# Runtime Verification of Timed Petri Nets*

José Ignacio Requeno Jarabo[1,*,†], Elena Gómez-Martínez[1,†], Hannes Kallwies[2,†], Melanie Haustein[2,†], Martin Leucker[2,†], Volker Stolz[3,†] and Patrick Stünkel[3,†]

[1]Universidad Complutense de Madrid, Calle del Prof. José García Santesmases, 9, 28040, Madrid, Spain
[2]University of Lübeck, Maria-Goeppert-Str. 3, 23562, Lübeck, Germany
[3]Western Norway University of Applied Sciences, Inndalsveien 28, 5063 Bergen, Norway

**Abstract**

Timed Petri net (TPN) is a type of Petri net for modeling concurrent systems that incorporates time durations as first-class citizens. This paper provides a means for analyzing TPN by runtime verification, a lightweight verification approach where a single run of a system is monitored. In particular, we assume the discrete-time semantics of TPN. We propose a TPN module for the definition and analysis of a TPN in the runtime verification framework TeSSLa, which is composed of the TeSSLa language, an interpreter, and a dedicated hardware monitor for online real-time analysis with minimal intrusion. The TeSSLa interpreter receives the definition of the TPN, the properties to analyze, and a set of execution traces and outputs an evaluation report. Our solution provides an efficient and user-friendly approach for the runtime verification of concurrent systems that are described by a high-level modeling language. For a practical evaluation of the approach, we have used parts of the library to model and analyze the workflow for examining specimens within the pathology department of Bergen's hospital.

**Keywords**

Timed Petri Net, Runtime Verification, TeSSLa

## 1. Introduction

A Petri net (PN) is an automaton-based modeling formalism that is suitable for describing concurrent systems [1, 2]. In particular, timed Petri net (TPN) is a type of PN that incorporates time durations for transitions as first-class citizens [3]. Time information in Petri nets is represented in different formats, including discrete and continuous semantics, which helps organize events in simulation traces [3]. The distinct features of time mechanisms depend on the specific application domain, such as whether event durations are modeled using deterministic or random variables, and whether time is linked to places [4], transitions [5] or tokens [6] in the PN. For this paper, we assume discrete-time semantics of TPN, aligning with the time semantics from Colored PNs and its companion, the CPN Tools framework [7].

There already exist tools and algorithms for computing and analyzing the properties of the TPN and its state space [8]. Nevertheless, some properties are unfeasible or impossible to statically analyze on a TPN, specially for infinite state spaces. Hence, the simulation of TPNs arises as an alternative to the exhaustive exploration of those systems. Thus, the simulation of a TPN returns an execution trace whose events are sorted by timestamp. The inspection of the execution logs can still solve interesting questions and, for instance, extract performance metrics such as system response time or average resource usage. This paper provides a means for analyzing TPN by runtime verification, a lightweight verification approach concerned with verifying a single system run by means of monitoring [9].

To this end, we propose the monitoring of time-related properties in the TeSSLa (Temporal Stream Based Specification Language) framework, a Stream Runtime Verification (SRV) language [10, 11]

*PNSE'24, International Workshop on Petri Nets and Software Engineering, 2024*
*Corresponding author.
†These authors contributed equally.
jrequeno@ucm.es (J. I. Requeno Jarabo); mariaelena.gomez@ucm.es (E. Gómez-Martínez); kallwies@isp.uni-luebeck.de (H. Kallwies); leucker@isp.uni-luebeck.de (M. Leucker); Volker.Stolz@hvl.no (V. Stolz); Patrick.Stuenkel@hvl.no (P. Stünkel)
0000-0001-5111-8357 (J. I. Requeno Jarabo); 0000-0002-7753-3345 (E. Gómez-Martínez); 0000-0002-8301-4752 (H. Kallwies); 0000-0002-3696-9222 (M. Leucker); 0000-0002-1031-6936 (V. Stolz); 0000-0002-0537-295X (P. Stünkel)

with an associated tool chain, including an interpreter and compiler. TeSSLa offers more flexibility than conventional *ad-hoc* extensions for programming software monitors for TPN and formalizes the execution traces in terms of streams. Transition-events from a TPN execution serve as input-events to TeSSLa. To simplify writing concise TeSSLa-specifications for TPN models, we have developed a TeSSLa library with functions for monitoring (timed) properties of Petri nets. Additionally, the TeSSLa framework offers the possibility to deploy a virtual copy of the TPN in a hardware device for online real-time analysis of systems with minimal intrusion, if no properties are monitored which require dynamic data structures.

The paper is organized as follows. Section 2 gives an overview of the relevant features of TeSSLa. Then, we present the basic concepts of a TPN and our new TPN library in Section 3. A case study illustrates the capabilities of our approach in Section 4. Section 5 outlines the related work. Finally, Section 6 gathers the conclusions and outlines the future work. A public repository contains the input models, intermediate files, and results presented in this paper, as well as a video illustrating the full workflow with the Titan tool [12].

