
**Algorithmen und Werkzeuge für Petrinetze 2024**
**an der DTU in Kopenhagen**

# AWPN 2024
# Workshop Proceedings

**https://fg-petrinetze.gi.de**

Eds.: Robert Lorenz, Robin Bergenthum, Patrizia Schalk


FACHGRUPPE
PETRINETZE UND
VERWANDTE SYSTEMMODELLE

# Table of Contents

# Partially Ordered Event Logs and Concurrency Oracles

Sabine Folz-Weinstein[1], Robin Bergenthum[2] and Christian Beecks[1]

[1] Chair of Data Science, FernUniversität in Hagen, Hagen, Germany
[2] Faculty for Mathematics and Computer Science, FernUniversität in Hagen, Hagen, Germany
sabine.folz-weinstein@fernuni-hagen.de

Process mining analyzes recorded behavior which is represented by event logs. Traditionally, an event log is a multiset of traces, where each trace is a totally ordered sequence of activities [2, 3, 5, 11]. The order of activities in a trace is typically based on their temporal occurrences. However, this temporal order does not necessarily reflect the actual causal order, as causally unrelated activities may also be ordered in time [4]. Yet, the order of activities is crucial for producing high-quality outputs with process mining algorithms. Consequently, there is a growing amount of work suggesting the usage of event logs in which the activities are partially ordered [7, 8, 9, 13].

Using partial orders, we can explicitly express both uncertainty and concurrency [12, 13, 14]. This allows us to remove total order relations for which there is insufficient or unreliable evidence, such as those based on inaccurate or incomparable timestamp information due to manual input or varying time granularities. Additionally, partial orders enable us to model specific properties of the underlying activities, like non-zero duration and timely overlap, which is not possible under the assumption of a total order. Additionally, since a partially ordered trace can "summarize" the behavior of multiple sequential traces, partially ordered logs often provide a more compact representation of the recorded behavior. This compactification can improve algorithm runtimes and reduce space complexity. Finally, partially ordered logs and traces can help identify "use cases" of a process, facilitating (or enabling) user-interactive exploratory process mining.

Up to this point, there is no common understanding of the terms "partially ordered log/trace" within the process mining community, and they are used with very different interpretations, strongly connected to the above-named reasons for use. For example, there exist various different semantics of concurrency. The first semantic dimension relates to certainty or uncertainty regarding the non-order of activities [13]. If we have no evidence for a specific causal order between two or more activities, this could be interpreted as indication that the activities are indeed concurrent, meaning the non-order itself provides causal relation information. Alternatively, it could be interpreted as missing information, suggesting that we simply do not know the order - or whether an order should exist at all. This directly leads to the semantical relation between a partial order and its sequentializations with respect to the represented behavior: Does a process model allow the execution of a partially ordered trace only if it allows all its sequentializations [7, 9, 10], or if it allows any one of its sequentializations [1, 15]? Furthermore, we must distinguish if we are interested in the fact if a model can or cannot execute certain behavior or if we are interested in counting percentages or probabilities, e.g. how many of the sequential traces represented by a partially ordered

trace, or which fraction of a partially ordered log can be executed. Another dimension is that activities can either be considered atomic or to have a certain duration, which leads to different interpretations of concurrency with respect to an interleaving or an overlapping of the activities [13]. Furthermore, there is the dimensionality of scope: concurrency which is detected within one or several traces within an event log can be interpreted to either be valid throughout the log, i.e., on an activity level, or to be valid only at the point within a trace where it is detected, i.e. on an event level [6].

We must consider all these different dimensions when discussing, evaluating, choosing and using methods and algorithms based on partially ordered input and when using concurrency oracles. Obviously, there are different needs and considerations depending on the process mining discipline, especially within discovery and conformance checking. Thus, it is vital to precisely define and explain the semantical interpretation of the term "partially ordered log" when used in an algorithm.

In this work, we define a partially ordered trace as a pair of a trace of length $k$ and a partial order $\prec$ over the $k$ positions of the trace. We define a partially ordered event log as a multiset of partially ordered traces. We discuss various heuristics to add a partial order on top of a sequential event log, called concurrency oracles. We implemented a configurable concurrency oracle ("CCO") tool as pm4py plugin [16]. In its current version, it supports "alpha" and lifecycle-based concurrency detection on event (log-wise) or activity (trace-wise) level, exporting the result as .xes-file with additional partial order information per trace and event. It allows to either retain all cases in the log, to reduce the log to one representative per sequential trace, or to one representative per partially ordered trace and, thus, supports different levels of compactification of the log, depending on the semantical relation between sequential and partially ordered trace required in the specific target algorithm and use case.

