|  |
| --- |
|  |
| JPeNS – Group 5 |
| Java Petri Network Simulator |
|  |
| **Chris Hobbs** |
| **Jonathan Babbitt** |

April 10, 2012

|  |
| --- |
|  |

CMPT 166

Spring 2012

Professor H. Tsang

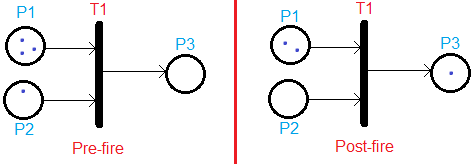
**What is JPeNS**

The Java Petri Network Simulator, or JPeNS for short, is a JAVA-based program that aids in the visual representation and learning of Petri Networks in an interactive way. It is meant to give the user both a hands-on and visual experience while learning about these networks by providing real-time feedback when transitions fire. To facilitate information-sharing there is import/export functionality built-in that allows learners to share their graphs with one another.

**Introduction to Petri Network Graphs**

Petri Networks are a way to visually represent or lay out a descriptive network for a trigger or action based system, these networks are drawn as directed bipartite graphs. Bipartite means that the nodes in the graph can be divided into two disjoint sets with edges connected nodes in opposite sets but not connecting to nodes within the same set. A directed graph means that the edges go in one direction, like one-way streets. In the case of Petri Network diagrams, the places and transitions are split into the two sets. That is to say, a place can only connect to a transition and a transition can only connect to a place. Places cannot connect to places and transitions cannot connect to transitions. However, multiple places can connect into one transition and one transition can connect out into multiple places. In a Petri Network graph the places are represented as circles and the transitions are represented as vertical bars.

The actions of the Petri Network function based off of tokens located in the places. Each transition requires one token from each precondition place to fire. If one or more of the preconditions does not have a token, the transition cannot fire. After the transition is fired, one token is generated in each of the post-condition places. The preconditions for a transition are all the places with directed edges (arrows) coming from the place into the transition. The post-conditions for the transition are all the places with directed edges coming from the transition into a place. [See figure 1.1]



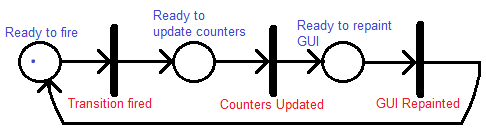
**Figure 1.1**

In the pre-fire state you can see that P1 contains three tokens, P2 contains one token, and P3 is empty. P1 and P2 are the preconditions for T1 and P3 is the post-condition. The post-fire state shows that T1 takes one token from both P1 and P2 and places one token in P3. This network cannot fire again as P2 is now empty so T1’s preconditions are no longer met.

JPeNS enables an easy way to create and distribute Petri Networks. It is very simple to make an XML file that defines the desired network and the file can be easily sent to anyone who wishes to view it. All they would need it JPeNS and the XML file; from there they can import it and view the network. This would be ideal for educating people on Petri Networks as the instructor could distribute premade XML files to the students for interactive examples.

**Why JPeNS**

We chose our topic as Petri Networks because they are a very interesting and useful topic. They have a wide range of applications such as software design, workflow management, process modeling, data analysis, diagnosis, and simulation. See figure 1.2 as an example of a Petri Network for software design.



**Figure 1.2**

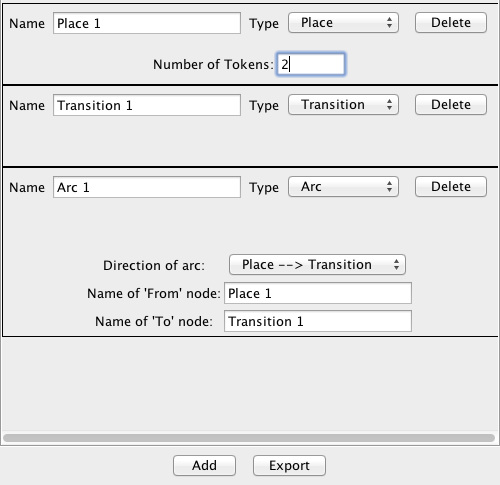
This is an example of a Petri network depicting software design. This network is the basic structure of how JPeNS’ simulation mechanism works.

As you can see, Petri Networks are a priceless tool when it comes to explaining, planning, or diagraming action-based systems. However, even though it is a valuable tool, it can be a difficult one to learn. Those who do not have a background in computing or engineering typically have a more difficult time grasping this concept. Since Petri Networks have such a broad range of possible uses, it’s a shame that it may be underutilized in other fields. This is where JPeNS comes in. It will make the education and understanding of this topic more intuitive so that more people may be inclined to learn and use Petri Networks in their field.

Currently there are other tools already available that will create graphs, even directed graphs. However, the tools available for mapping Petri Networks are very sparse. For example, while searching for existing tools we found the Simple-Java-Petrinet[[1]](#footnote-1) on GitHub but it lacks any visual representation.

## System Description

JPeNS has a simple layout divided into 3 sections, a header with the control buttons, a side panel that lists run-time information, and a central panel that shows a visual representation of the network. The control buttons in the header allow the user to “fire” individual transitions on the graph (provided they have their criteria met). The transitions use a color-coded system (which is made available through the help menu) to notify the user whether they can fire or not based on the input arcs to the transitions having the necessary number of tokens. Additionally there is a window that allows the user to create new Petri Network graphs.

When the user fires a transition they can observe the transfer of tokens by looking at the information panel located on the left-hand side of the application window. Each place displays the number of tokens it currently has so that at each firing the number can be monitored for change. In addition to the number of tokens changing in the information panel, the colors of the transitions and places change in the main graph window in relation to the ability of each transition to fire and whether each place contains any tokens or not.

