WatchPoints for Renew*

Marcel Hansson, Daniel Moldt, Kim Wittenburg and Christian Bracker

University of Hamburg, Faculty of Mathematics, Informatics and Natural Sciences, Department of Informatics
http:// www.paose.de

Abstract

Petri net–based simulation frameworks are widely used to model, analyze, and execute concurrent and distributed systems. In Renew, reference nets support the development of complex, executable net-based applications, such as those built with the Paose framework. While Renew already provides breakpoints for interactive debugging, these mechanisms interrupt execution and are therefore limited for observing long-running or application-like simulations.

In this paper, we introduce WatchPoints, a non-intrusive observation mechanism for Petri net simulation in Renew. WatchPoints can be attached to net elements and are triggered during simulation without stopping execution. This allows modelers to observe firing behavior, execution frequencies, and interaction patterns while preserving the dynamics of the modeled system. As such, WatchPoints support exploratory modeling, debugging, and analysis of Petri nets beyond traditional stepwise execution. WatchPoints thus enhance the observability of Petri net executions and provide a general foundation for non-intrusive monitoring of executable net models.

WatchPoints are designed to integrate seamlessly into existing Renew workflows and can be connected to external monitoring infrastructures. In particular, WatchPoint triggers could be used to create Kieker monitoring records in the future. Which could enable the observation of Petri net execution using established Java monitoring and analysis tools.

As an outlook, this provides a foundation for seamless monitoring of applications that combine Petri nets and Java code, as commonly found in Paose-based systems.

Keywords

Petri Nets, Petri Nets Execution, Simulation, Observability

1. Introduction

Breakpoints are an important concept in system development, and some Petri net tools such as Renew already provide them, along with basic logging of transition firings. However, understanding net models becomes more difficult when other types of events are relevant, such as concrete time measurements. Additionally, tool developers are interested in detailed performance data, which requires more advanced observation mechanisms.

Monitoring and observing systems, software, or models is therefore essential and demands sophisticated approaches. A common technique in software systems is code instrumentation, where additional probes are inserted to collect runtime information such as execution times or interaction patterns Besides commercial solutions, Kieker offers such instrumentation and analysis capabilities for Java systems [1]. How similar mechanisms can be applied to Petri net tools remains an open question.

To address this, we develop a prototype for Renew to explore extensions for observability. The core idea is to introduce WatchPoints, which currently provide simple logging for net events. These logs can be analyzed and visualized, e.g., via heatmaps. In the future, WatchPoints will integrate with Kieker to generate monitoring records.

The remainder of the paper is structured as follows: Section 2 introduces foundations, Section 3 defines objectives, Section 4 describes functionality,

Section 5 compares related work, and Section 6 concludes.

PNSE'26, International Workshop on Petri Nets and Software Engineering, 2026

* Supported by participants of our teaching project classes and many student theses.

2. Foundations

This section briefly recalls relevant foundations: reference nets, the Renew tool, and the Paose framework.

2.1. Renew

WatchPoints are implemented as a plugin for Renew, the reference net workshop [2]. Renew is an editor and simulator for Petri net formalisms, particularly reference nets. Since its initial release in 1999, it has evolved continuously, with version 4.2 released in 2025. Its plugin-based architecture enables extensions such as WatchPoints.

2.2. Reference nets

Reference nets extend classical Petri nets with high-level and high-order concepts [3]. In contrast to simple P/T nets, tokens in reference nets may represent arbitrary Java objects and references to other nets, enabling hierarchical modeling [4].

Interactions are realized via synchronous channels [5], and transitions can invoke Java methods. This integration of net structure and executable code makes reference nets well suited for modeling distributed and application-oriented systems.

2.3. Paose

The Paose framework builds on reference nets and Renew to support Petri net–based applications [6, 7]. Here, nets act as executable components interacting with Java code. Observing execution behavior, such as firing frequencies or timing, becomes crucial, highlighting the usefulness of WatchPoints.

2.4. Kieker

Kieker is a observability framework for Java applications [8, 1]. It supports performance analysis and architecture discovery. In this work, Kieker is relevant as a future integration target for WatchPoints to capture Petri net execution events.

3. Objectives

WatchPoints aim to provide insights into the execution behavior of reference nets. They enable non-intrusive observation of net events during simulation, preserving system dynamics while recording relevant data such as execution times. Unlike breakpoints, WatchPoints do not interrupt execution but log events like transition firings or marking changes. This supports analysis of execution frequencies, interaction patterns, and temporal behavior, insights that are difficult to obtain otherwise.