We evaluate the generation of partially ordered logs with our "CCO" tool using different combinations of parameters and several well-known BPI challenge logs, which illustrate the importance of a configurable concurrency oracle tool and a clear distinction of the purpose of the generated partially ordered log. If we have, for example, an event log with atomic activities like the BPI 2019 log which models a purchase order process, we have activities which are very likely to be concurrent in their execution throughout the log like the recording of the reception of goods or services and the recording of the reception of an invoice. Assuming that a partial order represents all its sequentializations, we may be interested in a compactification of the log. In this case, we observe that the 251.734 cases of 11.973 sequential traces fold together to 5.851 partially ordered traces, i.e., about 50% reduction of variants. In this example, it also makes sense to keep only one trace representative and, thus, have a significantly smaller log for further analysis. However, if we have an event log with lifecycle information like the BPI 2017 log modeling a loan application process, we may mainly be interested in adding concurrency information on event level and keep all trace variants, e.g. for conformance checking. Here, we observe that the 31.509 cases of 6.844 sequential traces fold together to 6.566 partially ordered traces due to isomorphy, i.e., almost no reduction in size, but a qualitative enhancement. The CCO allows us to experiment with different parameter combinations and pre-analyze the detected concurrency, e.g. concerning differences, specific patterns or validity.

## References


1. van der Aa, H., Leopold, H., Weidlich, M.: Partial order resolution of event logs for process conformance checking. In: Decision Support Systems, Volume 136, Article no. 113347, Elsevier (2020).
2. van der Aalst, W. M. P.: Process Mining: Discovery, Conformance and Enhancement of Business Processes. Springer (2011).
3. van der Aalst, W. M. P.: Process Mining: Data Science in Action (https://doi.org/10.1007/978-3-662-49851-4). Springer (2016).
4. van der Aalst, W. M. P., Santos, L. F. R.: May I take your order? On the Interplay Between Time and Order in Process Mining. In: Business process management workshops, BPM 2021, pp. 99-110. Springer (2021).
5. van der Aalst, W. M. P., Carmona, J.: Process Mining Handbook. Springer (2022).
6. Armas-Cervantes, A., Dumas, M., La Rosa, M., Maaradji, A.: Local Concurrency Detection in Business Process Event Logs. In: ACM Transactions on Internet Technology, vol. 19, no. 1, pp. 1 – 23 (2019).
7. Bergenthum, R.: Prime Miner - Process Discovery using Prime Event Structures. ICPM 2019, pp. 41 – 48 (2019).
8. Dumas, M., García-Bañuelos, L.: Process Mining Reloaded: Event Structures as a Unified Representation of Process Models and Event Logs. PETRI NETS 2015, LNCS 9115, pp. 33 – 48. Springer (2015).
9. Folz-Weinstein, S., Bergenthum, R., Desel, J., Kovár, J.: ILP² Miner – Process Discovery for Partially Ordered Event Logs Using Integer Linear Programming. In: Gomes, L., Lorenz, R. (eds.) Application and Theory of Petri Nets and Concurrency, PETRI NETS 2023, LNCS 13929, pp. 59 - 76, Springer (2023).
10. Grabowski, J.: On Partial Languages. In: Fundamenta Informaticae, vol. 4, no. 2, pp. 427 – 498. IOS Press (1981).
11. IEEE Task Force on Process Mining: Process Mining Manifesto. Business Process Management Workshops, Lecture Notes in Business Inormation Processing, vol. 99, pp. 169 – 194. Springer (2012).
12. Janicki, R., Koutny, M.: Structure of Concurrency. In: Theoretical Computer Science 112, no. 1, pp. 5 – 52. Elsevier (1993).
13. Leemans, S. J. J., van Zelst, S. J., Lu, X.: Partial-order-based process mining: a survey and outlook. Knowledge and Information Systems, vol. 65, pp. 1 – 29. Springer (2023).
14. Pratt, V.: Modelling Concurrency with Partial Orders. In: International Journal of Parallel Programming 15, pp. 33 – 71. (1986).
15. Sommers, D., Sidorova, N., van Dongen, B.: Conformance Checking with Model Projections - Rethinking Log-Model Alignments for Processes with Interacting Objects. In Application and Theory of Petri Nets and Concurrency, PETRI NETS 2024, pp. 61-82. Springer (2024).
16. Configurable Concurrency Oracle Tool repository, https://github.com/sabinefw/ConfigurableConcurrencyOracleTool , last access 2024/08/07

# Exploratory Process Discovery for Petri Nets

Jakub Kovář[1][0000−0002−7775−3698] and Robin Bergenthum[2][0000−0003−0464−8843]

[1] Lehrgebiet Programmiersysteme, FernUniversität in Hagen, Germany
[2] Fakultät für Mathematik und Informatik, FernUniversität in Hagen, Germany

The goal of process discovery is to automatically build a process model from an event log [1,3]. In applications, business processes become more complex and process discovery often generates spaghetti-like models [2]. This is why exploratory discovery enables a user to interactively filter the input and thus, alter the generated discovery result.

Existing interactive discovery approaches generate either directly-follows models [8], or process trees [9]. In these approaches, we order traces of an event log by their frequencies and move a slider to add or remove traces to the model. Thus, we can "zoom-in" and "zoom-out" of frequent behaviour. This feature is so helpful that almost every commercial process mining tool provides a similar feature to visualise an event log.

Using interactive discovery, we can discover both directly-follows models and process trees very efficiently and, thus, organise and replay different cases of a workflow. But if we only focus on the directly-follows relation, we generate models with a high level of generalisation and a low level of precision. We argue, in an interactive and exploratory setting, where we hand-pick a set of traces, the resulting model should be precise. If we want to dig deep and analyse and explore the control flow of a business process faithfully, we must use Petri net like models.