## 2. Preliminaries: TeSSLa framework

We make use of the Temporal Stream Based Specification Language (TeSSLa) for monitoring runtime properties on TPNs. TeSSLa is composed of a specification language [10] and a tool chain [11] for Stream Runtime Verification (SRV). SRV formulates a monitoring property in terms of a stream transformation from input streams to output streams bearing information about the monitored property. SRV allows for more complex monitor outputs than just truth domain verdicts which only indicate whether the monitored property is satisfied by the received trace or not. For instance, an SRV monitor can calculate statistics on the received data such as the average duration during the occurrence of specific events.

The fundamental concept of SRV is the concept of a stream. TeSSLa uses timed streams, i.e., streams where every event has a timestamp from a global time domain $\mathbb{T}$ (e.g., $\mathbb{N}$ or $\mathbb{R}$) attached. TeSSLa assumes monotonicity and continuity of streams [10]. In other words, the output streams are based on the previous values of the input streams, and the output events can be computed incrementally (i.e., they are *future independent*). Continuity implies time progression, so the clock increases a finite number of time units until a new event happens.

As already stated, a TeSSLa specification defines a set of input streams and their types. Events from these streams are incrementally fed from outside to the monitor, which is generated from the specification. The specification additionally contains definitions of intermediate streams. Therefore, it assigns expressions over input- and intermediate streams to the identifiers of intermediate streams. Some of the intermediate streams are marked as output streams. The events of these output streams are incrementally printed by the monitor as soon as all input events they depend on have arrived.

The expressions used for stream definitions in TeSSLa are applications of six core operators on input or intermediate stream identifiers. These core operations provide basic functionality, like applying a function to the current values of other streams, accessing last events, retrieving the timestamp of certain events, or generating events at new timestamps. The TeSSLa implementation [11] allows the definition of functions that can be lifted to streams directly in the specification in a functional style. Additionally, it provides the possibility to define macros, i.e., new operators based on the core operators. The implementation also comes with a standard library of several frequently used macros.

## 3. TeSSLa Library for Monitoring Timed Petri Nets

The TeSSLa library we have developed for this paper offers dedicated macros and functions for monitoring complex (timed) properties of Petri nets. SRV is a useful approach for monitoring advanced properties, like the residence time of tokens in a place or the throughput of certain transitions, on simulated or real executions of Petri nets. In this context, SRV is able to complement the static analysis

of the Petri net, especially for properties where a static determination is impossible or intractable because of infinite state space.

Generally speaking, a Petri net is composed of places, transitions, arcs, and tokens flowing through the net. The notions of parallelism, synchronization or shared resources are determined by the way the places and transitions are connected. We work with a timed extension of Petri nets (TPNs), where timed transitions fire a token to the following place according to a predefined delay. For example, Fig. 1 shows such a TPN where the tasks in the `Todo` place require two resources from the pool of `Workers` in order to complete it in 2 hours (2h).

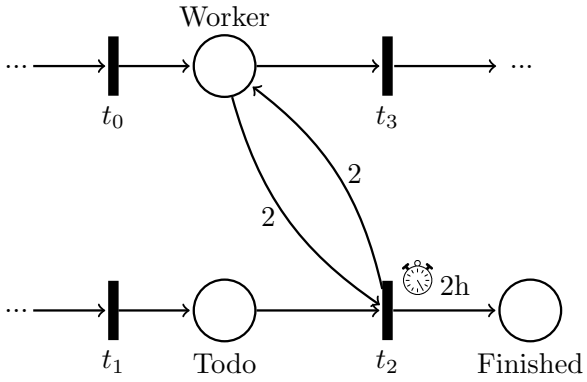


Figure 1: Example of a timed Petri net

The functionality of the library is split into two major parts. The first part contains methods for monitoring place-specific properties, and the second one such for monitoring the token flow through the network. Common to both parts are macros for modeling the network, or sections of it.

The overall idea of the library is that the information which transitions are firing at a certain time in the Petri net are available as inputs of the TeSSLa specification. Therefore, each relevant transition in the net has a corresponding input stream in the specification. The events of this stream carry information on how often the transition is fired at the particular timestamp. Places are modeled through corresponding macros described below. Each place-macro makes use of all the input streams corresponding to its incoming and outgoing transitions: through this construction, the place-macro essentially becomes a monitor encapsulating the place in the net, and the output-streams of these macros can be connected in the TeSSLa specification to implement local or global observers. The translations of the Petri net into this part of the specification follow a fixed scheme and can be automated. Fig. 2 shows a TeSSLa specification for the scenario from Fig. 1, which will be described below.

```
1  import PetriNet.NonTokenTracking.Places
2  import PetriNet.NonTokenTracking.Transitions
3
4  in t0: Events[Int]
5  in t1: Events[Int]
6  in t2: Events[Int]
7  in t3: Events[Int]
8
9  def Worker = FIFOPlace(addTransitions(t0, delayTokens(2 * t2, 2h)),
10                         addTransitions(2 * t2, t3), 0, None[Int])
11 def Todo = FIFOPlace(t1, t2, 0, None[Int])
12 def timeCrit = Todo.maxWaitTimeT(time(period(10min))) > 5h
13 def error = timeCrit && Worker.tokenCnt >= 2
14 out error
```

