. The fix for this was to manually let the event bubble up through the parent, while translating to the coordinate system to that of the parent.

Capacity

Capacity is displayed in PIPE as a tooltip. The capacity that was shown in the tooltip would not change until something else was changed too. This was caused by a missing change event when the capacity was changed. Adding this made the tooltip update automatically.

Chapter 5

Conclusions

The goal of this project was to build a version of PIPE 5 that supported EN systems and a way to analyse them. During this project we created three modules. The first module analyses a Petri net and determines if it will behave like an EN system. This module can also transform Petri nets that look like EN systems to Petri nets that behave like EN systems. The second module can create useful configuration and coverability graphs for EN systems and P/T systems. The third module can create process nets based on a fire sequence for an EN system.

To make PIPE 5 usable by students, a large number of bugs in the parts of the program students are likely to use were fixed during this project. The drawing mode and animation mode should now be bug-free when being used for EN systems and P/T systems. To facilitate students using this program further, some small tweaks were made to improve the experience with the program, such as improved dialogs for loading and saving Petri nets and exiting the program, as well as visually disabling functionality that is not implemented.

The final build will be available as 4 Github projects[1]. The changes that were made during this project have been proposed as pull requests for the original PIPE project on Github[2]. At the time of writing, some of the changes have been approved, while others are still pending.

[1] https://github.com/search?q=user%3Adcduba+PIPE+fork%3Atrue
[2] https://github.com/search?q=user%3Asarahtattersall+PIPE

5.1 Future work

The support for coloured Petri nets in PIPE is currently not complete. It is currently not possible to add a guard function to transitions and to add explicit colour sets to places. Besides this, the way PIPE handles adding new token types is very error-prone. The dialog to add and remove tokens should be reworked and the graphical representation should be updated immediately. Changes to token types should be undoable and never put the Petri net in a state where it is no longer editable or can no longer be animated.

PIPE is still missing zoom functionality. Zooming allows someone to zoom out and view a larger portion of a Petri net on the screen, or zoom in to observe a small section of the Petri net. It is also missing the ability to create a branching process of the current Petri net. The button to use an algorithm to layout the Petri net is disabled as well. Finally, the option to export the Petri net as the portable Timed Net format, or the Postscript format for use outside PIPE are not implemented.

Views of arcs in particular are still messy, and a bug still exists where the control points of an arc do not always become invisible when the connected transition or place is deleted. The interface currently gives options to delete the control points on either end of an arc. While the bug has been fixed that actually deleted these points, resulting in an arc with no end point, the option should still be made invisible. Similarly splitting the control point at the end of an arc currently yields an invisible runtime error. The option should either be hidden, or split the edge before the end point.

Bibliography

[1] Petri-lld on sourceforge. https://sourceforge.net/projects/petrilld/files/petrilld/. Accessed: 21 August 2016.

[2] Petri nets world: Complete overview of petri nets tools database. http://www.informatik.uni-hamburg.de/TGI/PetriNets/tools/complete_db.html. Accessed: 21 August 2016.

[3] Platform independent petri net editor download on sourceforge. https://sourceforge.net/projects/pipe2/. Accessed: 15 August 2016.

[4] Bonet, P., Lladó, C. M., Puijaner, R., and Knottenbelt, W. J. PIPE v2. 5: A Petri net tool for performance modelling. In *Proc. 23rd Latin American Conference on Informatics (CLEI 2007)* (2007).

[5] Chiola, G., Marsan, M. A., Balbo, G., and Conte, G. Generalized stochastic Petri nets: A definition at the net level and its implications. *IEEE Transactions on software engineering 19*, 2 (1993), pp. 89–107.

[6] de Zwijger, W. Demonstration tools for petri nets. Bachelor thesis, Leiden University, March 2010.

[7] Engelfriet, J., and Rozenberg, G. Lecture notes Theory of Concurrency, 1998.

[8] Esparza, J., and Meyer, P. Petri nets - homework 1. https://www7.in.tum.de/um/courses/petri/SS2015/exercises/ex1.pdf. Accessed: 15 August 2016.

[9] Jensen, K. *Coloured Petri Nets: Basic Concepts, Analysis Methods and Practical Use*, vol. 1. Springer Science & Business Media, May 1997.

[10] Larsen, K. G. *Computer Aided Verification: 3rd International Workshop, CAV'91*. Springer Science & Business Media, April 1992.

[11] Lomazova, I. A. On occurrence net semantics for Petri nets with contacts. In *International Symposium on Fundamentals of Computation Theory* (1997), Springer, pp. 317–328.

[12] McGill University. Petri nets assignment. http://msdl.cs.mcgill.ca/people/hv/teaching/MoSIS/assignments/PN. Accessed: 15 August 2016.

[13] Reisig, W., and Rozenberg, G. Lectures on Petri nets, LNCS 1491, 1492, 1998.

[14] Rozenberg, G. *Advances in Petri Nets 1987*, vol. 266. Springer Science & Business Media, 1987.

[15] Tattersall, S. PIPE — The great re-plumbing. Master thesis, Imperial College London, June 2014.