In this work, we outline the key sub-tasks of exploratory discovery for Petri nets. Such discovery has not been introduced yet, because it is hard to modify the behaviour of a Petri net interactively. As a proof-of-concept, we present a possible implementation, the 🦓 Miner for Petri nets, available online at www.fernuni-hagen.de/ilovepetrinets/zebra. We discuss the different sub-tasks of exploratory discovery and compare our implementation to the state-of-the-art exploratory discovery Directly-Follows Visual Miner [8]. The steps of interactive discovery for Petri nets are as follows.

In a first step of exploratory discovery, we *select* a subset of traces and *discover* our initial model. For a directly-follows based approach, this operation is inexpensive. We argue, if we *select* and hand-pick a sub-set of traces in an exploratory setting, we are looking for a precise approach. In regular discovery, being too precise and generating over-fitting models is a major disadvantage. If we only consider a small subset of traces of some event log, the high run-time of highly precise region-based synthesis approaches may not be so problematic.

In a second step, we *validate* if the process model is a precise representation of the selected set of traces. A synthesised Petri net model is the best upper approximation of the selected traces, therefore it may also contain traces of the input log that were not selected by the user. This weakens the relation between

the set of selected traces and the current model and can cause confusion. We replay all non-selected traces by firing, as presented in [4], in the synthesised model. We highlight the set of already enabled traces of the event log, as this provides important information about the quality of the current model.

In a third step, we *rate* the current selection of traces using the current process model as a reference. For directly-follows based models, we look for our preferred balance between fitness and simplicity of the resulting model. For Petri net models, we can additionally use structural and behavioural properties of the model to help us judge, how well the selected traces align. For example, a precise discovery algorithm will often produce unsound Petri net models. These unsound models provide very good feedback on the currently selected set of traces. They indicate that there may be other, not yet selected traces of the event log, which complete the current, incomplete model. If a precise algorithm produces a sound model, the set of selected traces fits together very well. Identifying these subsets of traces in an interactive setting provides an in-depth insight into the structure of the recorded behaviour.

The fourth and final step restarts the iterative exploratory procedure. We *adjust* the current selection of traces based on information from the previous step and *update* the current process model accordingly. Directly-follows based approaches can *update* the model very efficiently. For classical Petri net discovery approaches, if we change the set of input traces, we must restart the synthesis procedure. This adds a huge run-time cost and sabotages the interactivity of exploratory discovery. For this reason, the Miner implements *discover* and *update* using a new region-based discovery technique based on the token trail semantics for Petri nets [5,7]. We outline this Petri net region theory in [6]. Petri net regions can synthesise a Petri net model from a set of labelled nets. Roughly speaking, we can take two models and synthesise them into the most precise upper approximation of their combined behaviour. Furthermore, we store intermediate results to be able to remove traces later at no cost. This incremental synthesis uses smaller inputs and so has a more favourable run-time. Our preliminary experiments show it is possible to use this technique in an interactive setting.

Adjusting the selection of traces, together with the current model as a reference, we can find our preferred balance between fitness and simplicity of the resulting model. Furthermore, using an exploratory approach, we learn about frequent and infrequent behaviour recorded in the event log. We either "zoom-in" on a precise, hopefully readable model of only the frequent behaviour, or "zoom-out" to a more general, possibly more complex view. We want to generate precise models for a selected subset of traces, to uncover and explore the structure of the recorded behaviour. A perfect use case for Petri net models.

## References

1. Van der Aalst, W.M.P.: Process Mining. Springer Berlin, Heidelberg (2011)
2. Van der Aalst, W.M.P.: A practitioner's guide to process mining: Limitations of the directly-follows graph. Procedia Computer Science **164**, 321–328 (2019)
3. Van der Aalst, W.M.P., Carmona, J.: Process mining handbook. Springer Nature (2022)
4. Bergenthum, R.: Firing Partial Orders in a Petri Net. In: Proceedings of PETRI NETS 2021. pp. 399–419. Springer (2021)
5. Bergenthum, R., Folz-Weinstein, S., Kovář, J.: Token Trail Semantics - Modeling Behavior of Petri Nets with Labeled Petri Nets. In: Proceedings of PETRI NETS 2023. pp. 286–306. Springer (2023)
6. Bergenthum, R., Kovář, J.: A First Glimpse at Petri Net Regions. In: Proceedings of ATAED 2022. pp. 60–68. CEUR Workshop Proceedings 3167 (2022)
7. Kovář, J., Bergenthum, R.: Token Trail Semantics II - Petri Nets And Their Net Language. In: Proceedings of PETRI NETS 2024. pp. 175–196. Springer Nature Switzerland (2024)
8. Leemans, S.J., Poppe, E., Wynn, M.T.: Directly Follows-Based Process Mining: Exploration & a Case Study. In: Proceedings of ICPM 2019. pp. 25–32 (2019)
9. Schuster, D., van Zelst, S.J., Van der Aalst, W.M.P.: Cortado — an interactive tool for data-driven process discovery and modeling. In: Proceedings of PETRI NETS 2021. pp. 465–475. Springer (2021)

# State Machines (In Modular Petri Nets)

Sophie Wallner

Universität Rostock

A *modular Petri net* is a composition of individual Petri net systems that show both - individual internal behavior and synchronized interface behavior [4] [1] [3]. The main benefit of modular Petri nets is the parallel interleaved computation of the component's state spaces [3] to counteract the state explosion problem. As each component's state space is calculated individually, we can exploit the structure of the component for verification. By taking advantage of the heterogeneity of the components we can accelerate the state space generation and facilitate the verification process. In this work, we take a closer look at the Petri net structure *state machines* and show how to exploit its characteristics in the context of a modular Petri net.