**Figure 2:** Usage of the TeSSLa Petri net library, for the scenario depicted in Fig. 1

Arcs of the network are represented as intermediate streams of custom type `Token`. Places are modeled concretely by macros `FIFOPlace` or `LIFOPlace` which internally maintain the tokens given to the place as queue or stack, depending on the policy which is suitable for the place. Further macros (`newTokens`, `mergeTokens`, `convertTokens` and `delayTokens`) allow to implement the behavior of (timed) transitions to pass tokens between the places. They do so by receiving streams of individual tokens from the places and recombining them to input streams of other places. This way, the library allows observations about the specific token flow in the network, even if this information is not directly provided by the network or the corresponding real-world application.

For monitoring of place-specific properties, the library provides functions to receive the number of tokens in a place and to detect under- or overflows of the place, if a maximum token capacity is given, as well as functions to measure the minimum, maximum and average dwell time of tokens in a place. Concerning flow-specific properties, the `Token` type which is used in the library allows to have IDs and clock values attached to specific tokens. If a token passes a certain transition, it may get a clock value attached which stores the timestamp of when the passing happened. If new tokens are spawned, they inherit the clock value from their parents. Finally, the clock values of arriving tokens can be used in other transitions in the network to measure the time that tokens needed for passing through the network. With this information again the minimum, average and maximum time of this token flow can be calculated and used to reveal quantitative parameters of the network or to check temporal constraints.

If the complexity of the network does not allow the simulation of every individual token because of memory and computational restrictions, our TeSSLa library provides a sub-module named `PetriNet.NonTokenTracking`. The `PetriNet.NonTokenTracking` sub-module, which is used in Fig. 2, is a variant of the mentioned macros in `PetriNet.TokenTracking` where only the number of tokens passing through the network and the time they remain in the places is recorded but not the individual tokens. This version also contains functions which are providing information about the number of tokens in certain places and e.g. their waiting time but is not capable of tracking and attaching clock values to individual tokens.

In this case the number of tokens is calculated by a multiplication between the weights of the transitions and the input streams which indicate how often these transitions are triggered. Finally, the macro `addTransitions` asynchronously sums up the token numbers from the different transitions, e.g., in the case of `Worker`, which receives tokens from `t0` and `t2`. The timed behavior of transition `t2` (holding tokens back for two hours) is modeled by a call to `delayTokens` which retards an event from stream `2 * t2` by two hours. Note that in TeSSLa it is possible to directly use time units in the specification. As third and fourth parameters, the `FIFOPlace` accepts the number of initial tokens in the place (0 in our example) and optionally (not used here) a maximal token capacity of the place, which can be used to monitor that the place is not overflowing.

The macro `FIFOPlace` returns a structure of several streams which bear information about the current state of the modeled place. One of them is the stream `maxWaitTimeT(t)` which contains the amount of time the longest-waiting token has remained in the place up to the timestamp $t$, passed as argument. In line 12, we define a Boolean stream `timeCrit` where we check if a token has waited more than five hours in the `Todo` place. We evaluate this stream on a ten-minute basis by triggering it with `period(10min)`. Finally, the output stream `error` raises a *true* event whenever the waiting time in the `Todo` place is critical and more than two tokens in `Worker` are available, i.e., workers are not immediately taking a job that is due.

## 4. Case study: Pathology Laboratory

We have applied our approach to a real-world use case at *Haukeland University Hospital* (HUS) in Bergen, Norway, who are currently investigating and implementing tools for *digital pathology*. A major part of that topic is chiefly concerned with machine learning for digital image analysis [13]. However, an equally important role is played by tools that support, monitor, and steer the production workflow of the digital pathology images, which happens in the laboratory [14]. Our case study is based on simulation and analysis of that workflow.

**The Histology Workflow.** Pathology is the study of the causes and effects of diseases. *Histology* is one of the major sub-disciplines of pathology and describes the study of tissue specimens under a microscope. With over 55 000 analyzed cases every year[1], the histology division at the pathology laboratory at Haukeland Hospital handles the biggest share of incoming pathology specimen and is the focus of the remainder of this case study. Before a pathologist can analyze a tissue specimen under a

[1]Numbers are from 2022. There is an expected case growth rate of 1.5% every year.

microscope, or nowadays, with the advent of digitization in this field, on a screen [13], the specimen has to undergo an elaborate "*preparation process*" in the pathology laboratory. We will sketch this workflow and its stages using a simplified presentation, which is depicted in Fig. 4 in the Appendix and based on [14].

