Introducing *MyPetri.net*: A Petri Net Editor and Simulator for Students

Joseph M. Vanliew
College of Computing
Grand Valley State University
Allendale, MI, USA
vanliewj@mail.gvsu.edu

Erik M. Fredericks
College of Computing
Grand Valley State University
Allendale, MI, USA
frederer@gvsu.edu

Byron DeVries
College of Computing
Grand Valley State University
Allendale, MI, USA
devrieby@gvsu.edu

***Abstract*—Petri nets are a powerful formalism for modeling and analyzing concurrent, distributed, and complex systems. However, extensions that enable greater representative capabilities (e.g., coloured Petri nets) include significant complexities beyond elementary Petri nets or even Turing complete Petri nets with inhibitor arcs. Currently, there is a lack of free and open-source Petri net tools that are both maintained and free of the complications of coloured Petri nets. In this paper we present a free and open-source Petri net editor and simulator, *MyPetri.net*, that provides a modern web-based editor for Petri nets and several modes of execution or simulation. While more complex editors offer formal verification methods, our approach introduces a unit testing–like framework that allows users to set and verify place values after execution, similar to traditional software testing. Finally, we present an informal discussion on the impact of the *MyPetri.net* tool in the classroom.**

***Index Terms*—Petri nets, Editor, Verification, Educational Tools**

I. Introduction

The robust analysis of complex systems heavily relies on formalisms like Petri nets [1] and Finite State Machines [2]. However, rigorously verifying the dynamic behavioral properties of these models often remains a specialized task. A significant hurdle is the accessibility of verification tools that frequently demand deep formal methods expertise, thereby limiting their application by a broader range of system modelers, especially students. *MyPetri.net*, an intuitive web-based environment, directly confronts this challenge by integrating a user-centric verification toolkit. This toolkit empowers modelers to define and check specific behavioral properties by setting inputs and verifying outputs via configurable scenarios directly within the *MyPetri.net* modeling and simulation workflow, making Petri net modeling and verification more attainable.

Existing Petri net modeling solutions are either focused on significantly more complicated Petri net variants (e.g., coloured Petri nets [3]) or are no longer maintained [4] and, in some cases unrunnable on modern systems [4]. Recent developments of Petri net modeling and simulation systems often do not include inhibitor arcs [5], limiting the range of representable systems and omitting verification methods. Additional representative capabilities, like inhibitor arcs, have the dual impact of enabling higher theoretical and practical computability. Inhibitor arcs, like the extensions in coloured Petri nets, enable Turing Completeness [6] as well as greater representability for the system modeler. We provide simulation and modeling of Turing Complete Petri nets with test-based verification.

Petri nets, especially in their less extended forms, are often used to model concurrent systems. However, when viewed only as a modeling and simulation methodology, the more complex formal verification techniques are prohibitively difficult for early practitioners. However, just as executable systems can be tested using individual test-cases, executable specifications can be tested in the same way. Therefore, we introduce *MyPetri.net*, a Petri net modeling and simulation framework that not only allows a user-friendly interface for modeling and executing Petri nets with inhibitor arcs, but also allows for configuration of input values (i.e., places set with specific token counts) and expected results (i.e., places with required / verified token counts) after execution.

MyPetri.net backend was developed using a RESTful API written in Java using the SpringBoot framework using Gradle as a build environment. The front-end employed the React framework in Typescript using the Vite build tool. Both saved files and testing configurations are saved using the JSON format in order to support human readability. *MyPetri.net* supports a variety of execution, or simulation, modes in addition to configuration-based verification. Additional implementation details and features are discussed in detail in Section III.

The contributions of this paper are as follows within the *MyPetri.net* framework:

- A user-friendly tool for modeling and simulating Petri nets with significant "quality of life" user interface enhancements (e.g., save, load, copy, paste, undo), and
- A method for verification of modeled Petri nets using a configuration-based input/output specification based on places within the Petri net, and
- Free and open access to both the online tool[1] and source-code[2].

The remainder of the paper is organized into the following sections. Section II covers background information. Section III details the *MyPetri.net* tool while Section IV provides a

[1]https://mypetri.net
[2]https://github.com/Joseph-Vanliew/MyPetri

discussion of the impact within the classroom after a semester of use. Finally, Sections V and VI covers related work and our conclusions, respectively.

II. Background

This section discusses background information on Petri nets and verification.

A. Petri Nets

Petri nets, while often defined mathematically via tuples [7], can be understood via their graphical representations. Specifically, Petri nets are bipartite graphs where arcs, or arrows, connect places (circles) to transitions (rectangles) and transitions to places, but never place to place or transition to transition. A place can have a number of tokens or marks representing data within the system. Often the initial set of tokens in a place is called its marking. Transitions are fired when all places with an arc to a transition have at least one token *per* connecting arc. When a transition fires, a token or marking is consumed for each arc connected *to* the transition (i.e., arrows pointing to the transition from a place). Any arcs connected *from* the transition (i.e., arrows pointing from the transition to a place) gain a token or marking for each connected arc.

However, Petri nets with only standard arcs are not Turing Complete [6], therefore we also include inhibitor arcs (a line with a small circle instead of an arrow at the head) that connect a place to a transition (but never the other way around) and necessitate that the connected place be empty of tokens or markings in order to allow the transition to fire. For example, Figure 1 shows a simple Petri net with several places, two transitions, and both standard (e.g., from 'input1' to the left transition) and an inhibitor arc (i.e., from 'input2' to the left transition).