**Definition 1 (State Machine).** *A* state machine *is a Petri net system* $[N, m_0]$*, where every transition has exactly one pre- and one post place, i.e.* $\forall t \in T : |\bullet t| = |\{p \mid (p,t) \in F\}| = 1$ *and* $|t\bullet| = |\{p \mid (t,p) \in F\}| = 1$.

A State Machine $[N, m_0]$ decays into SCCs $\{K_1, \ldots, K_n\}$ where $K_i \subseteq P$ for $i \in \{1, \ldots, n\}$ induce state machines themselves. SCCs only have ingoing and outgoing arcs from resp. to places: For SCC $K_i$ with $i \in \{1, \ldots, n\}$, we accumulate those transition as $\bullet K_i = \{t \mid (t,p) \in F_j, p \in K_i\}$ and $K_i\bullet = \{t \mid (p,t) \in F, p \in K_i\}$. SCC $K_i$ is a *predecessor* of SCC $K_{i'}$ if $(K_i\bullet) \cap (\bullet K_{i'}) \neq \emptyset$ for $i, i' \in \{1, \ldots, n\}$. With this we can introduce a *partial order* $\leq$ such that $K_i \leq K_{i'}$ if $K_i$ is a predecessor of $K_{i'}$, and $K_i \leq K_{i''}$ if $K_{i'} \leq K_{i''}$ and $K_i$ is a predecessor of $K_{i'}$. For SCC $K_i$ with $i \in \{1, \ldots, n\}$, the *backward cone* $R_i$ is the set of itself and all of its transitive predecessors, i.e. $R_i = \{K_{i'} \mid K_{i'} \leq K_i\}$. Backward cones of SCCs can be interleaved.

For a state machine, we can build the *reachability graph* as usual; let $R = [V, E]$ be the reachability graph of $[N, m_0]$ such that $V = RS(\{m_0\})$ and $(m, t, m') \in E$, iff $m \xrightarrow{t} m'$. Every marking $m$ can be abstracted to an *SCC-marking* $\mu$ that contains the number of tokens in each SCC of $[N, m_0]$.

**Definition 2 (SCC-marking).** *For marking* $m$*, we define the according* SCC-marking $\mu : m \times \{K_1, \ldots, K_n\} \to \mathbb{N}$ *such that* $\mu(m, K_i) = \sum_{p \in K_i} m(p)$ *for* $i \in \{1, \ldots, n\}$.

The following statements follow directly from the state machine structure:

1. The number of tokens in every reachable marking is constant, i.e. $\forall m \in V : \sum_{p \in P} m(p) = const$. No Token cannot be created or vanished because all weights are 1 and every transition has a constant in-out-ratio.

2. Let $m$ be a reachable marking and $\mu$ the according SCC-marking. Then, any marking $m'$ with $\mu(m', K_i) = \mu(m, K_i)$ for all $i \in \{1, \ldots, n\}$ is reachable from $m$. Within each $K_i$, every distribution of tokens over places is possible because of the strong connectedness. So, the reachability of $m'$ can be reduced to the reachability its corresponding SCC-marking $\mu$.
3. Let $m \in V$ be a reachable marking and $\mu$ the according SCC-marking. Then, Marking $m'$ with $\mu(m', K_i) = \sum_{K_l \in R_i} \mu(m, K_l)$ for all $i \in \{1, \ldots, n\}$ is reachable from $m$.

In General, reachability of markings depends on moving the right number of tokens to the right SCCs without using tokens multiple times. The reachability of marking $m'$ from marking $m$ can be expressed as a linear system of equations (ILP) $\Pi(m, m')$ that results in a solution vector $\boldsymbol{x} \in \mathbb{N}^{n \cdot n}$.

**Theorem 1 (Linear System of Equations For Reachability Analysis).** *Let $m$ be a reachable marking of $[N, m_0]$. Marking $m'$ is* reachable *from $m$, iff ILP $\Pi(m, m')$ has an integer solution $\boldsymbol{x} > 0$. The ILP has the form*

$$\mu(m, K_i) + \sum_{l=1}^{n} x_{li} - \sum_{l=1}^{n} x_{il} = \mu(m', K_i) \qquad \forall i \in \{1, \ldots, n\} \quad (1)$$

$$x_{li} = 0 \qquad \forall l, i \in \{1, \ldots, n\} \text{ if } K_l \notin R_i \quad (2)$$

We can use this for the verification of properties in state machines. *State predicates* serve as a base for the verification.