Appendix A

Added examples

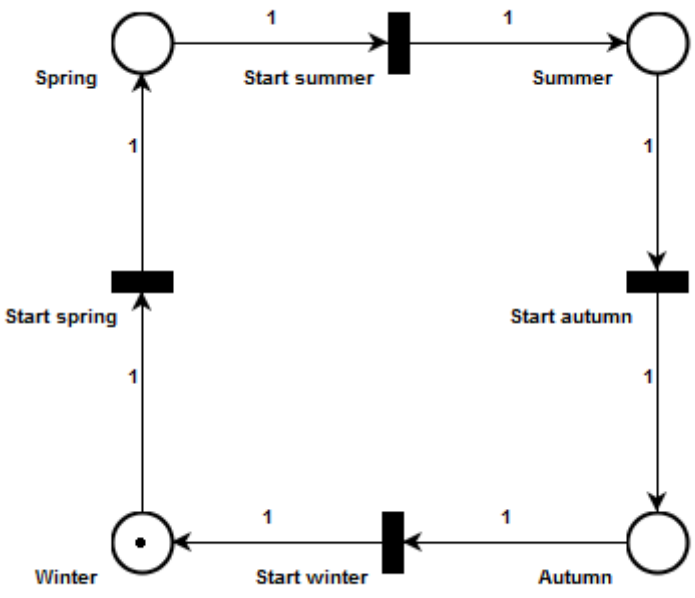


Figure A.1: EN system with seasons

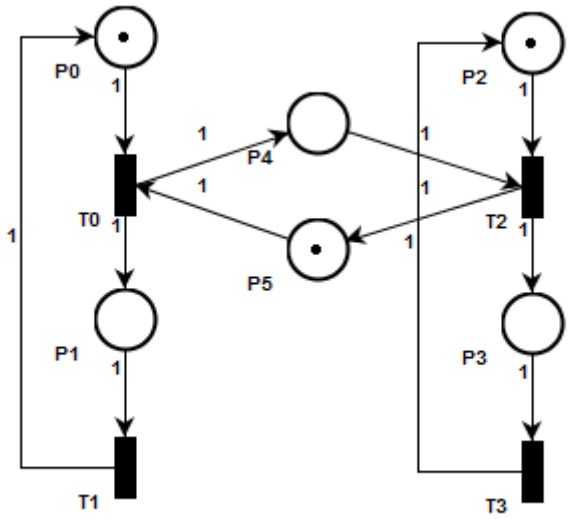


Figure A.2: "Producer-consumer" EN system without contact

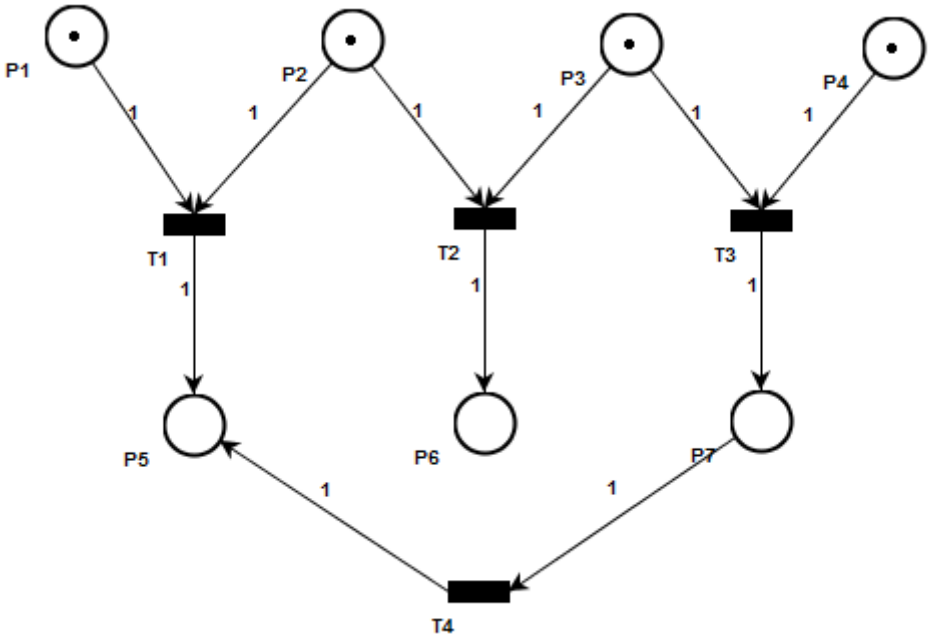


Figure A.3: EN system with conflict and confusion

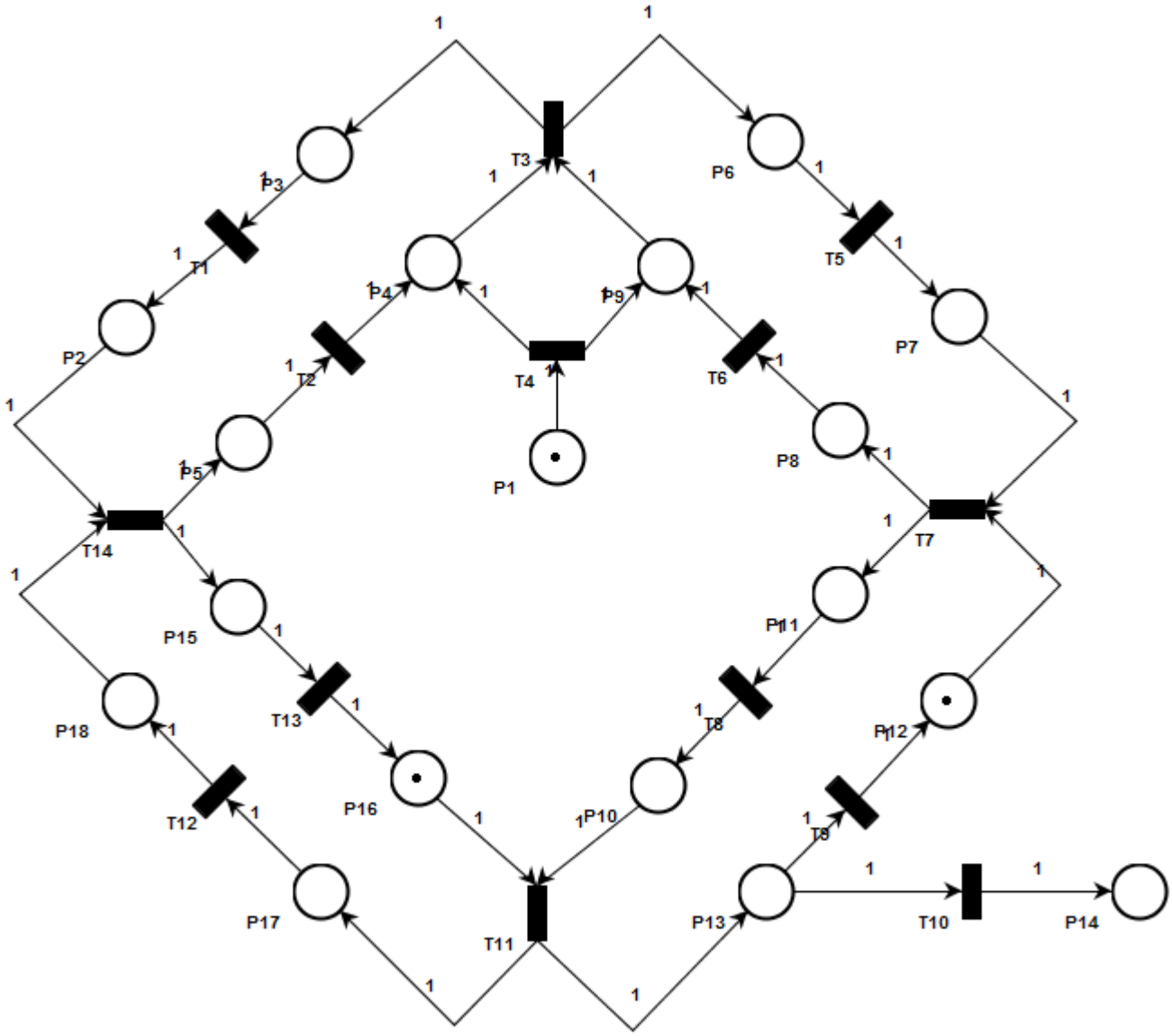


Figure A.4: Large EN system without contact

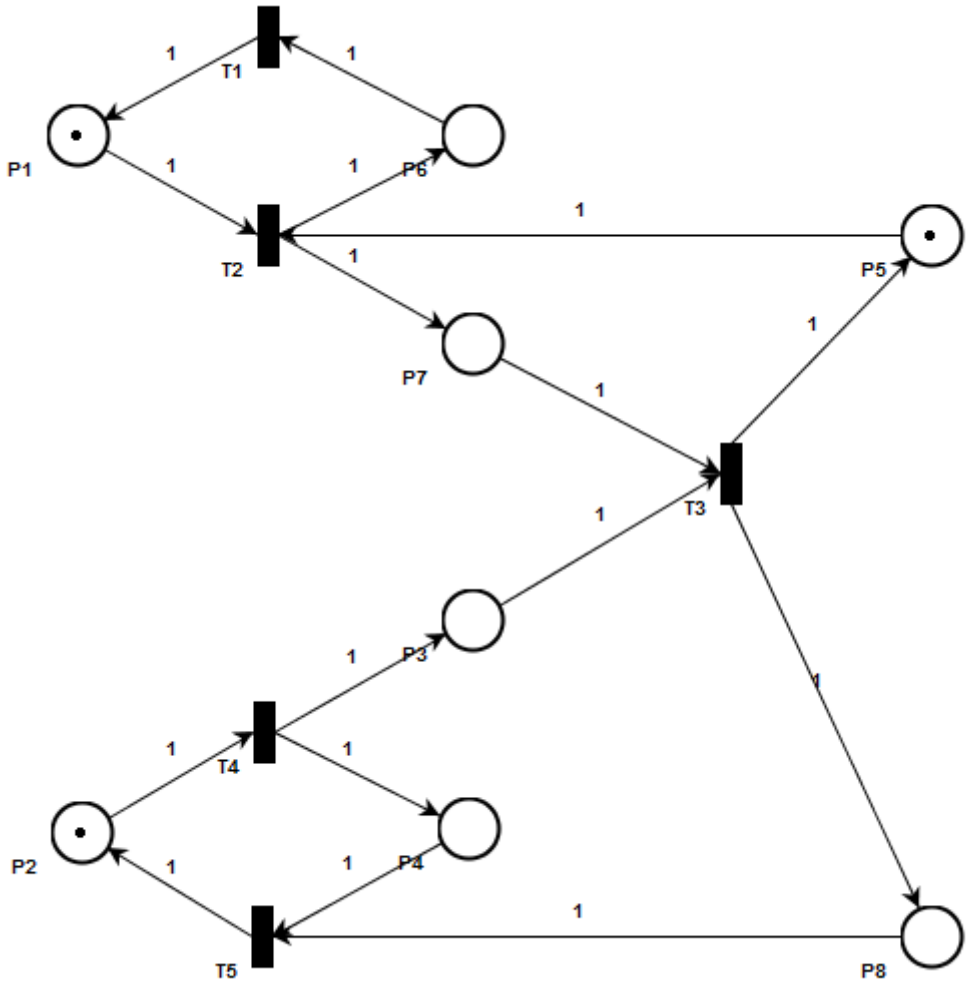


Figure A.5: EN system without contact

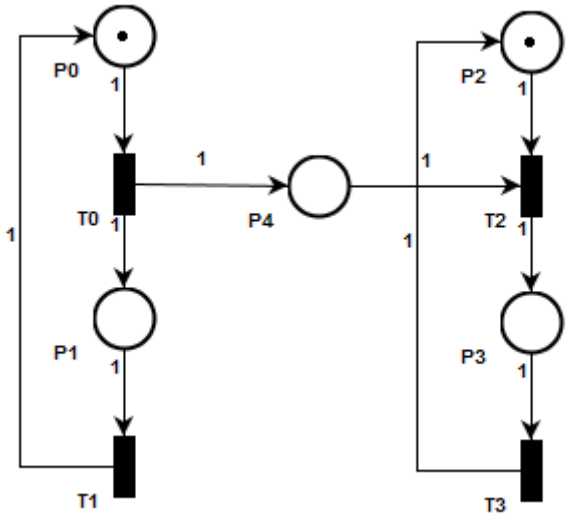


Figure A.6: "Producer-consumer" EN system with contact

Appendix B

Source code

This appendix contains the source code of the three modules that were added. During this project, changes were made in 118 files scattered over 4 Github projects. Listing all these changes in this appendix would create an unreadable mess. The reader is encouraged to review the changes that were made on Github[1].

B.1 EN module

B.1.1 ENValidator.form

```
<?xml version="1.0" encoding="UTF-8"?>
<form xmlns="http://www.intellij.com/uidesigner/form/" version="1" bind-to-class="pipe.gui.validation.ENValidator">
  <grid id="27dc6" binding="mainPanel" layout-manager="FormLayout">
    <rowspec value="center:max(d;4px):noGrow"/>
    <rowspec value="top:4dlu:noGrow"/>
    <rowspec value="center:max(d;4px):noGrow"/>
    <rowspec value="top:4dlu:noGrow"/>
    <rowspec value="center:d:grow"/>
    <rowspec value="top:4dlu:noGrow"/>
    <rowspec value="center:max(d;4px):noGrow"/>
    <rowspec value="top:4dlu:noGrow"/>
    <rowspec value="center:max(d;4px):noGrow"/>
    <colspec value="fill:d:grow"/>
    <constraints>
      <xy x="20" y="19" width="484" height="551"/>
    </constraints>
    <properties/>
    <border type="none"/>
    <children>
      <splitpane id="ec84">
        <constraints>
          <grid row="2" column="0" row-span="1" col-span="1" vsize-policy="3" hsize-policy="3" anchor="1" fill="1" indent="0" use-parent-layout="false">
            <preferred-size width="200" height="200"/>
          </grid>
          <forms defaultalign-vert="false"/>
        </constraints>
        <properties>
```