WatchPoints target reference net–based applications in Renew, including Paose systems, but are equally applicable to general Petri net modeling. Typical use cases include

- modeling, where WatchPoints support analysis and validation by providing insights into net behavior;
- Petri net based applications, where WatchPoints allow observability of the running net simulations;
- development (of Petri net–based applications and Renew), where they assist in identifying performance bottlenecks.

The primary intended users are modelers, Paose developers, and Renew researchers. WatchPoints aim to improve observability across modeling, development, and application contexts, in particular where detailed runtime information is required for analysis or optimization.

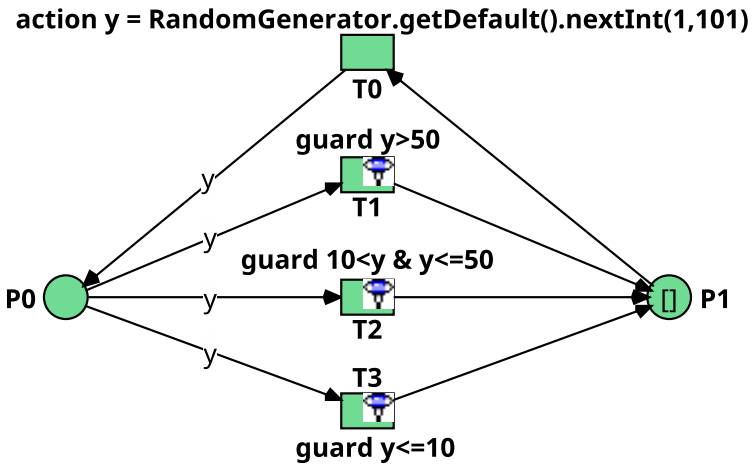

Figure 1: Net *WP-Example* with WatchPoints at T1 to T3.

4. Functionality

WatchPoints extend Renew by adding non-intrusive observation points to net elements. They are attached to places or transitions and trigger on execution events, producing log entries instead of interrupting simulation.

4.1. Using WatchPoints

WatchPoints are managed similarly to breakpoints via menu entries and are visually indicated in the editor. They can be attached to selected net elements and remain active during simulation. Once attached, WatchPoints are visually indicated directly in the drawing view of Renew using a small eye symbol. Figure 1 illustrates a reference net with attached WatchPoints. The transition *T0* generates a random number between 1 and 100 (inclusive) on place *P0*. The watched transitions *T1* to *T3* have different guards for the token value. Based on the guards *T1* is the most likely to fire and *T3* the least likely in one iteration.

4.2. Triggering and Reactions

WatchPoints react to events such as transition firing (start, complete) or place marking changes (token added/removed). When triggered, they generate log entries. Currently, only CSV logging is supported, but future extensions will integrate Kieker (see next section). These events are for transitions:

- fire start, the beginning of the transition firing (in a Reference net, this is before the transition actions are executed);
- fire complete, the end of the transition firing.

For places the events are:

- marking changed, the tokens on a place change;
- $+1$ token, the place gets one additional token;
- -1 token, the place loses a token;
- all tokens removed, the place lost at least one token and is now unmarked.

4.3. Logging and Observing Output

The current implementation produces CSV logs containing timestamps, net and element identifiers, and event types. This format is simple and widely supported, making it suitable for debugging and analysis. However, it does not scale well for large simulations, motivating future integration with Kieker.

```
1769781230023,WP-Example,0,T3,"[3,_default]",transitionFiringFinished
1769781230029,WP-Example,0,T3,"[3,_default]",transitionFiringFinished
1769781230030,WP-Example,0,T1,"[4,_default]",transitionFiringFinished
1769781230033,WP-Example,0,T1,"[4,_default]",transitionFiringFinished
```

Listing 1: Log excerpt for the example net

In listing 1 a sample log output is shown from a simulation of the net shown in Figure 1. The log contains in that order: a timestamp, net name, net id, element name, element id (with a group indicator in case a graphical element belongs to another semantic element), event type. The first line for example means, that at timestamp of 1769781230023ms in the net *WP-Example* with an id of 0 the transition *T3* with an id of 3 finished firing. The next line tells us, that T3 fired a few milliseconds later again.