**The Case Study's Goals.** A central goal at the hospital is to develop means that help optimizing the laboratory *throughput rate* by intelligently utilizing the limited resources. Due to the high degree of manual activities, variations among the numbers and complexity of incoming specimens, and also variations in the characteristics of the workflow resources (worker availability, worker's experience and skill level, etc.), this poses a challenging task. The approach proposed in [14] is to apply *process mining* to create a workflow model from existing event logs. The resulting workflow model can then be used for *simulation* purposes, which produce *statistics* and *performance measures*. The latter can be utilized by a *planner* or a similar tool to assess the *utility* of concrete actions in order to optimize the workflow. Actions in this scenario are changes to the workflow configuration, e.g., how many worker resources are available or how specimens are routed to which workflow branch (i.e., lab).

Out of the available workflow modeling formalisms, we have chosen PNs to model the histology workflow since they can faithfully model the "*pathology tokens*" (cases, cassettes, blocks, slides) flowing to the different processing steps of the lab simultaneously. However, modeling the "real" pathology lab at Haukeland with all its sub-laboratories and optional processing steps results in a much more complicated version than the one presented in Fig. 4. In this case study, we restrict to the histology lab in order to analyze the resulting *performance*. Concretely, we will be interested in the following key performance indicators (KPIs): (i) What is the *minimum*, *maximum* and *average* time that a case spends in the pathology lab from registration to final report (the case turnaround time)? (ii) Is the number of cases waiting on grossing, the number of blocks waiting on sectioning, or the number of slides waiting on staining growing indefinitely or not (i.e., is the current case arrival rate sustainable for the currently available resource configuration)?

**Implementation of the pathology workflow as TPN.** The files for the case study project (i.e., a detailed Petri net model, the TeSSLa specification and the output report) are located together with the TeSSLa library for TPNs in the git repository [12] and included in the Appendix. The transitions of the Petri net are annotated with the average execution time per activity. Those times were calculated by our partners at the hospital, who had access to the historical event logs produced by the information system in the lab. By applying process mining [14], durations for each activity were derived from the raw event data. By combining results of the statistical aggregation functions *mean* and *median* for each activity, the "average" time could be calculated in order to produce meaningful simulation results in this first iteration. Shared resources include their initial marking, and the weights on the arcs represent the number of tokens they produce or consume. For instance, the grossing stage creates on average three cassettes per specimen, which are later processed in batches of one hundred blocks by the processing machine. Machines generally work on batches of tokens, while a technician handles one token at a time. Also, the lab technicians who perform the various manual activities in the lab are modeled as one shared resource pool.

The computation of the KPIs of our case study relies on the simulations of the TPN in Titan [15]. The TeSSLa interpreter receives the simulation trace with the list of fired transitions sorted by timestamp, and the TeSSLa code uses the new TeSSLa library for analyzing TPNs. The first part of the TeSSLa code defines the input streams, which corresponds to the transitions of the Petri net to monitor (the names match with the names of the transitions of the TPN model in Titan). The places of the TPN are instantiated as places with FIFO policy in the TeSSLa library, so that the TeSSLa interpreter can track the flow of particular tokens through the net and compute the complete execution time since the beginning (i.e., the grossing stage) to the end (inspecting the stained slides under the microscope).

**Case study results.** TeSSLa returns a report with the selected KPIs *min*, *max* and *average* time a case spends in the pathology lab. Given the arrival rate of incoming specimens, the experiments prove that the current configuration of the Haukeland University Hospital is properly scaled in terms of processing machines and lab technicians because the diagnostics are completed within a working day. For instance, cassettes are processed when they reach a batch size of 115 elements, the queue of blocks

to be checked never exceeds 110 elements and the stained slides are inspected by the pathologists, every time they reach groups of 20 items. At instant 84 248, TeSSLa reports that the *avgRunTime* is 80 737 seconds (one day has 86 400 seconds). Besides the fluctuation of the performance metrics through a working day, the TeSSLa output also reports information about some places. For instance, it shows that the cassettes periodically drain completely over time. The results are computed running a simulation with 10 000 execution steps, which comprises around 6 working days in the pathology department. The simulation and runtime monitoring take less than 9 minutes on a conventional PC.