[1] https://github.com/search?q=user%3Adcduba+PIPE+fork%3Atrue

```
  <continuousLayout value="true"/>
  <dividerSize value="0"/>
  <resizeWeight value="1.0"/>
</properties>
<border type="none"/>
<children>
  <grid id="62816" layout-manager="GridLayoutManager" row-count="9" column-count="1" same-size-horizontally="false" same-size-vertically="false" hgap="-1
      " vgap="-1">
    <margin top="0" left="0" bottom="0" right="0"/>
    <constraints>
      <splitpane position="left"/>
    </constraints>
    <properties/>
    <border type="none"/>
    <children>
      <component id="f2919" class="javax.swing.JLabel">
        <constraints>
          <grid row="0" column="0" row-span="1" col-span="1" vsize-policy="0" hsize-policy="0" anchor="8" fill="0" indent="0" use-parent-layout="false"/>
        </constraints>
        <properties>
          <text value="EN Validation checks"/>
        </properties>
      </component>
      <vspacer id="b3828">
        <constraints>
          <grid row="1" column="0" row-span="1" col-span="1" vsize-policy="6" hsize-policy="1" anchor="0" fill="2" indent="0" use-parent-layout="false"/>
        </constraints>
      </vspacer>
      <component id="7f92" class="javax.swing.JCheckBox" binding="capacityCheck">
        <constraints>
          <grid row="2" column="0" row-span="1" col-span="1" vsize-policy="0" hsize-policy="3" anchor="8" fill="0" indent="0" use-parent-layout="false"/>
        </constraints>
        <properties>
          <enabled value="false"/>
          <text value="Places have capacity 1"/>
        </properties>
      </component>
      <component id="dbb63" class="javax.swing.JCheckBox" binding="tokenCheck">
        <constraints>
          <grid row="3" column="0" row-span="1" col-span="1" vsize-policy="0" hsize-policy="3" anchor="8" fill="0" indent="0" use-parent-layout="false"/>
        </constraints>
        <properties>
          <enabled value="false"/>
          <text value="Places contain at most 1 token"/>
        </properties>
      </component>
      <component id="e39c8" class="javax.swing.JCheckBox" binding="simplicityCheck">
        <constraints>
          <grid row="4" column="0" row-span="1" col-span="1" vsize-policy="0" hsize-policy="3" anchor="8" fill="0" indent="0" use-parent-layout="false"/>
        </constraints>
        <properties>
          <enabled value="false"/>
          <text value="Net contains only places and immediate transitions"/>
        </properties>
      </component>
      <component id="69bcb" class="javax.swing.JCheckBox" binding="selfloopCheck">
        <constraints>
          <grid row="6" column="0" row-span="1" col-span="1" vsize-policy="0" hsize-policy="3" anchor="8" fill="0" indent="0" use-parent-layout="false"/>
        </constraints>
        <properties>
          <enabled value="false"/>
          <selected value="false"/>
          <text value="Net contains no self-loops"/>
        </properties>
        <clientProperties>
          <hideActionText class="java.lang.Boolean" value="false"/>
          <html.disable class="java.lang.Boolean" value="false"/>
        </clientProperties>
      </component>
      <component id="9d68c" class="javax.swing.JCheckBox" binding="arcWeightCheck">
```

```
            <constraints>
              <grid row="5" column="0" row-span="1" col-span="1" vsize-policy="0" hsize-policy="3" anchor="8" fill="0" indent="0" use-parent-layout="false"/>
            </constraints>
            <properties>
              <enabled value="false"/>
              <text value="Arcs have weight 1"/>
            </properties>
          </component>
          <component id="e164c" class="javax.swing.JCheckBox" binding="inhibitorArcCheck">
            <constraints>
              <grid row="7" column="0" row-span="1" col-span="1" vsize-policy="0" hsize-policy="3" anchor="8" fill="0" indent="0" use-parent-layout="false"/>
            </constraints>
            <properties>
              <enabled value="false"/>
              <text value="Net contains no inhibitor arcs"/>
            </properties>
          </component>
          <component id="2c73" class="javax.swing.JCheckBox" binding="isolationCheck">
            <constraints>
              <grid row="8" column="0" row-span="1" col-span="1" vsize-policy="0" hsize-policy="3" anchor="8" fill="0" indent="0" use-parent-layout="false"/>
            </constraints>
            <properties>
              <enabled value="false"/>
              <text value="No isolated transitions"/>
            </properties>
          </component>
        </children>
      </grid>
      <grid id="9b419" layout-manager="GridLayoutManager" row-count="1" column-count="1" same-size-horizontally="false" same-size-vertically="false" hgap="-1
          " vgap="-1">
        <margin top="0" left="0" bottom="0" right="0"/>
        <constraints>
          <splitpane position="right"/>
        </constraints>
        <properties/>
        <border type="none"/>
        <children>
          <component id="cod24" class="javax.swing.JLabel" binding="validationIcon">
            <constraints>
              <grid row="0" column="0" row-span="1" col-span="1" vsize-policy="4" hsize-policy="4" anchor="0" fill="0" indent="0" use-parent-layout="false"/>
            </constraints>
            <properties>
              <text value="En validation icon"/>
              <toolTipText value="Does this Petri net validate as an EN system"/>
            </properties>
          </component>
        </children>
      </grid>
    </children>
  </splitpane>
  <grid id="ea027" layout-manager="GridLayoutManager" row-count="1" column-count="1" same-size-horizontally="false" same-size-vertically="false" hgap="-1"
      vgap="-1">
    <margin top="4" left="4" bottom="4" right="4"/>
    <constraints>
      <grid row="0" column="0" row-span="1" col-span="1" vsize-policy="0" hsize-policy="0" anchor="9" fill="0" indent="0" use-parent-layout="false"/>
      <forms defaultalign-vert="false"/>
    </constraints>
    <properties/>
    <border type="none"/>
    <children>
      <component id="a3fc1" class="javax.swing.JLabel" binding="informationLabel">
        <constraints>
          <grid row="0" column="0" row-span="1" col-span="1" vsize-policy="0" hsize-policy="0" anchor="8" fill="0" indent="0" use-parent-layout="false"/>
        </constraints>
        <properties>
          <text value="If a P/T net holds the following properties , the net behaves as an elementary net (EN)."/>
        </properties>
      </component>
    </children>
  </grid>
```

```
<grid id="115f6" layout-manager="GridLayoutManager" row-count="1" column-count="1" same-size-horizontally="false" same-size-vertically="false" hgap="-1"
      vgap="-1">
  <margin top="4" left="4" bottom="4" right="4"/>
  <constraints>
    <grid row="4" column="0" row-span="2" col-span="1" vsize-policy="4" hsize-policy="3" anchor="0" fill="3" indent="0" use-parent-layout="false"/>
    <forms defaultalign-vert="false"/>
  </constraints>
  <properties/>
  <border type="none"/>
  <children>
    <scrollpane id="68dd0">
      <constraints>
        <grid row="0" column="0" row-span="1" col-span="1" vsize-policy="7" hsize-policy="7" anchor="0" fill="3" indent="0" use-parent-layout="false">
          <preferred-size width="-1" height="500"/>
        </grid>
      </constraints>
      <properties>
        <enabled value="true"/>
        <verticalScrollBarPolicy value="22"/>
      </properties>
      <border type="none"/>
      <children>
        <component id="440be" class="javax.swing.JTextArea" binding="resultPanel">
          <constraints/>
          <properties>
            <editable value="false"/>
            <enabled value="true"/>
            <lineWrap value="true"/>
          </properties>
        </component>
      </children>
    </scrollpane>
  </children>
</grid>
<splitpane id="ad940">
  <constraints>
    <grid row="8" column="0" row-span="1" col-span="1" vsize-policy="3" hsize-policy="3" anchor="0" fill="1" indent="0" use-parent-layout="false">
      <preferred-size width="200" height="200"/>
    </grid>
    <forms/>
  </constraints>
  <properties>
    <dividerSize value="0"/>
    <enabled value="false"/>
  </properties>
  <border type="none"/>
  <children>
    <component id="b88bc" class="javax.swing.JButton" binding="convertButton">
      <constraints>
        <splitpane position="right"/>
      </constraints>
      <properties>
        <preferredSize width="500" height="32"/>
        <selected value="false"/>
        <text value="Convert"/>
        <visible value="true"/>
      </properties>
    </component>
    <component id="3b6e8" class="javax.swing.JButton" binding="refreshButton">
      <constraints>
        <splitpane position="left"/>
      </constraints>
      <properties>
        <text value=""/>
        <toolTipText value="Validate again"/>
      </properties>
    </component>
  </children>
</splitpane>
</children>
```

```
  </grid>
  <buttonGroups>
    <group name="loadGroup">
      <member id="3408b"/>
      <member id="35f8e"/>
      <member id="ac1b4"/>
    </group>
  </buttonGroups>
</form>
```