**Definition 3 (State Predicate, Visible Places, Coefficient-Mapping).** *Let $[N, m_0]$ be a Petri net system. An* atomic state predicate *$\varphi$ is a comparison of formal sum to an integer $a_1 m(p_1) \cdot \ldots \cdot a_k m(p_k) \leq a$ where $a_1, \ldots, a_k, a \in \mathbb{Z}$ and $k \leq |P|$. An atomic state predicate is satisfied in a marking, if the inequality is true. We call the occurring places $p_1, \ldots, p_k$ in $\varphi$* visible*, defining a set $vis(\varphi) \subseteq P$. Given an atomic state predicate $\varphi$, we can access the coefficient for visible place $p$ with the* coefficient mapping *$\alpha(p, \varphi) = a$. A* state predicate *is a conjunction $\Phi = \varphi^1 \wedge \ldots \wedge \varphi^g$, where $\varphi^h$ is an atomic state predicate for $h \in \{1, \ldots, g\}$. A state predicate is satisfied in a marking, if every atomic state predicate is true. We set $vis(\Phi) = \bigcup_{h \in \{1, \ldots, g\}} vis(\varphi^h)$.*

We want to reduce the verification of a state predicate as well to solving an ILP. Therefore, we need to introduce new variables: For each visible place $p \in vis(\Phi)$, we introduce variable $p$ that is part of the solution of the ILP. A solution assigning value $v \in \mathbb{N}$ to $p$ corresponds to a marking $m$ where $m(p) = v$. We build the ILP $\Pi(m_0, \Phi)$ with reference to Theorem. 1 with little adaptions: For every $K_i$ with $i \in \{1, \ldots, n\}$, $\mu(m_0, K_i) + \sum_{l=1}^{n} x_{li} - \sum_{l=1}^{n} x_{il}$ relaxes to the maximum bound for tokens on the visible places in $K_i$, and every atomic state predicate $\varphi^h$ needs to be satisfied in one marking.

**Theorem 2 (Satisfiability of State Predicates).** *Let $\Phi = \varphi^1 \wedge \ldots \wedge \varphi^g$ be a state predicate where $\varphi^h$ is an atomic state predicate for $h \in \{1, \ldots, g\}$. State*

*Predicate $\Phi$ is* satisfiable *in $[N, m_0]$, iff the following linear system of equations $\Pi(m_0, \Phi)$ has an integer solution $\boldsymbol{xp} > 0$:*

$$x_{li} = 0 \, \forall l, i \in \{1, \ldots, n\} \text{ if } K_l \notin R_i \tag{3}$$

$$\sum_{p \in K_i \cap vis(\Phi)} p \leq d_i + \sum_{l=1}^{n} x_{li} - \sum_{l=1}^{n_j} x_{il} \, \forall i \in \{1, \ldots, n\} \tag{4}$$

$$\sum_{i=1}^{n} \sum_{p \in K_i \cap vis(\varphi^h)} \alpha(p, \varphi^h) p \leq a^h \, \forall h \in \{1, \ldots, g\} \tag{5}$$

Furthermore, we can use the theory on state machines for verification of formulas of the temporal logic $CTL_1 \subset CTL$ only consists of state predicates that may be extended with **EF** to express the possibility of satisfaction or with **AG** to express the necessity of satisfaction. This is in style of the branching time logic $L_1$ from [2] for 1-safe Petri nets; an extension of propositional logic with a *possibility* operator $\Diamond$ and the *necessity* operator $\Box$. As their theoretical foundation is the same, $L_1$ and $CTL_1$ provide the same seven equivalence classes of combinations of the Operators:[], [**EF**], [**AG**], [**EFAG**], [**AGEF**], [**EFAGEF**], [**AGEFAG**]. The goal is to reduce verification of properties of those classes to solving an ILP as well. For example, with Theorem 2 we can verify properties of class [**EF**] directly; State Machine $N$ satisfies **EF**$\Phi$ if and only if $\Pi(m_0, \Phi)$ has a nontrivial integer solution. Our goal is to evaluate the verification of the other classes and find a systematic approach that we can embed in our verification portfolio [5].

Finally, we want to adapt this knowledge on state machines to modular Petri nets where some components are state machines. It is more likely that individual components are a state machine than that the entire system is. It would also be possible to modularize the Petri net system such that state machine instances are created. State machines lead to twofold advantages here: In the building process of the state spaces and in the verification of properties.

## References


1. Christensen, S., Petrucci, L.: Modular Analysis of Petri Nets. The Computer Journal **43**(3), 224–242 (Jan 2000)
2. Esparza, J., Heljanko, K.: Unfoldings: a partial-order approach to model checking. Springer Science & Business Media (2008)
3. Gaede, J., Wallner, S., Wolf, K.: Modular state spaces-a new perspective. In: International Conference on Applications and Theory of Petri Nets and Concurrency. pp. 312–332. Springer (2024)
4. Latvala, T., Mäkelä, M.: LTL Model Checking for Modular Petri Nets. In: Cortadella, J., Reisig, W. (eds.) Applications and Theory of Petri Nets 2004. pp. 298–311. Lecture Notes in Computer Science, Springer, Berlin, Heidelberg (2004)
5. Schmidt, K.: Lola a low level analyser. In: Application and Theory of Petri Nets 2000: 21st International Conference, ICATPN 2000 Aarhus, Denmark, June 26–30, 2000 Proceedings 21. pp. 465–474. Springer (2000)

# Utilization of Heterogeneity in Modular Reachability Analysis for Petri Nets

Marlene Schultz

University of Rostock, Schwaansche Str. 2, 18055 Rostock, Germany
marlene.schultz@uni-rostock.de

A *modular Petri net* is composed of multiple individual Petri nets, the modules. Modules are composed by fusing their *interface* transitions. The behavior of *internal* transitions is unrelated to other modules and is recorded in *local reachability graphs* for each module.