## 5. Related Work

Time information is defined in various formats for Petri nets along the literature [3] (e.g., discrete vs continuous semantics), and serves us to sort the events in simulation traces. Influenced by the specific application fields, the distinguishing characteristics of time mechanisms are whether the duration of events is modeled by deterministic variables or random variables and whether the time is associated with Petri net places [4], transitions [5] or tokens [6]. In this paper, we align with the definition of time proposed in Colored Petri Nets and its modeling framework CPN Tools [7]. CPN Tools already supports the performance analysis of complex systems [16] and provides native monitors to be written in SML. However, in our opinion, the usage of a dedicated Runtime Verification formalism together with its framework is a benefit when it comes to the specification of complex (correctness) properties. Nevertheless, we leave a thorough analysis of these approaches as future work. In addition, our TeSSLa library can complement other PN tools that do not count on such monitoring formalism and adapt the semantics to the traces that other PN tools generate.

TeSSLa belongs to the family of stream runtime verification languages. These languages have been pioneered by LOLA in 2005 [17] which is a formalism for synchronous streams, while TeSSLa also allows for asynchronous ones, i.e., such where the stream events do not have to be located on a fixed time grid, but can appear at arbitrary timestamps over a dense time domain. Further prominent SRV formalisms include the languages RTLOLA [18, 19] and Striver [20]. These languages can be considered to be located between TeSSLa and LOLA, as they allow for restricted handling of asynchronous streams and can therefore provide guarantees about monitoring resources. The TeSSLa representation of a Petri net enables us to define local (at a single transition/place) or global (connecting multiple events from transitions/places) properties on the flow of tokens. The *tracking* Petri net module in TeSSLa tracks token-identities and their timestamps in a *map*-data structure. We note that this corresponds to using Colored Petri net tokens that explicitly accumulate this data in a per-token history through manipulation of each transition. Our monitoring-approach does not require any modification of the inscriptions on the net, as any events are collected at runtime from the simulator. The global observation capabilities through TeSSLa would also need to be modeled through dedicated modification of the net structure.

## 6. Conclusions

In this paper, we have presented an approach for the analysis of timed Petri nets using runtime verification techniques. In particular, we rely on the stream-based processing language TeSSLa for the definition of the Petri net structure and the properties to check. We have provided a library for directly representing the Petri net structure in TeSSLa, extended by capabilities to track the execution workflow. The TeSSLa specification is driven by the transition-events from a run of the Petri net simulation provided by an external tool. It is also possible to run the TeSSLa analysis on recorded traces, hence there is no need to rerun e.g., CPN Tools while prototyping the specification. On top of the structural scaffolding in TeSSLa, users can add monitors that observe the flow of tokens through the network and compute domain-specific statistics.

As future work, we will continue improving the modeling and analysis capabilities of Petri nets in stream languages such as TeSSLa. For instance, we will consider colors (i.e., data types) for treating the tokens differently in Petri nets. Besides, we will include priorities and probabilities in the arcs

and transitions of the Petri net. Additionally, we will improve the user experience and the usability. For instance, we will provide templates for predefined useful dynamic properties the end user desires to analyze (e.g., usage of resources or system throughput), so that the end user only have to choose the right macro. Petri net modeling tools such as Titan could automatically translate the places and transitions in the graphical model into the corresponding elements of the TeSSLa library, so that users only need to define the desired properties in the TeSSLa specification.

## 7. Acknowledgments

This paper was supported by the Spanish Ministry of Science and Innovation under project AwESOME (PID2021-122215NB-C31).