B.1.2 ENValidator.java

```
package pipe.gui.validation;

import org.apache.commons.collections.CollectionUtils;

import uk.ac.imperial.pipe.exceptions.InvalidRateException;
import uk.ac.imperial.pipe.models.petrinet.*;

import java.io.File;
import javax.swing.*;
import java.awt.FileDialog;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.IOException;
import java.net.URL;
import java.util.*;
import java.util.concurrent.ExecutionException;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;

/**
 * Performs the exploration and steady state analysis of a Petri net.
 * Displays useful performance analysis metrics
 */
public class ENValidator {
    private static final String PLACE_INTRO = "## Places ##\n";
    private static final String TRANS_INTRO = "## Transitions ##\nTransitions in EN systems are immediate, have no priority and are single servers.\n";
    private static final String ARC_INTRO = "## Arcs ##\nArcs in EN systems can not have a weight (other than 1). Self-loops are not allowed, because in an EN
        system a transition is only enabled if all output places do not contain a token, while in P/T systems a transition is enabled if the resulting number
        of tokens in output places does not exceed the capacity.\n";

    private JPanel mainPanel;

    private JLabel validationIcon;

    private JCheckBox capacityCheck;

    private JCheckBox tokenCheck;

    private JCheckBox simplicityCheck;

    private JCheckBox selfloopCheck;

    private JCheckBox arcWeightCheck;

    private JCheckBox inhibitorArcCheck;

    private JCheckBox isolationCheck;

    private JLabel informationLabel;

    private JButton convertButton;

    private JButton refreshButton;

    private JTextArea resultPanel;
```

```
private PetriNet net;

/**
 * Sets up the UI
 */
private void setUp() {
    ImageIcon icon = new ImageIcon(getImageURL("refresh.png"));
    refreshButton.setIcon(icon);

    //Select correct checkboxes upon opening form
    validateNet();

    convertButton.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {
            convertNet();

            validateNet();
        }
    });

    refreshButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                    validateNet();
            }
    });
}

private URL getImageURL(String name) {
    return this.getClass().getResource("/" + "images" + File.separator + name);
}

private void convertNet() {
    Collection<Place> places = net.getPlaces();
    Collection<Transition> transitions = net.getTransitions();
    Collection<InboundArc> inboundArcs = net.getInboundArcs();
    Collection<OutboundArc> outboundArcs = net.getOutboundArcs();

    for(Place place : places) {
            if(place.getNumberOfTokensStored() > 1) {
                    //Working on a copy, so the HashMap does not change while changing the actual tokens
                    for(Map.Entry<String,Integer> entry : new HashMap<String,Integer>(place.getTokenCounts()).entrySet()) {
                            place.removeAllTokens(entry.getKey());
                    }
                    place.setTokenCount("Default", 1);
            }
            place.setCapacity(1);
    }

    for(Transition transition : transitions) {
            transition.setPriority(1);
            transition.setInfiniteServer(false);
            transition.setTimed(false);
    }

    for(InboundArc inboundArc : inboundArcs) {
            for(Map.Entry<String,String> entry : inboundArc.getTokenWeights().entrySet()) {
                    inboundArc.setWeight(entry.getKey(), "");
            }
                    inboundArc.setWeight("Default", "1");
    }
    for(OutboundArc outboundArc : outboundArcs) {
            for(Map.Entry<String,String> entry : outboundArc.getTokenWeights().entrySet()) {
                    outboundArc.setWeight(entry.getKey(), "");
            }
            outboundArc.setWeight("Default", "1");
    }
}
```

```
private void validateNet() {
    Collection<Place> places = net.getPlaces();
    Collection<Transition> transitions = net.getTransitions();
    Collection<InboundArc> inboundArcs = net.getInboundArcs();
    Collection<OutboundArc> outboundArcs = net.getOutboundArcs();
    ImageIcon icon;

    boolean hasENCapacity = true;
    boolean hasENTokens = true;
    boolean isSimpleNet = true;
    boolean hasSelfloops = false;
    boolean hasArcWeights = false;
    boolean hasInhibitorArcs = false;
    boolean hasIsolatedTransitions = false;

    String validationResult = "";

    Set<Transition> inboundConnectedTransitions = new HashSet<>();
    Set<Transition> outboundConnectedTransitions = new HashSet<>();

    /* An EN system consists of only simple places and transitions, where places
     * have a capacity of 1, and at most 1 token. A P/T system with these properties
     * only behaves as an EN system if it contains no selfloops.
     */
    validationResult += PLACE_INTRO;
    for(Place place : places) {
        if(!place.hasCapacityRestriction() || place.getCapacity() > 1) {
            validationResult += "- Place " + place.getId() + " has " + (place.hasCapacityRestriction() ? ("a capacity of " + place.getCapacity()) : "
                an infinite capacity") + " and should have capacity 1.\n";
            hasENCapacity = false;
        }
        if(place.getNumberOfTokensStored() > 1) {
            validationResult += "- Place " + place.getId() + " has " + place.getNumberOfTokensStored() + " tokens and should have at most 1 token.\n"
                ;
            hasENTokens = false;
        }
    }
    if(hasENCapacity && hasENTokens) {
        validationResult += "- None\n";
    }

    validationResult += "\n" + TRANS_INTRO;
    for(Transition transition : transitions) {
        if(transition.isTimed() || transition.getPriority() != 1 || transition.getRate().getRateType() != RateType.NORMAL_RATE || transition.
            isInfiniteServer()) {
            validationResult += "- Transition " + transition.getId() + " is not a \"simple\" transition.\n";
            isSimpleNet = false;
        }
    }
    if(isSimpleNet) {
        validationResult += "- None\n";
    }

    validationResult += "\n" + ARC_INTRO;
    for(InboundArc inboundArc : inboundArcs) {
        Place start = inboundArc.getSource();
        Transition middle = inboundArc.getTarget();

        for(OutboundArc outboundArc : outboundArcs) {
            if(outboundArc.getSource() == middle && outboundArc.getTarget() == start) {
                validationResult += "- Place " + start.getId() + " loops to itself via transition " + middle.getId() + ".\n";
                hasSelfloops = true;
            }
        }
    }
    for(InboundArc inboundArc : inboundArcs) {
        Place start = inboundArc.getSource();
        Transition end = inboundArc.getTarget();
        if(inboundArc instanceof InboundInhibitorArc) {
            validationResult += "- An inhibitor arc between " + start.getId() + " and " + end.getId() + " exists while none should exist.";
```

```
                hasInhibitorArcs = true;
        }
        for(Map.Entry<String,String> entry : inboundArc.getTokenWeights().entrySet()) {
                if(entry.getKey().equals("Default") && !entry.getValue().equals("1")) {
                        validationResult += "- The arc between " + start.getId() + " and " + end.getId() + " has weight \"" + entry.getValue() + "\"
                                while a weight of \"1\" is expected.\n";
                        hasArcWeights = true;
                } else if(!entry.getKey().equals("Default") && !entry.getValue().equals("") && !entry.getValue().equals("0")) {
                        validationResult += "- The arc between " + start.getId() + " and " + end.getId() + " has a weight for tokens of type " + entry.
                                getKey() + " while none is expected.\n";
                        hasArcWeights = true;
                }
        }

        inboundConnectedTransitions.add(end);
}
for(OutboundArc outboundArc : outboundArcs) {
        Transition start = outboundArc.getSource();
        Place end = outboundArc.getTarget();
        for(Map.Entry<String,String> entry : outboundArc.getTokenWeights().entrySet()) {
                if(entry.getKey().equals("Default") && !entry.getValue().equals("1")) {
                        validationResult += "- The arc between " + start.getId() + " and " + end.getId() + " has weight \"" + entry.getValue() + "\"
                                while a weight of \"1\" is expected.\n";
                        hasArcWeights = true;
                } else if(!entry.getKey().equals("Default") && !entry.getValue().equals("") && !entry.getValue().equals("0")) {
                        validationResult += "- The arc between " + start.getId() + " and " + end.getId() + " has a weight for tokens of type " + entry.
                                getKey() + " while none is expected.\n";
                        hasArcWeights = true;
                }
        }

        outboundConnectedTransitions.add(start);
}

        inboundConnectedTransitions.retainAll(outboundConnectedTransitions);
if(transitions.size() > inboundConnectedTransitions.size()) {
        hasIsolatedTransitions = true;
        transitions.removeAll(inboundConnectedTransitions);
        for(Transition transition : transitions) {
                validationResult += "- Transition " + transition.getId() + " should have both inbound and outbound arcs.\n";
        }
}


if(!hasSelfloops && !hasArcWeights && !hasInhibitorArcs && !hasIsolatedTransitions) {
        validationResult += "- None\n";
}

//The checkboxes give easy confirmation which conditions are not met
capacityCheck.setSelected(hasENCapacity);
tokenCheck.setSelected(hasENTokens);
simplicityCheck.setSelected(isSimpleNet);
selfloopCheck.setSelected(!hasSelfloops);
arcWeightCheck.setSelected(!hasArcWeights);
inhibitorArcCheck.setSelected(!hasInhibitorArcs);
isolationCheck.setSelected(!hasIsolatedTransitions);

//We cannot convert if it contains selfloops, because we do not know how the user wants to solve this
if(hasSelfloops) {
        convertButton.setEnabled(false);
        convertButton.setToolTipText("Please remove all selfloops before converting to an EN system");
        validationResult += "\n\nPlease remove all selfloops before converting to an EN system.";
} else if(hasInhibitorArcs) {
        convertButton.setEnabled(false);
        convertButton.setToolTipText("Please remove all inhibitor arcs before converting to an EN system");
        validationResult += "\n\nPlease remove all inhibitor arcs before converting to an EN system.";
} else if(hasIsolatedTransitions) {
        convertButton.setEnabled(false);
        convertButton.setToolTipText("Please remove all transitions that do not have both an inbound and outbound arc.");
        validationResult += "\n\nPlease remove all transitions that do not have both an inbound and outbound arc.";
```

```
        } else if(hasENCapacity && hasENTokens && isSimpleNet && !hasSelfloops && !hasArcWeights && !hasIsolatedTransitions) {
                convertButton.setEnabled(false);
                convertButton.setToolTipText("System is already an EN system");
        } else {
                convertButton.setEnabled(true);
                convertButton.setToolTipText("Click to resolve all problems highlighted above");
        }

        //Easy confirmation if system is actually EN system via large icon with checkmark/cross
        validationIcon.setText(null);
        if(hasENCapacity && hasENTokens && isSimpleNet && !hasSelfloops && !hasArcWeights && !hasInhibitorArcs && !hasIsolatedTransitions) {
                icon = new ImageIcon(getImageURL("envalid.png"));
        } else {
                icon = new ImageIcon(getImageURL("eninvalid.png"));
        }
        validationIcon.setIcon(icon);

        resultPanel.setText(validationResult);
    }

    public ENValidator(PetriNet petriNet, FileDialog fileDialog) {
        this.net = petriNet;
        setUp();
    }

    /**
     * Main method for running this externally without PIPE
     *
     * @param args           command line arguments
     */
    public static void main(String[] args) {
        //Does not make sense to run this as a stand-alone program
    }

    public JPanel getMainPanel() {
        return mainPanel;
    }
}
```

B.2 Process generator module

B.2.1 ProcessGenerator.form

```
<?xml version="1.0" encoding="UTF-8"?>
<form xmlns="http://www.intellij.com/uidesigner/form/" version="1" bind-to-class="pipe.gui.process.ProcessGenerator">
  <grid id="27dc6" binding="mainPanel" layout-manager="GridLayoutManager" row-count="6" column-count="1" same-size-horizontally="false" same-size-vertically="
        false" hgap="-1" vgap="-1">
    <margin top="0" left="0" bottom="0" right="0"/>
    <constraints>
      <xy x="20" y="20" width="911" height="561"/>
    </constraints>
    <properties/>
    <border type="none"/>
    <children>
      <grid id="8949b" layout-manager="GridLayoutManager" row-count="1" column-count="1" same-size-horizontally="false" same-size-vertically="false" hgap="-1"
            vgap="-1">
        <margin top="4" left="4" bottom="4" right="4"/>
        <constraints>
          <grid row="0" column="0" row-span="1" col-span="1" vsize-policy="0" hsize-policy="7" anchor="1" fill="1" indent="0" use-parent-layout="false"/>
        </constraints>
        <properties/>
        <border type="none"/>
        <children>
          <component id="1043b" class="javax.swing.JLabel">
            <constraints>
```

```
            <grid row="0" column="0" row-span="1" col-span="1" vsize-policy="0" hsize-policy="7" anchor="8" fill="0" indent="0" use-parent-layout="false"/>
          </constraints>
          <properties>
            <text value="Generate a process net from a fire sequence (series of fired transitions). Enter the transition names, seperated with commas and/or
                spaces in the field below."/>
          </properties>
        </component>
      </children>
    </grid>
    <component id="e608f" class="javax.swing.JTextField" binding="fireSequenceTextField">
      <constraints>
        <grid row="1" column="0" row-span="1" col-span="1" vsize-policy="0" hsize-policy="7" anchor="9" fill="1" indent="0" use-parent-layout="false">
          <preferred-size width="150" height="-1"/>
        </grid>
      </constraints>
      <properties>
        <enabled value="true"/>
        <toolTipText value="Input a fire sequence"/>
      </properties>
    </component>
    <scrollpane id="b8176" binding="infoScrollPane">
      <constraints>
        <grid row="2" column="0" row-span="1" col-span="1" vsize-policy="7" hsize-policy="7" anchor="0" fill="3" indent="0" use-parent-layout="false">
          <minimum-size width="-1" height="400"/>
        </grid>
      </constraints>
      <properties>
        <enabled value="true"/>
        <verticalScrollBarPolicy value="22"/>
        <visible value="true"/>
      </properties>
      <border type="none"/>
      <children>
        <component id="783ce" class="javax.swing.JTextPane" binding="infoTextPane">
          <constraints/>
          <properties>
            <contentType value="text/html"/>
            <editable value="false"/>
            <minimumSize width="6" height="400"/>
          </properties>
        </component>
      </children>
    </scrollpane>
    <component id="920da" class="javax.swing.JButton" binding="generateButton">
      <constraints>
        <grid row="5" column="0" row-span="1" col-span="1" vsize-policy="3" hsize-policy="3" anchor="0" fill="1" indent="0" use-parent-layout="false"/>
      </constraints>
      <properties>
        <text value="Generate"/>
        <toolTipText value="Generate a process net in a new tab"/>
      </properties>
    </component>
    <component id="ee8df" class="javax.swing.JCheckBox" binding="condenseCheckBox">
      <constraints>
        <grid row="3" column="0" row-span="1" col-span="1" vsize-policy="0" hsize-policy="3" anchor="8" fill="0" indent="0" use-parent-layout="false"/>
      </constraints>
      <properties>
        <text value="Make process net as compact as possible"/>
      </properties>
    </component>
    <component id="7aaec" class="javax.swing.JProgressBar" binding="progressBar">
      <constraints>
        <grid row="4" column="0" row-span="1" col-span="1" vsize-policy="0" hsize-policy="6" anchor="0" fill="1" indent="0" use-parent-layout="false"/>
      </constraints>
      <properties/>
    </component>
  </children>
 </grid>
</form>
```