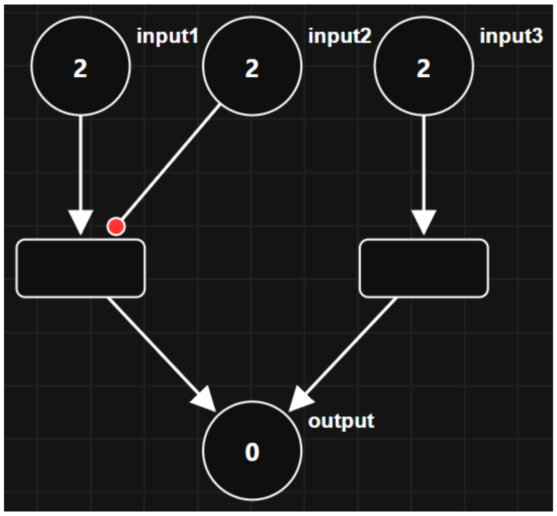


Fig. 1. Example Petri Net

In this Petri net only the right transition is enabled and could fire, due to the number of tokens in the 'input2' place where the inhibitor arc from 'input2' to the left transition requires zero tokens.

B. Verification

Similar to other models of systems, Petri nets have a rich range of formal analysis and verification that takes advantage of their mathematical properties to prove properties related to reachability [8], fairness [9], and others [10]. Additionally, model checking has been used to assess temporal properties of Petri nets [11], [12]. However, within the context of this paper our discussion of verification refers to software testing-based verification. That is, just as software is tested by providing known inputs to a software system and measuring the response of that software system against expected results [13] we provided predefined tokens for specific places in a Petri net. After executing that Petri net we compare specific places in the Petri net to predefined sets of tokens that we expect. Importantly, providing a single set of inputs and verifying a specific set of outputs for a software system or a Petri net does not guarantee correct behavior across *all* inputs [13].

III. *MyPetri* Tool

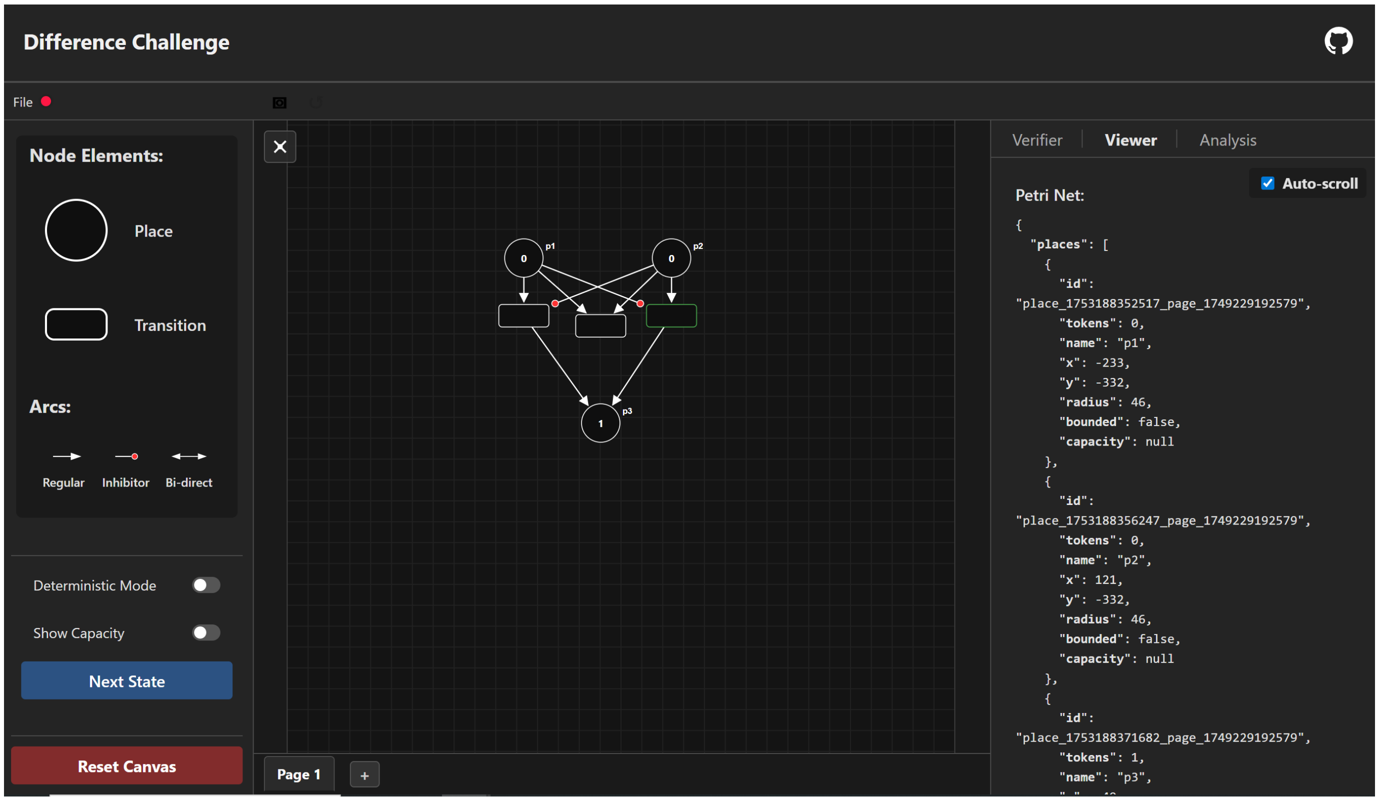
In this section we detail the features and interface of the *MyPetri.net* tool for designing, simulating, and testing Petri net models. Throughout this section we will reference elements of the graphical user interface as displayed in the screenshot in Figure 2.