## References

[1] C. A. Petri, Kommunikation mit Automaten, Ph.D. thesis, Universität Hamburg, Germany, 1962.
[2] T. Murata, Petri Nets: Properties, Analysis and Applications, Proc. IEEE 77 (1989) 541–580.
[3] A. Cerone, A. Maggiolo-Schettini, Time-based expressivity of time Petri nets for system specification, Theoretical Computer Science 216 (1999) 1–53. doi:https://doi.org/10.1016/S0304-3975(98)00008-5.
[4] J. Sifakis, Use of Petri nets for Performance Evaluation, Acta Cybernetica 4 (1979) 185–202.
[5] C. Ramchandani, Analysis of Asynchronous Concurrent Systems by Timed Petri Nets, Ph.D. thesis, Massachusetts Institute of Technology (MIT), 1974.
[6] P. M. Merlin, D. J. Farber, Recoverability of Communication Protocols: Implications of a Theoretical Study, IEEE Trans. Commun. 24 (1976) 1036–1043.
[7] K. Jensen, L. M. Kristensen, L. Wells, Coloured Petri Nets and CPN Tools for Modelling and Validation of Concurrent Systems, Int. J. Softw. Tools Technol. Transf. 9 (2007) 213–254. See also https://cpntools.org/.
[8] G. Gardey, O. H. Roux, O. F. Roux, State space computation and analysis of time petri nets, Theory Pract. Log. Program. 6 (2006) 301–320. URL: https://doi.org/10.1017/S147106840600264X. doi:10.1017/S147106840600264X.
[9] M. Leucker, C. Schallhart, A brief account of runtime verification, J. Log. Algebraic Methods Program. 78 (2009) 293–303. URL: https://doi.org/10.1016/j.jlap.2008.08.004. doi:10.1016/J.JLAP.2008.08.004.
[10] L. Convent, S. Hungerecker, M. Leucker, T. Scheffel, M. Schmitz, D. Thoma, TeSSLa: Temporal stream-based specification language, in: T. Massoni, M. R. Mousavi (Eds.), Formal Methods: Foundations and Applications - 21st Brazilian Symposium, SBMF 2018, volume 11254 of *Lecture Notest in Computer Science*, Springer, 2018, pp. 144–162. doi:10.1007/978-3-030-03044-5_10.
[11] H. Kallwies, M. Leucker, M. Schmitz, A. Schulz, D. Thoma, A. Weiss, TeSSLa - An Ecosystem for Runtime Verification, in: T. Dang, V. Stolz (Eds.), Runtime Verification (RV 2022), volume 13498 of *Lecture Notest in Computer Science*, Springer, 2022, pp. 314–324. doi:10.1007/978-3-031-17196-3_20.
[12] TeSSLa-TPN Library, 2024. https://gitlab.isp.uni-luebeck.de/public_repos/pn_tessla_artifact/.
[13] L. Pantanowitz, A. Sharma, A. B. Carter, T. Kurc, A. Sussman, J. Saltz, Twenty Years of Digital Pathology: An Overview of the Road Travelled, What is on the Horizon, and the Emergence of Vendor-Neutral Archives, Journal of Pathology Informatics 9 (2018) 40. doi:10.4103/jpi.jpi_69_18.
[14] P. Stünkel, S. Leh, F. Leh, Process Data Science for Workflow Optimization in Digital Pathology: A status report, in: Y. Lamo, A. Rutle (Eds.), Proceedings of The International Health Data Workshop co-located with 10th International Conference on Petrinets (Petri Nets 2022), Bergen, Norway,

June 26th-27th, 2022, volume 3264 of *CEUR Workshop Proceedings*, CEUR-WS.org, Bergen, Norway, 2022. URL: https://ceur-ws.org/Vol-3264/HEDA22_paper_10.pdf.

[15] E. Gómez-Martínez, E. Guerra, J. de Lara, A. Garmendia, Lifted structural invariant analysis of Petri net product lines, J. Log. Algebraic Methods Program. 130 (2023) 100824.

[16] L. Wells, Performance analysis using Coloured Petri Nets, in: 10th International Workshop on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS 2002), IEEE Computer Society, 2002, p. 217. doi:10.1109/MASCOT.2002.1167080.

[17] B. D'Angelo, S. Sankaranarayanan, C. Sánchez, W. Robinson, B. Finkbeiner, H. B. Sipma, S. Mehrotra, Z. Manna, LOLA: runtime monitoring of synchronous systems, in: 12th International Symposium on Temporal Representation and Reasoning (TIME 2005), 23-25 June 2005, Burlington, Vermont, USA, IEEE Computer Society, 2005, pp. 166–174. URL: https://doi.org/10.1109/TIME.2005.26. doi:10.1109/TIME.2005.26.

[18] P. Faymonville, B. Finkbeiner, M. Schledjewski, M. Schwenger, M. Stenger, L. Tentrup, H. Torfah, Streamlab: Stream-based monitoring of cyber-physical systems, in: I. Dillig, S. Tasiran (Eds.), Computer Aided Verification - 31st International Conference, CAV 2019, New York City, NY, USA, July 15-18, 2019, Proceedings, Part I, volume 11561 of *Lecture Notes in Computer Science*, Springer, 2019, pp. 421–431. URL: https://doi.org/10.1007/978-3-030-25540-4_24. doi:10.1007/978-3-030-25540-4_24.

[19] P. Faymonville, B. Finkbeiner, M. Schwenger, H. Torfah, Real-time stream-based monitoring, CoRR abs/1711.03829 (2017). URL: http://arxiv.org/abs/1711.03829. arXiv:1711.03829.

[20] F. Gorostiaga, C. Sánchez, Stream runtime verification of real-time event streams with the Striver language, Int. J. Softw. Tools Technol. Transf. 23 (2021) 157–183. URL: https://doi.org/10.1007/s10009-021-00605-3. doi:10.1007/S10009-021-00605-3.