B.2.2 ProcessGenerator.java

```
package pipe.gui.process;

import org.apache.commons.collections.CollectionUtils;
import org.apache.commons.lang.StringUtils;

import org.rendersnake.*;
import static org.rendersnake.HtmlAttributesFactory.*;

import uk.ac.imperial.pipe.exceptions.InvalidRateException;
import uk.ac.imperial.pipe.models.petrinet.*;
import uk.ac.imperial.pipe.animation.PetriNetAnimator;
import uk.ac.imperial.pipe.exceptions.PetriNetComponentNotFoundException;
import uk.ac.imperial.pipe.exceptions.PetriNetComponentException;
import uk.ac.imperial.pipe.naming.ComponentNamer;
import uk.ac.imperial.pipe.models.petrinet.name.*;
import uk.ac.imperial.utils.Pair;
import uk.ac.imperial.state.ClassifiedState;
import uk.ac.imperial.state.Record;
import uk.ac.imperial.pipe.visitor.ClonePetriNet;

import pipe.gui.widget.StateSpaceLoader;
import pipe.gui.widget.StateSpaceLoaderException;
import pipe.reachability.algorithm.*;

import java.io.File;
import javax.swing.*;
import javax.swing.event.*;

import java.awt.GridBagConstraints;
import java.awt.GridBagLayout;
import java.awt.Component;
import java.awt.FileDialog;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.FocusListener;
import java.awt.event.FocusEvent;
import java.io.IOException;
import java.net.URL;
import java.util.*;
import java.util.concurrent.ExecutionException;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.logging.Level;
import java.beans.PropertyChangeEvent;
import java.beans.PropertyChangeListener;
import java.beans.PropertyChangeSupport;
import java.lang.Math;

import pipe.gui.ModuleBridge;

/**
 * Performs the exploration and steady state analysis of a Petri net.
 * Displays useful performance analysis metrics
 */
public class ProcessGenerator {
    private JPanel mainPanel;

    private JScrollPane infoScrollPane;

    private JTextPane infoTextPane;

    private JTextField fireSequenceTextField;

    private JButton generateButton;

    private JCheckBox condenseCheckBox;

    private JProgressBar progressBar;
```

```
private final PetriNet net;

private PetriNet animatedNet;

private final PropertyChangeSupport changeSupport;

private PetriNetAnimator animator;

private String error;

private final int PROCESS_NET_LEFT_PADDING = 100;
private final int PROCESS_NET_TOP_PADDING = 200;
private final int PROCESS_NET_COLUMN_DISTANCE = 75;
private final int PROCESS_NET_ROW_DISTANCE = 100;
private final int PROCESS_NET_ANNOTATION_MIN_WIDTH = 600;

private final String NOT_ENABLED = "TRANSITION_NOT_ENABLED";
private final String NOT_EXIST = "TRANSITION_DOES_NOT_EXIST";

private final String PROTOCOL_REMOVE = "transition-remove";
private final String PROTOCOL_SELECT = "transition-select";
private final String PROTOCOL_ADD = "transition-add";

/**
 * We can only generate guaranteed correct process nets for nets with places that have capacity 1
 * @return
 */
private boolean isValidNet() {
    StateSpaceLoader stateSpaceLoader = new StateSpaceLoader(net, null);
    Map<Transition,Pair<Set<String>,Set<String>>> transitionMap = new HashMap<>();
    StateSpaceLoader.Results stateSpace = null;

    for(Place place : this.net.getPlaces()) {
            if(place.getCapacity() > 1 || !place.hasCapacityRestriction()) {
            this.error = "Process_net_generator_only_works_on_EN_systems";
                    return false;
            }
    }

    try {
            StateSpaceExplorer.StateSpaceExplorerResults results =
                    stateSpaceLoader.calculateResults(new StateSpaceLoader.ExplorerCreator() {
                                                          @Override
                                                          public ExplorerUtilities create(PetriNet petriNet) {
                                                              return new BoundedExplorerUtilities(petriNet, 1000);
                                                          }
                                                      }, new StateSpaceLoader.VanishingExplorerCreator() {
                                                          @Override
                                                          public VanishingExplorer create(ExplorerUtilities utils) {
                                                              return new SimpleVanishingExplorer();
                                                          }
                                                      }, 1
                    );
            stateSpace = stateSpaceLoader.loadStateSpace();
    } catch (InvalidRateException | TimelessTrapException | IOException | InterruptedException | ExecutionException | StateSpaceLoaderException e) {
        this.error = "Unable_to_generate_state_space_for_contact-detection";
    }

    for(Transition transition : net.getTransitions()) {
            Set<String> inboundPlaces = new HashSet<>();
            Set<String> outboundPlaces = new HashSet<>();

            for(InboundArc arc : net.inboundArcs(transition)) {
                    inboundPlaces.add(arc.getSource().getId());
            }
            for(OutboundArc arc : net.outboundArcs(transition)) {
                    outboundPlaces.add(arc.getTarget().getId());
            }
```

```
            transitionMap.put(transition, new Pair<>(inboundPlaces, outboundPlaces));
        }

        String tokenName = null;

        for(ClassifiedState state : stateSpace.stateMappings.values()) {
            Map<String, Map<String, Integer>> tokenMap = state.asMap();
            Set<String> configuration = new HashSet<>();

            for(Map.Entry<String, Map<String, Integer>> entry : tokenMap.entrySet()) {
                if(tokenName == null) {
                    for(String token : entry.getValue().keySet()) {
                        tokenName = token;
                        break;
                    }
                }

                if(entry.getValue().get(tokenName) == 1) {
                    configuration.add(entry.getKey());
                }
            }

            for(Map.Entry<Transition, Pair<Set<String>,Set<String>>> entry : transitionMap.entrySet()) {
                if(configuration.containsAll(entry.getValue().getLeft())) {
                    for(String outboundPlace : entry.getValue().getRight()) {
                        if(configuration.contains(outboundPlace)) {
                            this.error = "Transition " + entry.getKey().getId() + " has contact in configuration (" +
                                    StringUtils.join(configuration, ", ") + ")";
                            return false;
                        }
                    }
                }
            }
        }

        return true;
    }

    /**
     * Sets up the UI
     */
    private void setUp() {
        final PropertyChangeListener childListener = new PropertyChangeListener() {
            @Override
            public void propertyChange(PropertyChangeEvent e) {
                String innerProperty = e.getPropertyName();
                System.out.println("place: " + e.getPropertyName());
                if(innerProperty.equals(PlaceablePetriNetComponent.X_CHANGE_MESSAGE) ||
                        innerProperty.equals(PlaceablePetriNetComponent.Y_CHANGE_MESSAGE) ||
                        innerProperty.equals(PlaceablePetriNetComponent.WIDTH_CHANGE_MESSAGE) ||
                        innerProperty.equals(PlaceablePetriNetComponent.HEIGHT_CHANGE_MESSAGE)) {
                    //We do not need to change anything
                    return;
                }
                setUpDialog();
            }
            };

        setUpDialog();
        progressBar.setVisible(false);

        this.net.addPropertyChangeListener(new PropertyChangeListener() {
            @Override
            public void propertyChange(PropertyChangeEvent e) {
                String property = e.getPropertyName();
                if(property.equals(PetriNet.NEW_PLACE_CHANGE_MESSAGE)) {
                    ((Place) e.getNewValue()).addPropertyChangeListener(childListener);
                } else if(property.equals(PetriNet.NEW_TRANSITION_CHANGE_MESSAGE)) {
                    ((Transition) e.getNewValue()).addPropertyChangeListener(childListener);
                }
```

```
                System.out.println("main: " + property);

                setUpDialog();
        }
});

for(Place place : this.net.getPlaces()) {
        place.addPropertyChangeListener(childListener);
}
for(Transition transition : this.net.getTransitions()) {
        transition.addPropertyChangeListener(childListener);
}


//Called live during editing
fireSequenceTextField.getDocument().addDocumentListener(new DocumentListener() {
          public void changedUpdate(DocumentEvent e) {
            updateState();
          }
          public void removeUpdate(DocumentEvent e) {
            updateState();
          }
          public void insertUpdate(DocumentEvent e) {
            updateState();
          }
        });

//Called after losing focus
fireSequenceTextField.addFocusListener(new FocusListener() {
        @Override
        public void focusGained(FocusEvent e) {
                //Do nothing
        }

    @Override
    public void focusLost(FocusEvent e) {
        updateState();
    }
});

generateButton.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {
                generateFormattedPetriNet(getFireSequence(), condenseCheckBox.isSelected());
        }
});

infoTextPane.addHyperlinkListener(new HyperlinkListener() {
        public void hyperlinkUpdate(HyperlinkEvent e) {
        if(e.getEventType() == HyperlinkEvent.EventType.ACTIVATED) {
                List<String> fireSequence = getFireSequence();
                String[] action = e.getDescription().split("::");

                if(action[0].equals(PROTOCOL_REMOVE)) {
                        fireSequence.remove(Integer.parseInt(action[1]));
                        setFireSequence(fireSequence);
                } else if(action[0].equals(PROTOCOL_ADD)) {
                        addToFireSequence(action[1]);
                } else if(action[0].equals(PROTOCOL_SELECT)) {
                        String inputString = fireSequenceTextField.getText();
                        int start;
                        int entryIndex = Integer.parseInt(action[1]);
                        if(entryIndex == 0) {
                                //There can be whitespace before the entry
                                start = inputString.indexOf(fireSequence.get(entryIndex));
                        } else {
                                int commastart = StringUtils.ordinalIndexOf(inputString, ",", entryIndex);
                                start = inputString.indexOf(fireSequence.get(entryIndex), commastart);
                        }
                        int end = start + fireSequence.get(entryIndex).length();
```

```
                        fireSequenceTextField.requestFocusInWindow();
                        fireSequenceTextField.setCaretPosition(start);
                        fireSequenceTextField.moveCaretPosition(end);
                    } else {
                        System.out.println(action[o] + " is unknown!");
                    }

            }
        }
    });
}

private void setUpAnimator() {
    this.animatedNet = ClonePetriNet.clone(this.net);
    if(this.animatedNet == null) {
            JOptionPane.showMessageDialog(getMainPanel(), "Could not clone current Petri net: Petri net might be inconsistent.", "Error", JOptionPane.
                OK_OPTION);
            getMainPanel().getRootPane().setVisible(false);
    }
    this.animator = new PetriNetAnimator(this.animatedNet);
}

private void setUpDialog() {
    if(isValidNet()) {
            this.error = null;

                    //We can use a fire sequence on this net
                    fireSequenceTextField.setEnabled(true);

                    //The net used for animating might be different
                    setUpAnimator();

                    //Since the net was changed, this information might be out of date
                    updateState();
            } else {
                    fireSequenceTextField.setEnabled(false);
            generateButton.setEnabled(false);
            generateButton.setToolTipText(this.error);
            updateInfo();
            }
}

private void addToFireSequence(String transition) {
    List<String> sequence = getFireSequence();
    sequence.add(transition);
    fireSequenceTextField.setText(StringUtils.join(sequence, ", "));
}

private void updateState() {
    ProcessResult result = recalculateAnimations(getFireSequence());

    if (result.errors.size() == o) {
                    generateButton.setEnabled(true);
                    generateButton.setToolTipText("Generate a process net with given fire sequence");
                    this.error = null;
            } else {
                    generateButton.setEnabled(false);
                    generateButton.setToolTipText("Given fire sequence is invalid");
                    this.error = "Given fire sequence is invalid.";
    }

            updateInfo(result);
}

/**
 * Processes input string into names of transitions
 * @return tokenized string with names of fired transitions
 */
private List<String> getFireSequence() {
```

```
    String unparsedFireSequence = fireSequenceTextField.getText();
    String[] partlyParsedSequence = unparsedFireSequence.split("[\\s\\u0085\\p{Z}]*,[\\s\\u0085\\p{Z}]*");
    List<String> parsedSequence = new ArrayList<>();

    for( String transition : partlyParsedSequence ) {
        String parsedTransition = transition.trim();
        if(!parsedTransition.equals("")) {
            parsedSequence.add(parsedTransition);
        }
    }
    return parsedSequence;
}

private void setFireSequence(List<String> fireSequence) {
    fireSequenceTextField.setText(StringUtils.join(fireSequence, ", "));
}

/**
 * Simulates the current fire sequence, detects impossible situations and calculates transitions that can be fired
 * @return
 */
private ProcessResult recalculateAnimations(List<String> fireSequence) {
    animator.saveState();
    Set<String> availableTransitions = new HashSet<>();
    ArrayList<Pair<Integer,String>> errors = new ArrayList<>();
    int lastElement = fireSequence.size() - 1;

    for(int i = 0; i < fireSequence.size(); i++) {
        String firedTransition = fireSequence.get(i);
        if(!firedTransition.equals("")) {
            try {
                Transition transition = animatedNet.getComponent(firedTransition, Transition.class);

                if(!animator.getEnabledTransitions().contains(transition)) {
                    errors.add(new Pair<>(i, NOT_ENABLED));
                } else {
                    animator.fireTransition(transition);
                }
            } catch(PetriNetComponentNotFoundException e) {
                if(!(fireSequenceTextField.hasFocus() && i == lastElement)) {
                    errors.add(new Pair<>(i, NOT_EXIST));
                }
            }
        }
    }
        for(Transition transition : animator.getEnabledTransitions()) {
            availableTransitions.add(transition.getId());
        }

    animator.reset();
    return new ProcessResult(availableTransitions, errors);
}

private void updateInfo(ProcessResult result) {