A. Design Features

Petri net elements that can be added to the canvas (i.e., the middle area in Figure 2) are as follows:

- **Places**: Can be added by dragging the *place element* from the left on to the canvas. While the *place element* is selected, the initial marking of the tokens in the place can be altered by selecting the quantity of tokens in the middle of the place. Similarly, the a label can be set by selecting the place outside of the markings.
- **Transitions**: Can be added by dragging the *transition element* from the left on to the canvas, similar to the places. While the *transition element* is selected, a label can be set.
- **Arcs**: Standard arcs can be connected between any two places and transitions already on the canvas by selecting the first element and then the second element. Both place to transition and transition to place arcs are supported, based on the order of elements selected.
- **Bi-directional Arcs**: Similar to standard arcs, bidirectional arcs can be added to an existing transition and place. However, bidirectional arcs are equivalent to two standard arcs: one from the place to transition and one from the transition to place.
- **Inhibitor Arcs**: Can be added similar to standard arcs, but only from a place to a transition. No transition to place inhibitor arcs are allowed.

Throughout the editing and design of the Petri net within *MyPetri.net*, standard features for saving, loading, copying, pasting, undo, and alignment guides are included. These "quality of life" features ensure that the tool is not unpleasant to use. Additionally, descriptions of each arc are shown when



```
{
  "places": [
    {
      "id":
"place_1753188352517_page_1749229192579",
      "tokens": 0,
      "name": "p1",
      "x": -233,
      "y": -332,
      "radius": 46,
      "bounded": false,
      "capacity": null
    },
    {
      "id":
"place_1753188356247_page_1749229192579",
      "tokens": 0,
      "name": "p2",
      "x": 121,
      "y": -332,
      "radius": 46,
      "bounded": false,
      "capacity": null
    },
    {
      "id":
"place_1753188371682_page_1749229192579",
      "tokens": 1,
      "name": "p3",
```

Fig. 2. *MyPetri.net* Design Tool

they are selected and the JSON representation is automatically updated to the right of the canvas when ’Viewer’ is selected. While a single Petri net can be edited, additional canvas tabs can be added for multiple Petri nets within a single project.

Finally, by selecting the ’Show Capacity’ toggle on the left, a maximum number of tokens can be set for each Petri net within each place on the canvas. When the number is reached, additional tokens in the respective places are capped to the number listed.

B. Simulation Features

Petri nets can exhibit conflicts when multiple transitions are eligible to fire simultaneously, sometimes consuming tokens in a way that prevents other previously enabled transitions from firing, depending on the firing order. Rather than supporting exhaustive model checking to execute all such orders, we offer two distinct execution modes:

- **Deterministic Mode** toggle ***off***: Transitions are fired in random order. If a single transition is enabled, that transition will fire, but if more than one transition is enabled the transition that fires will not be deterministic.
- **Deterministic Mode** toggle ***on***: Transitions are fired in an order chosen by the user. If a single transition is enabled, no user input is required. However, if multiple transitions are enabled, the user must select between the transitions that are enabled to continue the execution of the Petri net.

Regardless of the mode the tokens are graphically shown traversing the arcs to and from the transition when they

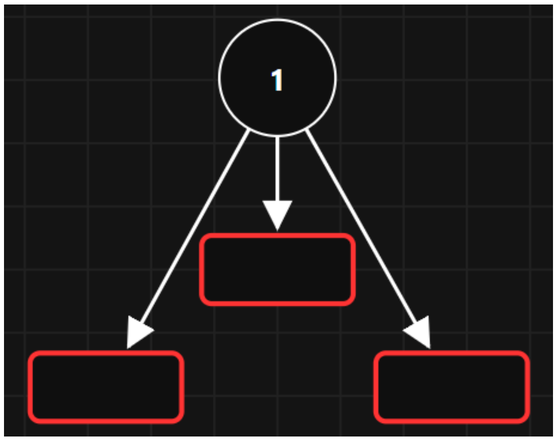
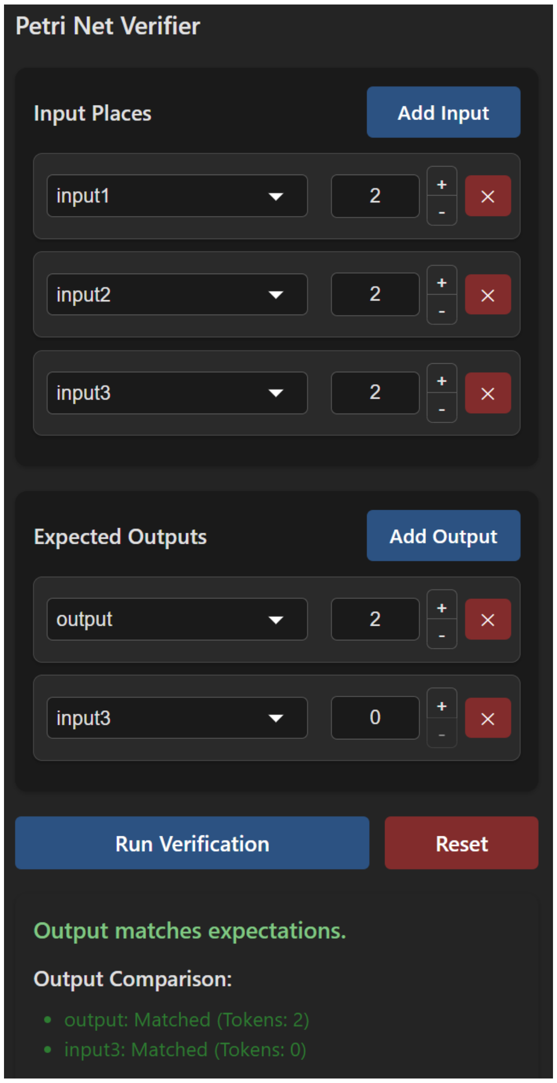