*Fusion vectors* describe which interface transitions of which modules need to synchronize for their firing. This behavior is recorded in the *synchronization graph*, linking the local reachability graphs together. For this, internal behavior between two firings of interface transitions is abstracted to form *segments*. A vertex in the synchronization graph collects segments for each module and records the changes from synchronized interface transitions. The *modular state space* [1] consists of the local reachability graphs and the synchronization graph.

The modular state space can be used to analyze the reachability of markings in Petri nets. However, constructing the local reachability graphs for each module can be highly inefficient because of the state explosion problem [3]. In this paper, we show how the construction of the local reachability graph can be circumvented if the given modules have a certain topology. We study for two different classes of Petri nets how their structural properties can be used to construct a *reduced* modular state space. The two classes are acyclic Petri nets and state machines.

In modular Petri nets with acyclic modules, the state equation is utilized to bypass the construction of the local reachability graphs. This is because in acyclic Petri nets, the solvability of the state equation is not only necessary but also sufficient for the reachability of a marking [2]. Information about the local reachability graph can therefore be obtained by solving a linear system of equations (ILP). As a result, all reachable markings of a module can be derived from initial markings using linear algebra. In order to construct the reduced modular state space, the interacting behavior of the modules must still be represented. Accordingly, all reachable markings that activate interface transitions are required for each segment in order to correctly reflect the behavior in the synchronization graph. We do not fully explore the segments, as we can deduce all reachable markings that activate an interface transition with the help of the state equation. Processing segments in local reachability graphs of acyclic modules with ILPs still leads to the complete and correct modular state space.

We develop a method for modular Petri nets with state machine modules that avoids the construction of local reachability graphs. A *state machine* is a Petri net system, where every transition has exactly one pre- and one post place. These structural restrictions make it possible to decompose the set of places of the net. The decomposition is performed using the strongly connected

components (SCC), the equivalence classes of strongly connected places. Within a single SCC, every distribution of present tokens is possible; therefore, only the number of tokens per component is sufficient for further processing of a marking. The backward cones of the individual components determine the way in which the tokens can move within the net. This can be expressed as an ILP from a given initial marking. As a result, statements can be made about the local state space based solely on the distribution of the tokens of an initial marker without having to construct it. Therefore, only the initial markings and the activating markings for the interface transitions need to be stored for each segment in order to reflect the behavior of the modular Petri net. The modular state space is still represented correctly and completely by processing the segments using these ILPs. The special case of strongly connected state machines allows us to use only the number of tokens to represent a segment. This is because for modules of this structure, any marking that has the same token count as the initial marking is reachable.

We show how the reachability of markings can be verified in the reduced modular state space of Petri nets with some modules of these considered net classes.

## References

1. Gaede, J., Wallner, S., Wolf, K.: Modular State Spaces - A New Perspective. In: Kristensen, L.M., van der Werf, J.M. (eds.) Application and Theory of Petri Nets and Concurrency. pp. 312–332. Springer Nature Switzerland, Cham (2024). https://doi.org/10.1007/978-3-031-61433-0_15
2. Kostin, A.E.: The novel algorithm for determining the reachability in acyclic petri nets **28**(2), 70–79 (1997). https://doi.org/10.1145/261342.261351
3. Valmari, A.: The state explosion problem. In: Reisig, W., Rozenberg, G. (eds.) Lectures on Petri Nets I: Basic Models: Advances in Petri Nets, pp. 429–528. Lecture Notes in Computer Science, Springer (1998). https://doi.org/10.1007/3-540-65306-6_21

# Symbolic Model Checking in the Modular State Space using Binary Decision Diagrams

Lukas Zech[0009−0003−7022−6021](✉)

University of Rostock, Schwaansche Str. 2, 18055 Rostock, Germany
lukas.zech@uni-rostock.de

**Abstract.** A *modular Petri net* is composed of multiple individual Petri nets, the modules. Modules are composed by fusing their *interface* transitions. The behavior of *internal* transitions is unrelated to other modules and is recorded in *local reachability graphs* for each module.
*Fusion vectors* describe which interface transitions of which modules need to synchronize for their firing. This behavior is recorded in the *synchronization graph*, linking the local reachability graphs together. For this, internal behavior between two firings of interface transitions is abstracted to form *segments*. A vertex in the synchronization graph collects segments for each module and records the changes from synchronized interface transitions. The *modular state space* [3] consists of the local reachability graphs and the synchronization graph.
Model Checking describes an exhaustive search through the state space to verify a property. For Symbolic Model Checking, the state space is encoded to combat the state explosion problem [5]. In this paper, we encode the local reachability graphs using Binary Decision Diagrams (BDD) [2,4]. To encode the reachability set of a module using BDDs according to [4], 1-safe modules are assumed. For each module, boolean variables are introduced for each place. A BDD encoding a reachability set then describes the characteristic function of the set as a directed acyclic graph in the form of a compressed decision tree. Similarly, BDDs can be used to encode segments in the local reachability graphs. Since BDDs encode *sets* and segments abstract away internal behavior, leaving only sets of markings, the encoding does not suffer great loss of information.
We study the reachability of markings in the composed Petri net that satisfy *state predicates*. A state predicate is a conjunction of *atomic* state predicates. An atomic state predicate is an inequality over places, i.e. $\sum_i \lambda_i p_i \leq \lambda$, with $\lambda_i, \lambda \in \mathbb{Z}$. We aim to verify the reachability of such markings using the modular state space, without constructing the state space of the composed Petri net. A method to solve this is presented in [3]. There, the problem can be divided into two parts: Atomic state predicates where all occurring places belong to the same module (1) and atomic state predicates where occurring places spread across multiple modules (2).
In the first case, reachability can be verified using just the local reachability graph of the corresponding module. The atomic state predicate can be encoded as a BDD that characterizes the set of markings satisfying the predicate. In [1], a method for this is presented assuming $\lambda_i \in \mathbb{N}$. By