The standard text logging format of Kieker is quite similar. It is also a CSV based format with timestamps, but instead of net informations it contains information regarding the tracing of methods (e.g. method name, call stack information, trace references) and environment (e.g. service/application name). In addition Kieker offers other formats, like binary compressed files or the option to send the data via a network connection somewhere else (e.g. a different computer for collecting the data of distributed simulations).

Using Kieker instead of our simple logging implementation would provide a few main benefits. It comes with an existing architecture for writing and reading the data, which would mean, that we don't need to implement this our self. This architecture is very optimized and designed to produce the least overhead possible, so that the impact on the running Paose applications would be minimal. It also scales very well with bigger applications. Kieker is made mainly for Java application and the Paose application built with the Java program Renew are a mix of nets and Java Code. Using Kieker for WatchPoints allows observing all three aspects (nets, Renew itself, Java Code of Paose) with one framework, combining the data and allowing deep analysis.

4.4. Analysis and Visualization

The first iteration of the WatchPoint that is presented here, has a limited set of analysis and visualization capabilities. Currently, it can apply heatmaps and calculate intervals between firing or marking changes. Analysis functions can be invoked via their respective menu entries. At the moment, the analysis and visualization tools process the complete monitoring log generated during simulation runs.

The *Apply Heatmap* command colors the elements in a net based on their occurrences in the monitoring log. Figure 2 shows the previous example net, now with an applied heatmap. The heatmap interpolates in the HSV color space to color elements from blue for low temperatures, over green and yellow to red for high temperatures. The temperature corresponds to the number of times the net element occurs in the WatchPoint log. Coloring is done relative to the most active net element. The *Interval Calculation* command can be used to analyze the interval between consecutive events, like the time it takes (average, minimum, maximum) between two transitions firing (or one transition firing a second time).

5. Comparison

WatchPoints are conceptually similar to monitoring features in CPN Tools [9], which also provide non-intrusive observation. However, reference nets in Renew integrate Java behavior and dynamic structures, leading to different monitoring requirements. More broadly, WatchPoints align with tracing approaches in distributed systems [10], where execution events are recorded without affecting system behavior.

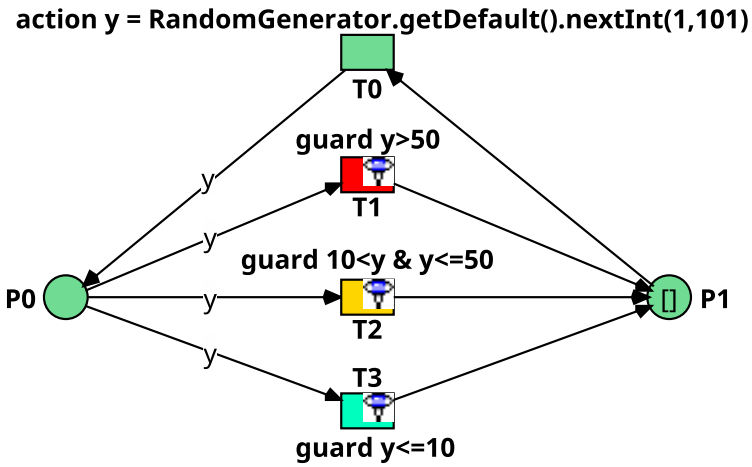

Figure 2: The WP-Example net with an applied heatmap.

Renew itself offers tool for monitoring already in the Paose approach for the Multi-Agent Nets (Mulan) implementation, namely the *Mulan Viewer* and *Mulan Sniffer* [11]. These are specifically tailored for the agent system. WatchPoints in contrast, work for all net simulations and are not bound to Mulan or Paose. In addition, the information provided by WatchPoints are different (e.g. concrete time measure within one net).

6. Conclusion

6.1. Summary

This paper introduced WatchPoints, a non-intrusive observation mechanism for Petri net simulations in Renew. WatchPoints extend traditional breakpoint-based debugging by allowing execution events to be recorded without interrupting simulation, providing structured insight into net behavior. Overall, WatchPoints enhance the observability of reference net simulation, facilitating exploratory modeling, debugging, and performance analysis. Even these simple extensions demonstrate that both modelers and tool developers can gain insights that would otherwise be unavailable without the implemented functionality.

6.2. Outlook

The current modifications of Renew are only one example of how our approach can be used. Clearly more advanced features can also be added to Renew. We currently work within a research project together with the main developers of Kieker to enhance Kieker and Renew in a sustainable way. While Renew provides a special, non-trivial test case, Kieker provides the technical background to implement several concepts and ideas for our Petri nets domain.