Fig. 3. Example Conflict Between Transitions

fire. Transitions that are enabled and fire are highlighted. Execution stops when no transitions are able to fire, or when the deterministic mode toggle is on, pauses when a transition selection must be made between conflicting enabled transitions as show by the selectable red transitions in Figure 3.

C. Verification Features

On the “Verification” tab on the right, named places can be added as predetermined inputs as well as expected outputs. For example, the sample Petri net in Figure 1 can be tested as shown in the screenshot in Figure 4.



input1 2
input2 2
input3 2

output 2
input3 0

Output matches expectations.

Output Comparison:

- output: Matched (Tokens: 2)
- input3: Matched (Tokens: 0)

Fig. 4. Example Verification

Each named place can be listed as an input and as an output for verification. Importantly, as inputs provide predefined tokens for specific places and outputs provide expected results, a single place can be both an input and an output in the same test. In the case of Figure 4, ‘input3’ is both listed under the inputs and outputs. The full set of inputs set the value of all places. Any places not listed will be initialized to zero. A maximum of 1,000 cycles will be run and failure will occur in any of the following cases:

- The Petri net still has transitions that can fire after the maximum number of cycles,
- The Petri net output places do not match the values listed for the outputs, or
- The Petri net encounters a conflict between two or more transitions that could fire simultaneously.

Since the behavior of the Petri net is executed and verified under these assumptions that the net *is* deterministic. The user cannot select which conflicting transition should be fired and transitions cannot be fired at random in the case of conflict, that non-determinism causes a failure during the verification process.

D. Web Application Design

MyPetri.net is meant to be accessible to anyone around the world on any PC or device with a web browser. Accessibility was an important part of the initial design process as educators around the world and their students would be able to access this on PCs or devices at home as well as complete lab work together in classrooms or lecture settings. Resulting files are small and easily shareable.

To support these goals, *MyPetri.net* is implemented as a modern web-application and is split into 3 components. A client, backend API, and deployment pipeline.

- Client: A React app with a custom canvas and SVG elements as well as arc drawing logic. Its virtual DOM is also particularly useful as objects in memory change rapidly and require many partial reloads. Vite handles the optimization of these static assets supporting reliably quick build time.
- Backend API: Built using the SpringBoot framework for Java it provides many starter dependencies and is appropriate for RESTful services like simulating the following states of a large object like a Petri net. SpringBoot’s Tomcat servlet integration is used for this project’s structure for handling routing and requests from the client.
- Deployment Pipeline: The application is ‘Dockerized’ packaging the Frontend client with the backend API server. Docker creates an image which is built and pushed to Google Cloud’s Artifact Registry which is subsequently released as a deployment to a Google Cloud Run Instance serving traffic via the *MyPetri.net* URL.

IV. Classroom Impact

This section includes a discussion of how the *MyPetri.net* tool effected the outcomes within a semester-long requirements specification course that included several weeks of Petri net modeling. In addition to modeling software systems using Petri nets, this course included challenges for representing different types of numerical computation in order for students to explore aspects of representability not available in other models (e.g., finite automata). This section introduces these challenges and discusses student performance using both pen-and-paper design and design using the *MyPetri.net* design tool along with Petri net template files and tests to verify their solutions.

A. Challenge: Addition

Before the *MyPetri.net* tool was developed, this challenge was presented to students in class and the majority came to an acceptable solution. The challenge itself is as listed: Given two places p_1 and p_2, each with a number of tokens, create a Petri net that returns the sum into a third place, p_3. An unfinished template Petri net with p_1, p_2, and p_3 is shown in Figure 5.

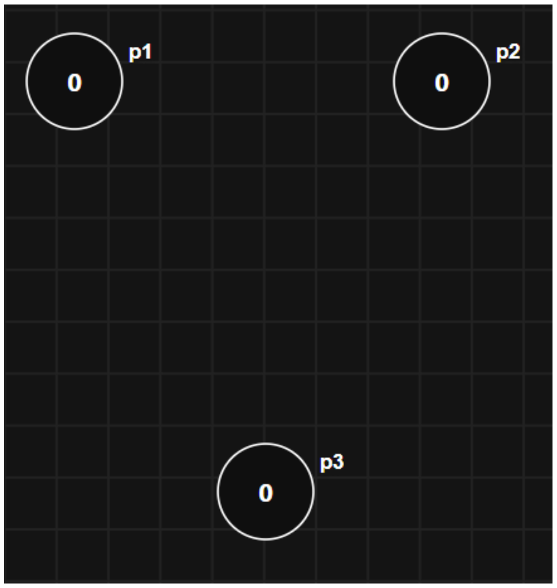


Fig. 5. Petri Net Template with Three Places

A valid solution is shown in Figure 6. As is true with all of these challenges, this solution is not the *only* solution. It would not be uncommon for students to have variations that still result in the correct result. Both p_1 and p_2 are connected to different transition with arcs that allow each marking to cause an independent transition to fire and place one token in place p_3.