joining that BDD with one that encodes a segment, the intersection of the marking sets can be checked for satisfying markings.
For the second case, according to [3], an atomic proposition can be split according to the module affiliation of places, resulting in partial sums for every module. We present a way to calculate these sums for segments encoded as BDDs by analyzing their structure. This way, these atomic state predicates can be verified for vertices in the synchronization graph. Combining both methods, we can verify such state predicates using the modular state space.

## References


1. Abío, I., Nieuwenhuis, R., Oliveras, A., Rodríguez-Carbonell, E., Mayer-Eichberger, V.: A New Look at BDDs for Pseudo-Boolean Constraints. Journal of Artificial Intelligence Research **45**, 443–480 (Nov 2012). https://doi.org/10.1613/jair.3653
2. Burch, J.R., Clarke, E.M., McMillan, K.L., Dill, D.L., Hwang, L.J.: Symbolic model checking: 1020 States and beyond. Information and Computation **98**(2), 142–170 (Jun 1992). https://doi.org/10.1016/0890-5401(92)90017-A
3. Gaede, J., Wallner, S., Wolf, K.: Modular State Spaces - A New Perspective. In: Kristensen, L.M., van der Werf, J.M. (eds.) Application and Theory of Petri Nets and Concurrency. pp. 312–332. Springer Nature Switzerland, Cham (2024). https://doi.org/10.1007/978-3-031-61433-0_15
4. Pastor, E., Cortadella, J., Roig, O.: Symbolic analysis of bounded Petri nets. IEEE Transactions on Computers **50**(5), 432–448 (May 2001). https://doi.org/10.1109/12.926158, conference Name: IEEE Transactions on Computers
5. Valmari, A.: The state explosion problem. In: Reisig, W., Rozenberg, G. (eds.) Lectures on Petri Nets I: Basic Models: Advances in Petri Nets, pp. 429–528. Lecture Notes in Computer Science, Springer, Berlin, Heidelberg (1998). https://doi.org/10.1007/3-540-65306-6_21, https://doi.org/10.1007/3-540-65306-6_21

# Improving the Simplicity of Workflow Nets

Patrizia Schalk[0009−0001−7757−6628]

University of Augsburg, Universitätsstraße 6a, 86135, Augsburg, Germany
patrizia.schalk@informatik.uni-augsburg.de

**Abstract.** Understanding a process model is as important as its conformance with the data it was created for. Stake-holders and process analysts need to understand a model to work efficiently with it, and certain algorithms work much faster when the model is simple. But especially large processes create big and complex models. In our research, we explore ways to improve the simplicity of workflow nets by finding regions that guarantee simpler structures. While doing so, we allow the language of the net to undergo minor changes, affecting the fitness and precision of the net. We control which changes are acceptable by using weights for the quality dimensions.

**Keywords:** Simplicity · Complexity · Workflow nets · Process Mining.

Evaluating the quality of discovered process models is a key part of process mining. The most commonly used measures concern the quality dimensions *fitness* and *precision*. Apart from these dimensions, there are *generalization* and *simplicity* [1]. The latter quality dimensions are not as well understood as the former: The generalization and simplicity of a model depend on factors for which we have no data. For example, simplicity depends on personal preferences of people that will work with the model. Since it is difficult to define formally, the simplicity dimension is often overlooked or handled implicitly during process discovery. This results in difficult to understand process models. For an example, take the workflow net $N_\alpha$ of Figure 1. This net is the result of the alpha miner [2], when called for the input language $\{aegi, agei, begi, bgei, cfhi, chfi, dfhi, dhfi\}$. It is difficult to immediately recognize the language of $N_\alpha$, but why is that? One reason is that $N_\alpha$ contains implicit places whose deletion would not change the language of the net. Another reason is that the net is not planar and has many crossing arcs. One could also argue that the concurrency in $N_\alpha$ introduces complexity. In general, there are many possible reasons to why a process model is complex [3]. But finding that a model is difficult to understand is not the end of the story. As soon as we know the reason, we might want to take action and simplify an existing

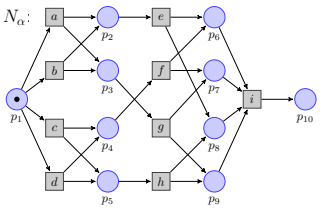


**Fig. 1.** A workflow net, $N_\alpha$, discoverd by the alpha miner.

process model, so it is easier to work with. As an example, take again the net $N_\alpha$ of Figure 1. If we want to increase the simplicity of $N_\alpha$ by reducing its size, we can delete the implicit places $p_7$ and $p_8$, resulting in the net $N_1$ of Figure 2. With this operation, we do not change the language of the net, so its fitness and