    HtmlCanvas html = new HtmlCanvas();
    List<String> fireSequence = getFireSequence();

    try {
        html.html().head()._head();
        html.body();

        if(result.errors.size() > 0) {
            html.b().content("The following issues currently exist");
            html.ul();
            for(Pair<Integer,String> error : result.errors) {
                html.li().write(fireSequence.get(error.getLeft()) + ": ");
                if(error.getRight().equals(NOT_ENABLED)) {
                    html.write("Transition " + (error.getLeft()+1) + " is not enabled. ");
                } else if(error.getRight().equals(NOT_EXIST)) {
                    html.write("No transition with such a name exists. ");
```

```
                        } else {
                                html.write("UKNOWN_ERROR");
                        }
                        html.write("(");
                        html.a(href(PROTOCOL_REMOVE + "::" + error.getLeft())).content("Remove").write("_|_");
                        html.a(href(PROTOCOL_SELECT + "::" + error.getLeft())).content("Select");
                        html.write(")");
                        html._li();
                }
                html._ul();
        }
        html.b().content("The_following_transitions_can_be_fired");
        html.ul();
        for(String availableTransition : result.availableTransitions) {
                html.li().write(availableTransition);
                html.write("_(");
                html.a(href(PROTOCOL_ADD + "::" + availableTransition)).content("Add");
                html.write(")");
                html._li();
        }
        html._ul();

        if(this.error != null) {
                html.b().content("Cannot_generate_process_net");
                html.ul();
                html.li().write(this.error);
                html._li();
                html._ul();
        }

        html._body()._html();
      infoTextPane.setText(html.toHtml());
    } catch(IOException e) {
        infoTextPane.setText("<html><head></head><body>Error</body></html>");
    }
}

private void updateInfo() {
    HtmlCanvas html = new HtmlCanvas();

    try {
        html.html().head()._head().body();
            if(this.error != null) {
                html.b().content("Cannot_generate_process_net");
                html.ul();
                html.li().write(this.error);
                html._li();
                html._ul();
            }
            html._body()._html();
            infoTextPane.setText(html.toHtml());
    } catch(IOException e) {
        infoTextPane.setText("<html><head></head><body>Error</body></html>");
      }
}

private void generateFormattedPetriNet(final List<String> fireSequence, final boolean condensed) {
    final PetriNet processNet = new PetriNet();
    final PetriNet localAnimatedNet = ClonePetriNet.clone(this.net);

    //Set up the basics of this net
    PetriNetName name = new NormalPetriNetName(localAnimatedNet.getNameValue() + "_process_(" + StringUtils.join(fireSequence, ",_") + ")");
    processNet.setName(name);

    Thread thread = new Thread() {
        @Override
        public void run() {
            generateButton.setEnabled(false);
            progressBar.setVisible(true);
            progressBar.setValue(0);
```

```
Map<Place, Place> processPlaceMap = new HashMap<>();
List<List<PlaceablePetriNetComponent>> columns = new ArrayList<>();
Map<Place, PlaceNamer> placeNamers = new HashMap<>();
Map<Transition, TransitionNamer> transitionNamers = new HashMap<>();

Comparator<PlaceablePetriNetComponent> comparator = new Comparator<PlaceablePetriNetComponent>() {
        @Override
            public int compare(PlaceablePetriNetComponent component1, PlaceablePetriNetComponent component2) {
                if(component2.getX() - component1.getX() != 0) {
                        //Left-most component should be after right-most element
                        return component2.getX() - component1.getX();
                } if(component1.getY() - component2.getY() != 0) {
                        //Top-most component must be before components below it
                        return component1.getY() - component2.getY();
                } else {
                        return component1.getId().compareTo(component2.getId());
                }
            }
        };

        progressBar.setValue(1);

for(Token token : localAnimatedNet.getTokens()) {
        processNet.addToken(token);
}

for(int i = 0; i < (fireSequence.size()+1)*2; i++) {
        columns.add(new ArrayList<PlaceablePetriNetComponent>());
}

try {
        progressBar.setValue(10);
        //Starting configuration and initialisation
        for(Place place : localAnimatedNet.getPlaces()) {
                Place processPlace = null;
                        placeNamers.put(place, new PlaceNamer(processNet, localAnimatedNet, place.getId()));

                if(place.getNumberOfTokensStored() > 0) {
                        processPlace = new DiscretePlace(placeNamers.get(place).getName());
                        processPlace.setTokenCounts(place.getTokenCounts());
                        processPlace.setX(PROCESS_NET_LEFT_PADDING);
                        //Y coordinate is further refined later
                        processPlace.setY(PROCESS_NET_TOP_PADDING);
                        columns.get(0).add(processPlace);
                        processNet.add(processPlace);
                }
                        processPlaceMap.put(place, processPlace);
        }
        for(Transition transition : localAnimatedNet.getTransitions()) {
                transitionNamers.put(transition, new TransitionNamer(processNet, localAnimatedNet, transition.getId()));
        }

                progressBar.setValue(30);
        //Process transitions in fire sequence
        for(int i = 0; i < fireSequence.size(); i++) {
                String transitionName = fireSequence.get(i);
                if(transitionName.equals("")) {
                        continue;
                }

                Transition transition = localAnimatedNet.getComponent(transitionName, Transition.class);
                Transition processTransition = new DiscreteTransition(transitionNamers.get(transition).getName());
                Set<Place> inboundPlaces = new HashSet<Place>();
                Set<Place> outboundPlaces = new HashSet<Place>();
                int transitionColumn = 0;

                processNet.add(processTransition);

                for(InboundArc inboundArc : localAnimatedNet.inboundArcs(transition)) {
```

```
                Place inboundPlace = inboundArc.getSource();
                Place processPlace = processPlaceMap.get(inboundPlace);
                if(processPlace == null) {
                        JOptionPane.showMessageDialog(getMainPanel(), "Transition " + transitionName + " fired, but it's inbound place "
                                + inboundPlace.getId() + " was not yet in the process net.", "Error", JOptionPane.OK_OPTION);
                        return;
                }
                processPlaceMap.put(inboundPlace, null);
                int placeColumn = (processPlace.getX() - PROCESS_NET_LEFT_PADDING) / PROCESS_NET_COLUMN_DISTANCE;

                if(transitionColumn <= placeColumn) {
                        transitionColumn = placeColumn + 1;
                }

                InboundArc processInboundArc = new InboundNormalArc(processPlace, processTransition, inboundArc.getTokenWeights());

                inboundPlaces.add(processPlace);

                processNet.add(processPlace);
                processNet.add(processInboundArc);
        }

        if(!condensed) {
                //If not in condensed format, each transition is in it's own column
                transitionColumn = 1 + i*2;
        }
        processTransition.setX(PROCESS_NET_LEFT_PADDING + transitionColumn * PROCESS_NET_COLUMN_DISTANCE);

        for(OutboundArc outboundArc : localAnimatedNet.outboundArcs(transition)) {
                Place outboundPlace = outboundArc.getTarget();
                if(processPlaceMap.get(outboundPlace) != null) {
                        JOptionPane.showMessageDialog(getMainPanel(), "Transition " + transitionName + " fired, but it's outbound place "
                                + outboundPlace.getId() + " was already in the process net.", "Error", JOptionPane.OK_OPTION);
                        return;
                }
                Place processPlace = new DiscretePlace(placeNamers.get(outboundPlace).getName());
                processPlace.setX(PROCESS_NET_LEFT_PADDING + (transitionColumn + 1) * PROCESS_NET_COLUMN_DISTANCE);
                processPlaceMap.put(outboundPlace, processPlace);
                OutboundArc processOutboundArc = new OutboundNormalArc(processTransition, processPlace, outboundArc.getTokenWeights());

                outboundPlaces.add(processPlace);

                processNet.add(processPlace);
                processNet.add(processOutboundArc);
        }

        columns.get(transitionColumn).add(processTransition);
        columns.get(transitionColumn + 1).addAll(outboundPlaces);
}

        progressBar.setValue(50);
//Initial height
for(int i = 0; i < columns.size(); i++) {
        List<PlaceablePetriNetComponent> column = columns.get(i);
        for(PlaceablePetriNetComponent component : column) {
                int y = component.getY();
                Collection<Arc> arcs = new HashSet<Arc>();
                if((i % 2) == 0 ) {
                        arcs.addAll(processNet.inboundArcs((Place) component));
                } else {
                        arcs.addAll(processNet.inboundArcs((Transition) component));
                }

                for(Arc arc : arcs) {
                        y += arc.getSource().getY();
                }
                if(arcs.size() > 0) {
                        y = y / arcs.size();
                }
                if(y < PROCESS_NET_TOP_PADDING) {
```

```
                        y = PROCESS_NET_TOP_PADDING;
                    }

                    component.setY(y);
                }

                Collections.sort(column, comparator);

                ensureRowDistance(column);
            }

                progressBar.setValue(60);
            //Fine-tuning of height based on transitions
            for(int i = 1; i < columns.size(); i += 2) {
                List<PlaceablePetriNetComponent> column = columns.get(i);
                for(PlaceablePetriNetComponent component : column) {
                    List<PlaceablePetriNetComponent> places = new ArrayList<>();
                    Collection<Arc> arcs = new HashSet<Arc>(processNet.inboundArcs((Transition) component));
                    for(Arc arc : arcs) {
                        places.add(arc.getSource());
                    }
                    Collections.sort(places, comparator);

                    int y = 0;
                    int num = 0;
                    y += ensureRowDistance(places);
                    num += places.size();

                    places.clear();
                    arcs = new HashSet<Arc>(processNet.outboundArcs((Transition) component));
                    for(Arc arc : arcs) {
                        places.add(arc.getTarget());
                    }
                    Collections.sort(places, comparator);

                    y += ensureRowDistance(places);
                    num += places.size();

                    if(num > 0) {
                        component.setY(y / num);
                    }
                }
            }

            for(List<PlaceablePetriNetComponent> column : columns) {
                ensureRowDistance(column);
            }

                progressBar.setValue(80);
            //Finishing touches
            Annotation annotation = new AnnotationImpl(100, 100, "Process net for " + StringUtils.join(fireSequence, ", "), Math.max(
                    PROCESS_NET_ANNOTATION_MIN_WIDTH, columns.size()*PROCESS_NET_ROW_DISTANCE), 30, false);
            processNet.add(annotation);

            progressBar.setValue(100);

            progressBar.setVisible(false);
            generateButton.setEnabled(true);
        } catch(PetriNetComponentNotFoundException e) {
            JOptionPane.showMessageDialog(getMainPanel(), "Could not find a Petri net component, even though we established somewhere else that it
                    should exist.", "Error", JOptionPane.OK_OPTION);
            return;
        } catch(PetriNetComponentException e) {
            JOptionPane.showMessageDialog(getMainPanel(), "Could not add a component to process net, even though we made sure they were unique...", "
                    Error", JOptionPane.OK_OPTION);
            return;
        }

            changeSupport.firePropertyChange(ModuleBridge.MODULE_ADD_PETRINET_MESSAGE, null, processNet);
        }
```

```
        };
        thread.start();
    }

    public ProcessGenerator(PetriNet petriNet, FileDialog fileDialog, PropertyChangeSupport changeSupport) {
        this.net = petriNet;
        this.changeSupport = changeSupport;

        setUp();
    }

    private int ensureRowDistance(Collection<PlaceablePetriNetComponent> components) {
        int y = 0;
                int lastY = PROCESS_NET_TOP_PADDING - PROCESS_NET_ROW_DISTANCE;
                for(PlaceablePetriNetComponent component : components) {
                        if(component.getY() - lastY < PROCESS_NET_ROW_DISTANCE) {
                                component.setY(lastY + PROCESS_NET_ROW_DISTANCE);
                        }
                        lastY = component.getY();
                        y += lastY;
                }

                return y;
    }

    /**
     * Main method for running this externally without PIPE
     *
     * @param args          command line arguments
     */
    public static void main(String[] args) {
        //Does not make sense to run this as a stand-alone program
    }

    public JPanel getMainPanel() {
        return mainPanel;
    }

    private final class ProcessResult {
        public final Set<String> availableTransitions;
        public final ArrayList<Pair<Integer,String>> errors;

        ProcessResult(Set<String> availableTransitions, ArrayList<Pair<Integer,String>> errors) {
                this.availableTransitions = availableTransitions;
                this.errors = errors;
        }
    }

    private class PlaceNamer extends ComponentNamer {
        PlaceNamer(PetriNet petriNet, PetriNet originalPetriNet, String placeName) {
                super(petriNet, placeName + "--", PetriNet.NEW_PLACE_CHANGE_MESSAGE, PetriNet.DELETE_PLACE_CHANGE_MESSAGE);
                for(Place place : originalPetriNet.getPlaces()) {
                names.add(place.getId());
                }
        }
    }

    private class TransitionNamer extends ComponentNamer {
        TransitionNamer(PetriNet petriNet, PetriNet originalPetriNet, String transitionName) {
                super(petriNet, transitionName + "--", PetriNet.NEW_TRANSITION_CHANGE_MESSAGE, PetriNet.DELETE_TRANSITION_CHANGE_MESSAGE);
                for(Transition transition : originalPetriNet.getTransitions()) {
                names.add(transition.getId());
                }
        }
    }
}
```