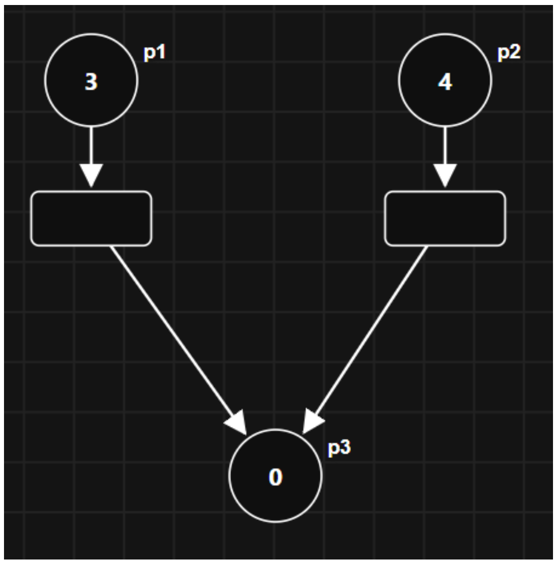


Fig. 6. Solution for Addition Challenge

Both before and after the *MyPetri.net* tool was developed almost all students came to an acceptable solution. Given the comparatively simplistic challenge, students were successful with both pen-and-paper alone as well as the *MyPetri.net* tool.

B. Challenge: Difference

Using a pen-and-paper approach, many of the students (roughly 3/4ths) would come to an acceptable solution. While more difficult than the last challenge due to the inclusion of inhibitor arcs, the principles of the last challenge carry forward for this challenge. The challenge itself is as listed: Given two places p_1 and p_2, each with a number of tokens, create a Petri net that returns the absolute difference, regardless of which place has more tokens, into a third place, p_3. An unfinished template Petri net with p_1, p_2, and p_3 is shown in Figure 5.

A valid solution is shown in Figure 7. Both places p_1 and p_2 can either cause the transition on their side (left or right) to fire or jointly cause the middle transition to fire. Due to the introduction of inhibitor arcs, neither the left nor right transition can fire until the place on the other side (p_2 and p_1, respectively) are empty. Therefore, the middle transition consumes one token from both the p_1 and p_2 places without creating any additional tokens in a subsequent place. This continues until either p_1 or p_2 is empty, at which point either the left or right transition will fire until the place nearest it (p_1 or p_2, respectively) is empty. The p_3 place will contain the absolute difference between p_1 and p_2 regardless if which place starts with a larger number of tokens. In the case of the initial markings in Figure 7, p_3 would have one token as its marking.

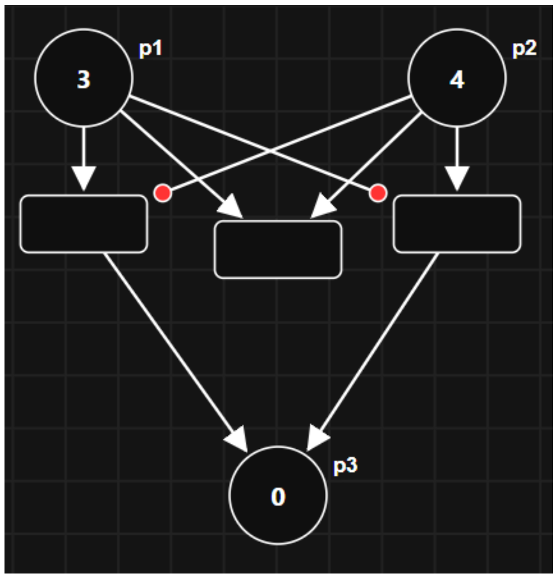


Fig. 7. Solution for Addition Challenge

After the *MyPetri.net* tool was developed almost all students came to an acceptable solution, despite the additional complexity.

C. Challenge: Multiplication

Before the *MyPetri.net* tool was developed, students manually designed and simulated Petri nets by hand. Due to the difficult nature of modeling with Petri nets, a minority of students considered them more desirable to use that other specification methods. While all students started the challenge during class time, the optional nature of the challenges meant that less than half of the students continued their attempts outside of class time after no students completed it in class.

The challenge itself is as listed: Given two places p_1 and p_2, each with a number of tokens, create a Petri net that returns the product into a third place, p_3. An unfinished template Petri net with p_1, p_2, and p_3 is shown in Figure 5.