**Figure 1.3**

Creating a new Petri Network graph for exporting and sharing.

Sharing information is an important component of JPeNS and so there is import/export functionality that allows students to create and share their graphs with one another. There is a dynamic form which allows users to create new objects to form their own graphs. They can choose what type each component is (place, transition, or arc) and then set properties for it. The file is saved in XML so that it can further be edited by hand if so desired. The File menu inside JPeNS also has an Import command that will load an XML file created by JPeNS and display it.

## Design

Arguably the most important aspect of JPeNS is the visual Graph layout since the way students are to learn from the program is by visually watching how transitions fire in the graph. Finding an appropriate library turns out to be somewhat challenging since the existing graphing libraries do not operate under the same constraints that Petri Network graphs do. For example, most libraries work well to provide a flow-chart layout but do not handle the constraint of all edges (arcs) having a place on one end and a transition on the other. Initially we found a small library that provided some basic logic for Petri Network graphs but had no visual representation, so we took this library and greatly expanded upon it.

Once we achieved a logically correct Petri Network simulator we began the search for a visual graphing library. Initially we looked at the open-source Graphviz[[2]](#footnote-2) library since one of us had previous experience with it. While this library could handle graphs, its Java support turned out to be rather weak unless we were to constantly render the graph to an image file and then display the image in the application. This would be both inefficient and would lack the interactivity we desired. After trying a few more libraries such as mxGraph, Jung, and JGraphT, we finally set ourselves on JGraphX[[3]](#footnote-3) since it provides both the interactivity that we want, and is based on Java. Being based on Java is the key to allowing it to interact with our logic. For example, all places, arcs, and transitions are mapped via classes in our logic code, but we didn’t want to have a redundant copy of all these objects inside a graph library since synchronizing them could be difficult and untrustworthy. So with JGraphX we were able to create JGraphX components inside each of our Petri Network object classes and then we could simply refer to our places, transitions, and arcs and modify their equivalent graph components internally.

JGraphX provides a decent amount of interactivity out of the box by allowing the user to freely rearrange the nodes on the screen to suit their preference. Initially this feature was almost useless since we were re-creating the graph from scratch each time a transition would fire. In order to make this feature useable we were required to invest resources to insert the JGraphX components into our original Petri Network classes of Places, Transitions, and Arcs. While the investment was large it was worth it to provide the user with consistency when they rearrange their graphs.

JGraphX provides many benefits but there are also some pitfalls with its documentation regarding custom stylization. Typically JGraphX is used inside a web browser with JavaScript instead of Java and in that case normal HTML and CSS styles would apply. However, when used inside a Java application the web standards do not apply and a list of styling options does not appear to exist. After hours of searching for a way to modify the margin between mxCells (the class which represents each object on the graph in JGraphX) and finding nothing, the graphs are initially rendered with nodes quite packed together. Fortunately the user can rearrange the nodes as they please in order to see things more clearly.

**Architecture**

JPeNS is divided into 3 packages: master, petrinet.gui, petrinet.logic. In master resides the tools to run the application (ie, main()) as well as the configuration data, the Importer/Exporter. Petrinet.gui contains the classes which are used for representing information to the User Interface, they also interface with the models in petrinet.logic. Petrinet.logic contains collections of objects that are used in Petri Networks. These objects contain their own individual parameters and requirements. For example, the Arc class knows that it requires a direction, a place, and a transition. Another example is the Transition class that tracks whether or not it is able to fire (has tokens at all of its input arcs).

**Results**

We did a couple of field tests with some people who have no prior knowledge of Petri Networks. First we tried to just explain to them what they were and how they worked. The majority of the test subjects reported that they still did not grasp the concept. We then tried explaining the concept with drawing diagrams and pictures and explaining what would happen after the transitions fired. This method did give the subjects something to look at however they had to still visualize what happened when a transition fired. This worked a little bit better than just straight explaining but some of the subjects were still not quite there for fully understanding it. We would give them an initial state of a Petri Network and asked them to draw what it would look like after certain transitions fired a given number of times. They had a little bit of trouble with doing this correctly. We then gave them example loaded up in JPeNS which allowed them to look at the network, see the values of tokens at each place and then step though each transition at their own pace. We then gave them another set of Petri Networks and asked them to draw what it would look like after certain transitions fired a given number of times. After using JPeNS nearly all the subjects answered the questions correctly.

From the tests performed it is obvious to see that JPeNS has had a positive impact on the education process. It both simplified and cut down on the number examples needed for the subject to comprehend the topic. It is fairly safe to say that JPeNS is an invaluable tool when it comes to teaching this concept.

**Conclusion**

Creating JPeNS has been a challenging and satisfying endeavor that has taught our team a number of new tools (Eclipse, Git, TortoiseMerge, Vim, JGraphX) and expanded our knowledge of Java GUI design and layout. Having a solid version control system is critical for projects such as these so that collaboration can happen with distributed backup and consolidation of resources. In addition, the ability to make branches in code is essential for testing various tools (switching between JGraphX, Jung, etc is as easy as switching branches in Git) without losing the ability to receive updates and patches from teammates.

In the end we are thrilled to be able to provide a working tool that can help users understand how these state machines work and to allow them to work with others by sharing their data. It is also our desire that others will eventually take the project to the next level with their own ideas by forking it on GitHub and developing it further.

1. Simple-Java-Petrinet: https://github.com/rmetzler/simple-java-petrinet [↑](#footnote-ref-1)
2. http://www.graphviz.org/ [↑](#footnote-ref-2)
3. JGraphX is based on mxGraph. http://www.jgraph.com/jgraph.html [↑](#footnote-ref-3)