## TeSSLa Specification for Pathology

The following files are available in the public repository and included here for reference.

```
1  import PetriNet.TokenTracking
2
3  in Grossing: Events[Int]
4  in Processing: Events[Int]
5  in Decalcination_t: Events[Int]
6  in Embedding_automatic: Events[Int]
7  in Embedding_manual: Events[Int]
8  in Case_Completeness_Checked: Events[Int]
9  in Sectioning: Events[Int]
10 in Staining_automatic: Events[Int]
11 in Staining_manual: Events[Int]
12 in Scanning: Events[Int]
13
14 def t1 = Transitions.delayTokens(Transitions.newTokens(3, Grossing), 474)
15 def tstart = Transitions.startTimeMeasure(1, t1)
16
17 def CassettesP = Places.FIFOPlace(tstart, 115 * Processing,
18                                   List.empty[Token], None[Int])
19
20 def Processed_CassettesP = Places.FIFOPlace(
21         Transitions.delayTokens(Transitions.convertTokens(CassettesP.tokenOut, 115 *
               Processing), 52200),
22         Transitions.addTransitions(20 * Embedding_automatic,Embedding_manual),
23         List.empty[Token], None[Int])
24
25 def BlocksP = Places.FIFOPlace(Transitions.mergeTokens(
26         Transitions.delayTokens(Transitions.convertTokens(Processed_CassettesP.tokenOut, 20
               * Embedding_automatic), 1041),
27         Transitions.delayTokens(Transitions.convertTokens(Processed_CassettesP.tokenOut,
               Embedding_manual), 135)),
28         10 * Case_Completeness_Checked, List.empty[Token], None[Int])
29
30 def Checked_BlocksP = Places.FIFOPlace(
```

```
31          Transitions.delayTokens(Transitions.convertTokens(BlocksP.tokenOut, 10 *
                Case_Completeness_Checked), 30),
32          Sectioning, List.empty[Token], None[Int])
33
34  def SlidesP = Places.FIFOPlace(
35          Transitions.delayTokens(Transitions.convertTokens(Checked_BlocksP.tokenOut,
                Sectioning), 139),
36          Transitions.addTransitions(20 * Staining_automatic, Staining_manual),
37          List.empty[Token], None[Int])
38
39  def Stained_SlidesP = Places.FIFOPlace(Transitions.mergeTokens(
40          Transitions.delayTokens(Transitions.convertTokens(SlidesP.tokenOut, 20 *
                Staining_automatic), 3600),
41          Transitions.delayTokens(Transitions.convertTokens(SlidesP.tokenOut, Staining_manual
                ), 3600)),
42          20 * Scanning, List.empty[Token], None[Int])
43
44  @VisDots
45  out CassettesP.tokenCnt
46  @VisDots
47  out Processed_CassettesP.tokenCnt
48  @VisDots
49  out BlocksP.tokenCnt
50  @VisDots
51  out Checked_BlocksP.tokenCnt
52  @VisDots
53  out SlidesP.tokenCnt
54  @VisDots
55  out Stained_SlidesP.tokenCnt
56
57  out Transitions.maxTime(Transitions.stopTimeMeasure(1, Stained_SlidesP.tokenOut).times) as
        maxRunTime
58  out Transitions.minTime(Transitions.stopTimeMeasure(1, Stained_SlidesP.tokenOut).times) as
        minRunTime
59  out Transitions.avgTime(Transitions.stopTimeMeasure(1, Stained_SlidesP.tokenOut).times) as
        avgRunTime
```