A valid solution is shown in Figure 8, though it is significantly more complicated than the last two solutions. The 'outer loop' transition initially takes one token from p_1 to enable the 'inner loop' transition that removes tokens from p_2 and stores them in the place 'store.' This 'store' place allows p_2 to be reloaded by the 'reload' transition once p_2 is empty and the 'restart' transition fires taking a token from the 'e' place that enabled the 'inner loop' transition. Once the 'store' place is empty, the 'outer loop' transition can fire to re-enable the 'e' place with a token allowing the 'inner loop' transition to fire and the process repeats. Therefore, for each token removed from p_1, all tokens are removed from p_2 and then re-loaded. Once both p_2 is empty, it will no longer re-enable the 'inner loop' transition via the 'e' place and the Petri net will stop executing. Throughout, each time 'reload' is fired to reload a token into p_2, the 'reload' transition also outputs a token into p_3. The total tokens in p_3 is the tokens in p_2 times the tokens in p_3, or in the case of Figure 8, the initial markings result in 6 tokens in p_3.

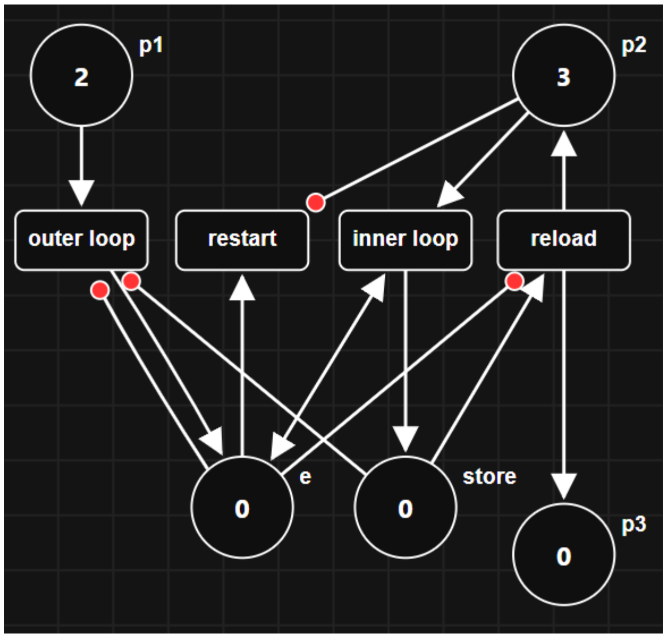


Fig. 8. Solution for Multiplication Challenge

After the *MyPetri.net* tool was developed, several students were able to complete the challenge during class time and several more completed during the optional out-of-class time. Perhaps most interesting was the method of design. While students who developed pen-and-paper Petri nets that they manually simulated would create the entire Petri net before trying to simulate, students using the *MyPetri.net* tool developed more iteratively. Specifically, they would allow the tool to execute partially completed Petri nets and, once a conflict between two transitions was identified requiring a manual choice during the interactive simulation, they would add additional arcs to limit once of the transitions.

D. Challenge: Division

Before the *MyPetri.net* tool was developed, this challenge was never presented to the students. Based on the success of a much higher proportion of students with the last challenge, this challenge was added.

The challenge itself is as listed: Given two places p_1 and p_2, each with a number of tokens, representing the dividend and divisor, respectively, create a Petri net that returns the floor of the quotient into a third place, p_3, and the remainder into a fourth Petri net, p_4. An unfinished template Petri net with p_1, p_2, p_3, and p_4 is shown in Figure 9.

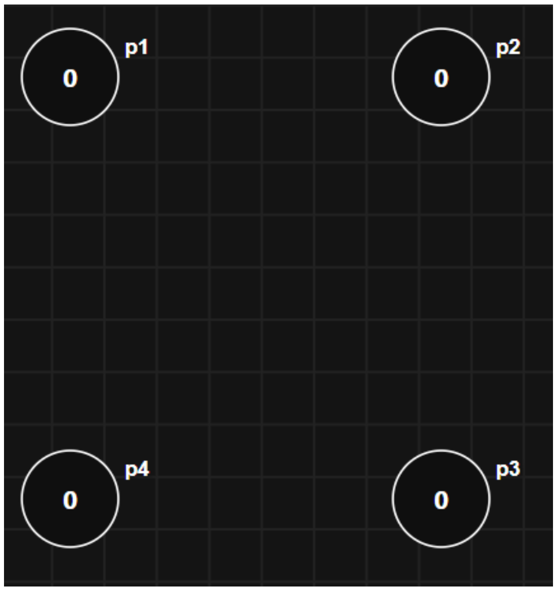


Fig. 9. Petri Net Template with Four Places

A valid solution is shown in Figure 10 where the 'load' transition decrements the tokens in both p_1 and p_2 until no more tokens remain in at least one. The 'enable' transition puts a token in the center place to enable the 'reload' transition that moves the original count in p_2 back into p_2 and removing the same quantity from p_4, at which point the 'disable' transition puts a token in p_3 indicating another even division of p_2 into p_1. If p_1 is evenly divisible by p_2, no further transitions are needed. If not, p_4 will retain the remainder due to the 'load' transition not firing due to zero tokens in p_1.

Impressively, despite being the most difficult challenge and providing no additional time compared to previous challenges, several students were able to apply the iterative development method to create versions of this Petri net that successfully solved the challenge presented.