B.3 State space module

B.3.1 ReachabilityGraph.java

```
package pipe.gui.reachability;

import net.sourceforge.jpowergraph.Edge;
import net.sourceforge.jpowergraph.Node;
import net.sourceforge.jpowergraph.defaults.DefaultGraph;
import net.sourceforge.jpowergraph.layout.Layouter;
import net.sourceforge.jpowergraph.layout.spring.SpringLayoutStrategy;
import net.sourceforge.jpowergraph.lens.*;
import net.sourceforge.jpowergraph.manipulator.dragging.DraggingManipulator;
import net.sourceforge.jpowergraph.manipulator.popup.PopupManipulator;
import net.sourceforge.jpowergraph.swing.SwingJGraphPane;
import net.sourceforge.jpowergraph.swing.SwingJGraphScrollPane;
import net.sourceforge.jpowergraph.swing.manipulator.SwingPopupDisplayer;
import net.sourceforge.jpowergraph.swtswinginteraction.color.JPowerGraphColor;
import pipe.gui.widget.GenerateResultsForm;
import pipe.gui.widget.StateSpaceLoader;
import pipe.gui.widget.StateSpaceLoaderException;
import pipe.reachability.algorithm.*;
import uk.ac.imperial.pipe.exceptions.InvalidRateException;
import uk.ac.imperial.pipe.models.petrinet.PetriNet;
import uk.ac.imperial.state.ClassifiedState;
import uk.ac.imperial.state.Record;
import uk.ac.imperial.pipe.models.petrinet.Place;
import uk.ac.imperial.utils.Pair;

import org.apache.commons.lang.StringUtils;

import javax.swing.*;
import java.awt.Container;
import java.awt.FileDialog;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Collection;
import java.util.Collections;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.Set;
import java.util.HashSet;
import java.util.concurrent.ExecutionException;
import java.util.logging.Level;
import java.util.logging.Logger;

/**
 * GUI class used to display and run the results of reachability and coverability classes
 */
public class ReachabilityGraph {

    /**
     * Class logger
     */
    private static final Logger LOGGER = Logger.getLogger(ReachabilityGraph.class.getName());

    /**
     * Maximum number of states to graphically display
     */
    private static final int MAX_STATES_TO_DISPLAY = 100;


    private JPanel panel1;
```

```
/**
 * Contains the graph based results of state space exploration
 */
private JPanel resultsPanel;

/**
 * Check box to determine if we include vanishing states in the exploration
 */
private JCheckBox includeVanishingStatesCheckBox;

/**
 * For saving state space results
 */
private JButton saveButton;

private JLabel textResultsLabel;
private JPanel textResultsPanel;


private JRadioButton reachabilityButton;

private JRadioButton coverabilityButton;

private JTextField maxStatesField;

private JPanel stateLoadingPanel;

private JPanel generatePanel;

private DefaultGraph graph = new DefaultGraph();

private StateSpaceLoader stateSpaceLoader;

/**
 * Asks user to select a petrinet. "use current Petri net" can be used to use current petrinet
 *
 * @param loadDialog the dialog to be shown
 * @param petriNet   current petri net
 */
public ReachabilityGraph(FileDialog loadDialog, PetriNet petriNet) {
    stateSpaceLoader = new StateSpaceLoader(petriNet, loadDialog);
    setUp();
}

/**
 * Calculates the maximum capacity of all places in this petriNet
 *
 * @return int 0 if infinite, otherwise maximum capacity
 */
private int getMaxCapacity() {
    int maxCapacity = 1;
    PetriNet petriNet = stateSpaceLoader.getPetriNet();

    for(Place place : petriNet.getPlaces()) {
            if(!place.hasCapacityRestriction()) {
                    //There is nothing larger than infinity
                    return 0;
            } else if(place.getCapacity() > maxCapacity) {
                    maxCapacity = place.getCapacity();
            }
    }
    return maxCapacity;
}

/**
 *
 * @return return true if all places have infinite capacity, and false if at least 1 has a limited capacity
 */
private boolean hasOnlyInfiniteCapacity() {
    PetriNet petriNet = stateSpaceLoader.getPetriNet();
```

```
        for(Place place : petriNet.getPlaces()) {
                if(place.hasCapacityRestriction()) {
                        return false;
                }
        }
        return true;
    }

    /**
     * Set up action listeners
     */
    private void setUp() {
        JPanel pane = setupGraph();
        resultsPanel.add(pane);
        stateLoadingPanel.add(stateSpaceLoader.getMainPanel(), 0);

        ActionListener disableListener = new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                reachabilityButton.setEnabled(false);
                coverabilityButton.setEnabled(false);
                includeVanishingStatesCheckBox.setEnabled(false);
            }
        };

        ActionListener enableListener = new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                reachabilityButton.setEnabled(true);
                coverabilityButton.setEnabled(true);
                includeVanishingStatesCheckBox.setEnabled(true);
            }
        };
//
        stateSpaceLoader.addPetriNetRadioListener(enableListener);
        stateSpaceLoader.addBinariesListener(disableListener);
        saveButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                saveBinaryFiles();
            }
        });
        GenerateResultsForm resultsForm = new GenerateResultsForm(new GenerateResultsForm.GoAction() {
            @Override
            public void go(int threads) {
                calculateResults(threads);
            }
        });
        generatePanel.add(resultsForm.getPanel());
    }

    /**
     * Sets up the graph and returns the JPanel to add to
     * the resultsPanel
     */
    private JPanel setupGraph() {
        SwingJGraphPane pane = new SwingJGraphPane(graph);
        LensSet lensSet = new LensSet();
        lensSet.addLens(new RotateLens());
        lensSet.addLens(new TranslateLens());
        lensSet.addLens(new ZoomLens());
        CursorLens draggingLens = new CursorLens();
        lensSet.addLens(draggingLens);
        lensSet.addLens(new TooltipLens());
        lensSet.addLens(new LegendLens());
        lensSet.addLens(new NodeSizeLens());
        pane.setLens(lensSet);

        pane.addManipulator(new DraggingManipulator(draggingLens, -1));
```

```
        pane.addManipulator(new PopupManipulator(pane, (TooltipLens) lensSet.getFirstLensOfType(TooltipLens.class)));


        pane.setNodePainter(TangibleStateNode.class, TangibleStateNode.getShapeNodePainter());
        pane.setNodePainter(VanishingStateNode.class, VanishingStateNode.getShapeNodePainter());
        pane.setNodePainter(TangibleStartStateNode.class, TangibleStartStateNode.getShapeNodePainter());
        pane.setNodePainter(VanishingStartStateNode.class, VanishingStartStateNode.getShapeNodePainter());


        pane.setEdgePainter(DirectedTextEdge.class,
                new PIPELineWithTextEdgePainter(JPowerGraphColor.BLACK, JPowerGraphColor.GRAY, false));
        pane.setEdgePainter(SpacerEdge.class,
                        new PIPESpacerEdgePainter(JPowerGraphColor.BLACK, JPowerGraphColor.GRAY, false));

        pane.setAntialias(true);

        pane.setPopupDisplayer(new SwingPopupDisplayer(new PIPESwingToolTipListener(),
                new PIPESwingContextMenuListener(graph, new LensSet(), new Integer[]{}, new Integer[]{})));

        return new SwingJGraphScrollPane(pane, lensSet);
    }

    /**
     * Calculates the steady state exploration of a Petri net and stores its results
     * in a temporary file.
     *
     * These results are then read in and turned into a graphical representation using mxGraph
     * which is displayed to the user
     * @param threads number of threads to use to explore the state space
     */
    private void calculateResults(int threads) {
        if(coverabilityButton.isSelected() && !hasOnlyInfiniteCapacity()) {
                JOptionPane.showMessageDialog(getMainPanel(),
                        "The coverability graph algorithm only works for Petri nets where every place has infinite capacity. Please complement all places
                                that have a capacity and set their capacity to infinite, then try again.",
                        "All places must have infinite capacity",
                        JOptionPane.ERROR_MESSAGE);
                return;
        }

        try {
            StateSpaceExplorer.StateSpaceExplorerResults results =
                    stateSpaceLoader.calculateResults(new StateSpaceLoader.ExplorerCreator() {
                                                          @Override
                                                          public ExplorerUtilities create(PetriNet petriNet) {
                                                              return getExplorerUtilities(petriNet);
                                                          }
                                                      }, new StateSpaceLoader.VanishingExplorerCreator() {
                                                          @Override
                                                          public VanishingExplorer create(ExplorerUtilities utils) {
                                                              return getVanishingExplorer(utils);
                                                          }
                                                      }, threads
                    );
            updateTextResults(results.numberOfStates, results.processedTransitions);
            if (results.numberOfStates <= MAX_STATES_TO_DISPLAY) {
                StateSpaceLoader.Results stateSpace = stateSpaceLoader.loadStateSpace();
                updateGraph(stateSpace.records, stateSpace.stateMappings);
            }

        } catch (InvalidRateException | TimelessTrapException | IOException | InterruptedException | ExecutionException e) {
            LOGGER.log(Level.SEVERE, e.toString());
            JOptionPane.showMessageDialog(panel1, e.toString(), "State space explorer error", JOptionPane.ERROR_MESSAGE);
        } catch (StateSpaceLoaderException e) {
            JOptionPane.showMessageDialog(panel1, e.getMessage(), "GSPN Analysis Error", JOptionPane.ERROR_MESSAGE);
        }
    }

    /**
     * Copies the temporary files to a permanent loaction
```

```
     */
    private void saveBinaryFiles() {
        stateSpaceLoader.saveBinaryFiles();
    }

    /**
     * Creates the explorer utilities based upon whether the coverability or reachability graph
     * is being generate
     *
     * @param petriNet  petrinet
     * @return explorer utilities for generating state space
     */
    private ExplorerUtilities getExplorerUtilities(PetriNet petriNet) {
        if (coverabilityButton.isSelected()) {
            return new CoverabilityExplorerUtilities(new UnboundedExplorerUtilities(petriNet));
        }