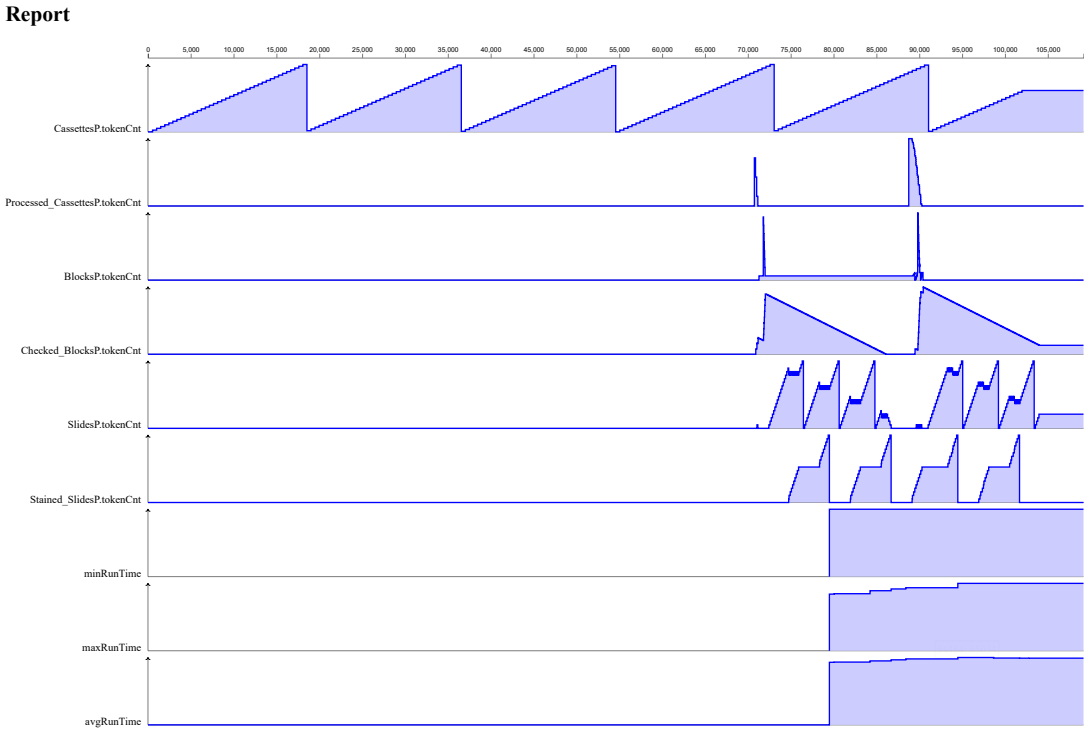


**Figure 3:** TeSSLa report including execution times and number of tokens per place

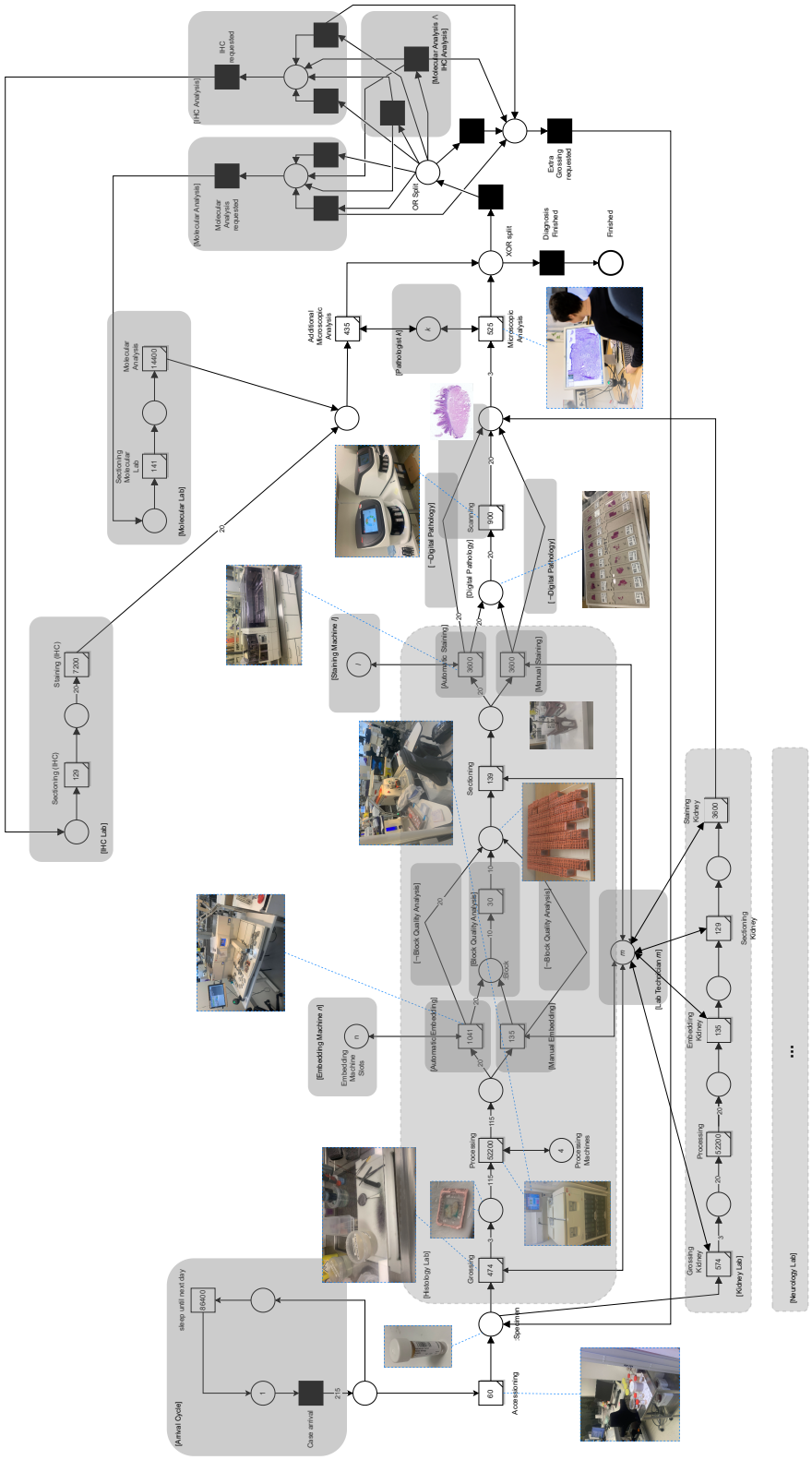

**Figure 4:** Petri net for the Pathology workflow at HUS