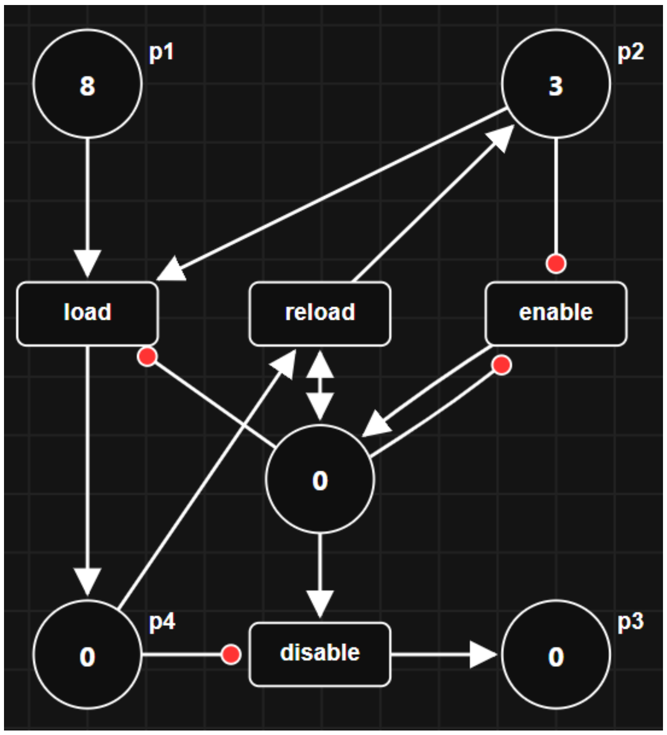


Fig. 10. Solution for Division Challenge

E. Summary

Colloquially, it is clear that students performed better and enjoyed the challenges more when working within the *MyPetri.net* tool along with provided verification tests. This is especially true as the challenges become more complex. While there is no formal comparison made between students groups and the student populations between semesters are entirely different the effect on student participation and success improved drastically. Additional formal studies would be needed to quantify this improvement, however additional challenges (i.e., the division challenge) were presented to the students and completed within the originally scheduled time allotted to Petri net content within the course. It is also worth noting, that larger-scale system modeling was not compared across pen-and-paper and *MyPetri.net* use.

V. Related Work

While Petri nets have been proposed for use in concurrent programming courses [14], our focus is on the modeling of Petri nets themselves outside of a programming course. Efforts to support Petri nets in the classroom include P3 [15], [16] which has not been maintained since publication in 2004 or tools that have not been publicly released [17].

The range of existing tools for Petri nets is quite large. Some existing general tools, like the Platform Independent Petri net Editor (PIPE) [18], [19] have not been maintained recently (since 2016). Fortunately, well maintained tools do exist, including:

- Workflow Petri Net Designer [20],
- Coloured Petri Net (CPN) Tools [21],
- TimeNET [22], and
- GRaphical Editor and Analyzer for Timed and Stochastic Petri Nets (GreatSPN) [23].

However, these tools are typically heavier weight focusing on business process modeling, coloured Petri nets, other significant extensions that increase complexity for students. Problematically, they also require installation which invariably causes compatibility issues across the range of student computing resources. The author's experience has been that it is difficult to use these otherwise excellent tools within the context of a limited engagement with Petri nets as a portion of a larger class focused on variety of topics and not just Petri nets.

In the form of *MyPetri.net* we introduce a web-based application for approachable Petri net design that has significant "quality of life" features and built-in test-based verification. Students can immediately bring up the webpage and use it without issue. While some web-based Petri net editors exist, often they are overly simplistic and do not include inhibitor arcs (e.g., APO [24] or, similar to the Online Petri Net Simulator [25], they do not have any built-in verification and minimal "quality of life" features.

Popular contemporary diagramming tools like *draw.io* that focus on general diagramming provide an excellent way for modelers and students to learn and practice system design concepts [26], [27]. Research has shown that diagramming tools significantly improve learning outcomes and concept retention in system design education [28]. However, *draw.io* and other similar general diagramming tools do not attempt to track and simulate the dynamic behavior of systems over time. While *draw.io* excels at static visualization, it lacks the capability to model and analyze the evolution of systems, such as state transitions, token flows, and concurrent processes that are essential for understanding complex system behaviors by the user. This is particularly important for systems like Petri Nets, where the ability to simulate and validate the dynamic interactions between components is crucial for system design and verification.

VI. Conclusion

In this paper we have presented *MyPetri.net* tool that allows design, simulation, and verification of Petri nets. Importantly, this tool has been designed with students in mind and allows for complex Petri net representations without the additional extensions introduced by coloured Petri nets. The features included have been detailed and all the Petri net graphics in this paper has been captured from *MyPetri.net*. Importantly, *MyPetri.net* is both free for use online and open-source to allow extensions as part of the existing project or forked projects.

As part of the development of *MyPetri.net* it was used in conjunction with a requirements specification course that includes coursework on Petri nets. Several challenges were described that had been used in subsequent offerings of the course as well as with *MyPetri.net*. Not only was student success positively impacted by transitioning from pen-and-paper methods, but students also changed their modeling behavior to use simulations of partially complete models to help identify missing arc connections and illicit necessary place and transition additions.

While the current version of *MyPetri.net* is sufficient for classroom use to help introduce students to Petri nets without the existing complexity that can be introduced by further extensions, we do plan additional further work. First, while we have focused on test-based verification we do plan additional analysis and formal verification tools in the 'under construction' section in the 'Analysis' section. Second, due to the JSON file format, we plan several follow on works based to both create and consume Petri nets via model transitions.

Acknowledgment

We gratefully acknowledge support from Grand Valley State University and the College of Computing.

References

[1] C. A. Petri, "Kommunikation mit automaten," Ph.D. dissertation, University of Bonn, Bonn, Germany, 1962.