        return new BoundedExplorerUtilities(petriNet, Integer.valueOf(maxStatesField.getText()));

    }

    /**
     * Vanishing explorer is either a {@link pipe.reachability.algorithm.SimpleVanishingExplorer} if
     * vanishing states are to be included in the graph, else it is {@link pipe.reachability.algorithm.OnTheFlyVanishingExplorer}
     *
     * @param explorerUtilities  previously generated explorer utilities
     */
    private VanishingExplorer getVanishingExplorer(ExplorerUtilities explorerUtilities) {
        if (includeVanishingStatesCheckBox.isSelected()) {
            return new SimpleVanishingExplorer();
        }
        return new OnTheFlyVanishingExplorer(explorerUtilities);
    }

    /**
     * Updates the text results with the number of states and transitions
     *
     * @param states       number of states
     * @param transitions number of transitions
     */
    private void updateTextResults(int states, int transitions) {
        StringBuilder results = new StringBuilder();
        results.append("Results: ").append(states).append(" states and ").append(transitions).append(" transitions");
        textResultsLabel.setText(results.toString());
    }

    /**
     * Updates the mxGraph to display the records
     *
     * @param records  state transitions from a processed Petri net
     * @param stateMap state map
     */
    private void updateGraph(Iterable<Record> records, Map<Integer, ClassifiedState> stateMap) {
        graph.clear();
        Map<Integer, Node> nodes = getNodes(stateMap);
        Collection<Edge> edges = getEdges(records, nodes);
        graph.addElements(nodes.values(), edges);
        layoutGraph();
    }

    /**
     * @param stateMap state map
     * @return All nodes to be added to the graph
     */
    private Map<Integer, Node> getNodes(Map<Integer, ClassifiedState> stateMap) {
        int maxCapacity = getMaxCapacity();

        Map<Integer, Node> nodes = new HashMap<>(stateMap.size());
        for (Map.Entry<Integer, ClassifiedState> entry : stateMap.entrySet()) {
            ClassifiedState state = entry.getValue();
```

```
            int id = entry.getKey();

            if(maxCapacity == 1) {
                nodes.put(id, createSimpleNode(state, id));
            } else {
                nodes.put(id, createRegularNode(state, id));
            }
        }
        return nodes;
    }

    /**
     * All edges to be added to the graph
     *
     * @param records  records of states, and all states that can be reached from each state
     * @param nodes    map of ids to the corresponding state nodes
     * @return         all directional edges between state nodes A and B, where B can be reached from A
     */
    private Collection<Edge> getEdges(Iterable<Record> records, Map<Integer, Node> nodes) {
        Collection<Edge> edges = new ArrayList<>();
        Map<Node,Set<Node>> connections = new HashMap<>();

        for (Record record : records) {
            int state = record.state;
            for (Map.Entry<Integer, Pair<Double, Collection<String>>> entry : record.successors.entrySet()) {
                int succ = entry.getKey();
                ArrayList<String> transitionNames = (ArrayList<String>) entry.getValue().getRight();
                Collections.sort(transitionNames);
                double rate = entry.getValue().getLeft();

                Node startNode = nodes.get(state);
                Node endNode = nodes.get(succ);
                edges.add(new DirectedTextEdge(startNode, endNode,
                        String.format("%s (%.2f)", StringUtils.join(transitionNames, ", "), rate)));

                //Keep track of all single connections
                addToSet(startNode, endNode, connections);
                addToSet(endNode, startNode, connections);
            }
        }

        return edges;
    }

    private void addToSet(Node key, Node val, Map<Node,Set<Node>> collection) {
        Set<Node> result;
        if(!collection.containsKey(key)) {
                result = new HashSet<>(1);
                result.add(val);
                collection.put(key, result);
        } else {
                result = collection.get(key);
                result.add(val);
        }
    }

    private boolean inSet(Node key, Node val, Map<Node,Set<Node>> collection) {
        return collection.containsKey(key) && collection.get(key).contains(val);
    }

    /**
     * Performs laying out of items on the graph
     */
    private void layoutGraph() {
        Layouter layouter = new Layouter(new SpringLayoutStrategy(graph));
        layouter.start();
    }

    /**
     * Creates a node that displays which places contain a token. Only to be used if maxCapacity == 1
```

```
     *
     * @param state classified state to be turned into a graph node
     * @param id    state integer id
     * @return Tangible or Vanishing state node with simple notation
     */
    private Node createSimpleNode(ClassifiedState state, int id) {
        String label = "";
        String toolTip = "";

        List<String> places = new ArrayList<String>();
        for(String place : state.getPlaces()) {
                places.add(place);
        }
        Map<String, Map<String, Integer>> tokenMap = state.asMap();
        Collections.sort(places);
                int numberOfTokens = tokenMap.get(places.get(0)).size();

                if(numberOfTokens > 1) {
                        //Well, that was a waste of time...
                        return createRegularNode(state, id);
                }

                String token = "";
                for(String onlyToken : tokenMap.get(places.get(0)).keySet()) {
                        token = onlyToken;
                }

        List<String> preparedLabel = new ArrayList<String>();
        List<String> preparedToolTip = new ArrayList<String>();

        for(String place : places) {
                int tokenCount = tokenMap.get(place).get(token);
                        if( tokenCount == 1 ) {
                                preparedLabel.add(place);
                        }
                        preparedToolTip.add("<b>" + place + ":</b> " + tokenCount);
        }

        label = StringUtils.join(preparedLabel, ", ");
        toolTip = StringUtils.join(preparedToolTip, "<br>");
        return createNode(state, label, toolTip, id);
    }

    /**
     * Creates a node that displays the number of tokens per type, per place for this state in a tuple
     *
     * @param state classified state to be turned into a graph node
     * @param id    state integer id
     * @return Tangible or Vanishing state node with tuple notation
     */
    private Node createRegularNode(ClassifiedState state, int id) {
        String label = "";
        String toolTip = "";

        List<String> places = new ArrayList<String>();
        for(String place : state.getPlaces()) {
                places.add(place);
        }
        List<String> tokens = new ArrayList<String>();
        Map<String, Map<String, Integer>> tokenMap = state.asMap();
        Collections.sort(places);
                int numberOfTokens = tokenMap.get(places.get(0)).size();
        for(String token : tokenMap.get(places.get(0)).keySet()) {
                tokens.add(token);
        }
        Collections.sort(tokens);
        List<String> preparedStrings = new ArrayList<String>();
        List<String> preparedToolTipStrings = new ArrayList<String>();

        label += "[" + Integer.toString(id) + "] ";
```

```
        for(String place : places) {
                List<String> tokenCountForPlace = new ArrayList<String>();
                List<String> toolTipTokenCountForPlace = new ArrayList<String>();
                String preparedString;

                for(String token : tokens) {
                        int tokenCount = tokenMap.get(place).get(token);
                        if(tokenCount == Integer.MAX_VALUE) {
                                tokenCountForPlace.add("  ");
                                toolTipTokenCountForPlace.add("  " + token);
                        } else {
                                tokenCountForPlace.add(Integer.toString(tokenCount));
                                toolTipTokenCountForPlace.add(Integer.toString(tokenCount) + " " + token);
                        }
                }
                preparedString = StringUtils.join(tokenCountForPlace, ",");

                if(numberOfTokens > 1) {
                        preparedStrings.add("(" + preparedString + ")");
                } else {
                        preparedStrings.add(preparedString);
                }

                preparedToolTipStrings.add("<b>" + place + ":</b> " + StringUtils.join(toolTipTokenCountForPlace, ", "));
        }
        label += "(";
        label += StringUtils.join(preparedStrings, ",");
        label += ")";

        toolTip = StringUtils.join(preparedToolTipStrings, "<br>");

        return createNode(state, label, toolTip, id);
    }

    /**
     * @param state classified state to be turned into a graph node
     * @param String label to be used on the node
     * @param String tooltip to be used when hovering over node with mouse
     * @param id    state integer id
     * @return Tangible or Vanishing state node
     */
    private Node createNode(ClassifiedState state, String label, String toolTip, int id) {
        if (state.isTangible() && id == 0) {
                return new TangibleStartStateNode(label, toolTip, id);
        } else if(state.isTangible()) {
            return new TangibleStateNode(label, toolTip, id);
        } else if(id == 0) {
                return new VanishingStartStateNode(label, toolTip, id);
        } else {
                return new VanishingStateNode(label, toolTip, id);
        }
    }

    /**
     * Constructor deactivates use current petri net radio button since none is supplied.
     *
     * @param loadDialog the dialog to be shown
     */
    public ReachabilityGraph(FileDialog loadDialog) {
        stateSpaceLoader = new StateSpaceLoader(loadDialog);
        setUp();
    }

    /**
     * Main method for running this externally without PIPE
     *
     * @param args command line arguments
     */
    public static void main(String[] args) {
        JFrame frame = new JFrame("ReachabilityGraph");
```

```
        FileDialog selector = new FileDialog(frame, "Select petri net", FileDialog.LOAD);
        frame.setContentPane(new ReachabilityGraph(selector).panel1);
        frame.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
        frame.pack();
        frame.setVisible(true);
    }

    /**
     * @return main panel of the GUI
     */
    public Container getMainPanel() {
        return panel1;
    }
}
```