As Kieker is also capable to monitor in a distributed setting, we are currently investigating the option to transfer our gained insights to the implementation within our distributed Renew simulation environment [12]. In distributed Petri nets simulations the understanding of the models, the overall behavior and the tool performance is even more crucial. There we expect to have even higher impact for modelers and tool developers.

Acknowledgment This research is funded by the Deutsche Forschungsgemeinschaft (DFG – German Research Foundation), grant no. 528713834.

Declaration on Generative AI

During the preparation of this work, the authors used ChatGPT, DeepL and LanguageTool in order to: Grammar and spelling check, Paraphrase and reword. After using these tool(s)/service(s), the authors reviewed and edited the content as needed and take full responsibility for the publication's content.

References

[1] W. Hasselbring, A. van Hoorn, Kieker: A monitoring framework for software engineering research, Software Impacts 5 (2020). doi:10.1016/j.simpa.2020.100019.

[2] O. Kummer, F. Wienberg, M. Duvigneau, L. Cabac, M. Haustermann, D. Mosteller, M. Hansson, Renew – the Reference Net Workshop, 2025. URL: http://www.renew.de/, release 4.2.

[3] O. Kummer, Referenznetze, Logos Verlag, Berlin, 2002. URL: http://www.logos-verlag.de/cgi-bin/engbuchmid?isbn=0035&lng=eng&id=.

[4] R. Valk, Petri nets as token objects - an introduction to elementary object nets, in: J. Desel, M. Silva (Eds.), 19th International Conference on Application and Theory of Petri nets, Lisbon, Portugal, number 1420 in Lecture Notes in Computer Science, Springer-Verlag, Berlin Heidelberg New York, 1998, pp. 1–25. URL: https://doi.org/10.1007/3-540-69108-1_1.

[5] S. Christensen, N. Hansen, Coloured Petri nets extended with channels for synchronous communication, in: R. Valette (Ed.), ATPN, volume 815 of *Lecture Notes in Computer Science*, Springer-Verlag, 1994, pp. 159–178.

[6] L. Cabac, Modeling Petri Net-Based Multi-Agent Applications, Dissertation, University of Hamburg, Department of Informatics, Vogt-Kölln Str. 30, D-22527 Hamburg, 2010. URL: https://ediss.sub.uni-hamburg.de/handle/ediss/3691.

[7] L. Cabac, M. Haustermann, D. Mosteller, Software development with Petri nets and agents: Approach, frameworks and tool set, Sci. Comput. Program. 157 (2018) 56–70. URL: https://doi.org/10.1016/j.scico.2017.12.003.

[8] A. van Hoorn, J. Waller, W. Hasselbring, Kieker: A framework for application performance monitoring and dynamic software analysis, in: Proceedings of the 3rd ACM/SPEC International Conference on Performance Engineering (ICPE 2012), ACM, 2012, pp. 247–248. doi:10.1145/2188286.2188326.

[9] K. Jensen, L. M. Kristensen, Coloured Petri Nets: Modelling and Validation of Concurrent Systems, Springer, 2009.

[10] B. H. Sigelman, L. A. Barroso, M. Burrows, et al., Dapper, a large-scale distributed systems tracing infrastructure, in: Proceedings of the 2010 ACM SIGOPS Symposium on Operating Systems Principles, ACM, 2010, pp. 1–13.

[11] L. Cabac, T. Dörges, H. Rölke, A monitoring toolset for Paose, in: R. Valk, K. M. van Hee (Eds.), 29th International Conference on Application and Theory of Petri Nets, Xi'an, China, volume 5062 of *Lecture Notes in Computer Science*, Springer-Verlag, 2008, pp. 399–408. URL: http://dx.doi.org/10.1007/978-3-540-68746-7_26.

[12] L. Clasen, S. Bartelt, Y. Stahl, D. Moldt, Distributed P/T net simulation prototypes based on event streaming, in: M. Köhler-Bußmeier, D. Moldt, H. Rölke (Eds.), Proceedings of the International Workshop on Petri Nets and Software Engineering 2024 co-located with the 45th International Conference on Application and Theory of Petri Nets and Concurrency (PETRI NETS 2024), June 24 - 25, 2024, Geneva, Switzerland, volume 3730 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2024, pp. 192–216. URL: https://ceur-ws.org/Vol-3730.