[2] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *Bulletin of Mathematical Biophysics*, vol. 5, no. 4, pp. 115–133, 1943.

[3] K. Jensen, *Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use. (3 vols.)*, ser. Monographs in Theoretical Computer Science. Berlin, Heidelberg: Springer-Verlag, 1997.

[4] D. Duba, "Refitting pipe: Extending the petri net tool pipe 5," Master's thesis, Leiden Institute of Advanced Computer Science, Leiden University, Leiden, The Netherlands, 2016.

[5] A. David, L. Jacobsen, M. Jacobsen, K. Y. Jørgensen, M. H. Møller, and J. Srba, "TAPAAL 2.0: Integrated development environment for timed-arc petri nets," in *Tools and Algorithms for the Construction and Analysis of Systems*, ser. Lecture Notes in Computer Science, vol. 7214. Berlin, Heidelberg: Springer, 2012, pp. 492–497.

[6] T. Agerwala, "A complete model for representing the coordination of asynchronous processes," Johns Hopkins University, Baltimore, MD, USA, Tech. Rep., 1974.

[7] J. L. Peterson, "Petri nets," *ACM Computing Surveys (CSUR)*, vol. 9, no. 3, pp. 223–252, 1977.

[8] E. W. Mayr, "An algorithm for the general petri net reachability problem," in *Proceedings of the thirteenth annual ACM symposium on Theory of computing*, 1981, pp. 238–246.

[9] E. Kindler and W. M. van der Aalst, "Liveness, fairness, and recurrence in petri nets." *Inf. Process. Lett.*, vol. 70, no. 6, pp. 269–27, 1999.

[10] J. Sifakis, "Structural properties of petri nets," in *International Symposium on Mathematical Foundations of Computer Science*. Springer, 1978, pp. 474–483.

[11] B. DeVries, "Mapping of uml diagrams to extended petri nets for formal verification," 2013.

[12] K. Wolf, "How petri net theory serves petri net model checking: a survey," in *Transactions on Petri Nets and Other Models of Concurrency XIV*. Springer, 2019, pp. 36–63.

[13] P. C. Jorgensen and B. DeVries, *Software Testing: A Craftsman's Approach*, 5th ed. CRC Press, 2021.

[14] J. P. Barros, "Specific proposals for the use of petri nets in a concurrent programming course," in *Proceedings of the 7th annual conference on Innovation and technology in computer science education*, 2002, pp. 165–167.

[15] D. Gasevic and V. Devedzic, "Software support for teaching petri nets: P3," in *Proceedings 3rd IEEE International Conference on Advanced Technologies*. IEEE, 2003, pp. 300–301.

[16] D. Gašević and V. Devedžić, "Teaching petri nets using p3," *Journal of Educational Technology & Society*, vol. 7, no. 4, pp. 153–166, 2004.

[17] C. Mei, X. Zhang, W. Zhao, K. Periyasamy, and M. Headington, "A tool for teaching petri nets," *Journal of Computing Sciences in Colleges*, vol. 26, no. 5, pp. 181–188, 2011.

[18] P. Bonet, C. M. Lladó, R. Puijaner, W. J. Knottenbelt *et al.*, "Pipe v2. 5: A petri net tool for performance modelling," in *Proc. 23rd Latin American Conference on Informatics (CLEI 2007)*, 2007, pp. 1–12.

[19] N. J. Dingle, W. J. Knottenbelt, and T. Suto, "Pipe2: a tool for the performance evaluation of generalised stochastic petri nets," *ACM SIGMETRICS Performance Evaluation Review*, vol. 36, no. 4, pp. 34–39, 2009.

[20] T. Freytag, "Woped–workflow petri net designer," *University of Cooperative Education*, pp. 279–282, 2005.

[21] K. Jensen, L. M. Kristensen, and L. Wells, "Coloured petri nets and cpn tools for modelling and validation of concurrent systems," *International Journal on Software Tools for Technology Transfer*, vol. 9, no. 3, pp. 213–254, 2007.

[22] R. German, C. Kelling, A. Zimmermann, and G. Hommel, "Timenet: a toolkit for evaluating non-markovian stochastic petri nets," *Performance Evaluation*, vol. 24, no. 1-2, pp. 69–87, 1995.

[23] E. G. Amparore, G. Balbo, M. Beccuti, S. Donatelli, and G. Franceschinis, "30 years of greatspn," *Principles of Performance and Reliability Modeling and Evaluation: Essays in Honor of Kishor Trivedi on his 70th Birthday*, pp. 227–254, 2016.

[24] A. Jagusch, "APO: An application for online petri net design and analysis," https://apo.adrian-jagusch.de/, 2014.

[25] P. Gburzynski, "Online petri net simulator," https://petrinetsimulator.com, accessed: 2025-07-23.

[26] J. Smith, S. Brown, and M. Wilson, "Visual modeling tools in software engineering education: A systematic review," *Journal of Software Engineering Education*, vol. 15, no. 2, pp. 45–67, 2023.

[27] R. Johnson and J. Lee, "The impact of visual modeling tools on system design learning outcomes," *International Journal of Software Engineering Education*, vol. 8, no. 3, pp. 112–128, 2022.

[28] W. Chen, C. Martinez, and L. Thompson, "Comparative analysis of visual modeling tools in software engineering education," *Journal of Computing Education*, vol. 12, no. 1, pp. 78–95, 2023.