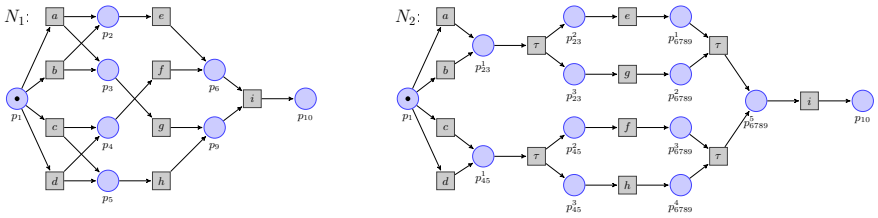


**Fig. 2.** Two simplified versions of $N_\alpha$ according to different complexity measures.

precision scores remain unchanged. If we instead consider the density of the net, we can improve the simplicity of $N_\alpha$ by finding less dense representations of xor- and parallel joins, resulting in the net $N_2$ of Figure 2. Again, the language of the net does not change. We are interested in a general set of rules we can perform to simplify the net without changing its behavior.

We found that there are several algorithms that reduce the size of a workflow net based on a set of reduction rules [4–6]. However, these algorithms aim to simplify the model to execute model checking techniques more efficiently, so they allow changing the language of the model and retain only special properties. Other approaches introduce super-nodes with special semantics to lower the cognitive load while understanding a process model [7].

The goal of our research is to define an algorithm that increases the simplicity of workflow nets while allowing only for minimal changes in its fitness and precision score. To do so, we take an event log, measures to compute fitness, precision, and simplicity, and weights for these dimensions as input. With the weights, we can express which quality dimensions are most important to us, so too drastic changes on their scores are avoided. Currently, we do not take the generalization dimension into account, since existing generalization measures have serious weaknesses [8]. In a first step towards our goal, we formulate generic rules for each complexity measure that simplify a workflow net without changing its language, like in the example of Figure 2. For the second step, we alleviate the constraint of language-equivalence. Finally, we aim to define *nifty regions*. In contrast to minimal regions, which guarantee language-inclusion of the input language, *nifty regions* focus on keeping the workflow net as simple as possible according to a simplicity measure. For example, *nifty regions* avoid edge crossings or implicit places. With our approach based on regions, we can then formulate process discovery techniques that actively take the simplicity of a net into account. This enables new ways for comparing simplicity measures and deepening the understanding of the simplicity dimension.

## References


1. W.M.P. van der Aalst, *"Process mining: Data science in action"* (2016) Berlin, Heidelberg: Springer,
   DOI: 10.1007/978-3-662-49851-4
2. W.M.P. van der Aalst, T. Weijters, L. Maruster, *"Workflow mining: Discovering process models from event logs* (2004) IEEE Trans. Knowl. Data Eng.,
   DOI: 10.1109/TKDE.2004.47
3. J. Mendling, *"Metrics for process models"*, (2008) Berlin, Heidelberg: Springer,
   DOI: 10.1007/978-3-540-89224-3
4. H. Bride, O. Kouchnarenko, F. Peureux, *"Reduction of workflow nets for generalised soundness verification"*, (2017) LNCS vol. 10145,
   DOI: 10.1007/978-3-319-52234-0_6
5. P.E. Hoffmann, *"Workflow nets: Reduction rules and games*, (2017) Ph.D. thesis, Technische Universität München
6. M. Chiachío, A. Saleh, S. Naybour, J. Chiachío, John Andrews, *"Reduction of Petri net maintenance modeling complexity via Approximate Bayesian Computation"*, (2022) Reliability Engineering & System Safety vol. 222,
   DOI: 10.1016/j.ress.2022.108365
7. V.P. Cosma, A.K.F. Christfort, T.T. Hildebrandt, X. Lu, H.A. Reijers, T. Slaats, *"Improving simplicity by discovering nested groups in declarative models"*, (2024) CAiSE 2024, LNCS vol. 14663,
   DOI: 10.1007/978-3-031-61057-8_26
8. A.F. Syring, N. Tax, W.M.P. van der Aalst, *"Evaluating conformance measures in process mining using conformance propositions"*, (2019) Trans. Petri Nets Other Model. Concurr. vol. 14,
   DOI: 10.1007/978-3-662-60651-3_8

# Model Factors Influencing Petri Net Understandability: A Case Study on Simplicity

Marc Kimmel, Patrizia Schalk, and Robert Lorenz

University of Augsburg, Universitätsstraße 6a, 86159 Augsburg, Germany
marc.kimmel@uni-a.de, patrizia.schalk@informatik.uni-augsburg.de,
robert.lorenz@informatik.uni-augsburg.de

**Abstract.** A user's ability to understand a Petri net is influenced not only by personal factors, such as their familiarity with the semantics and confidence with specific structures, but also by structural factors like the size of the net or the number of arcs. Identifying the most important factors impacting understandability requires empirical studies rather than mathematical analysis. In this work, we outline our plans to conduct a survey to evaluate how different model factors impact the understandability of Petri net models. We expect this survey to yield insights into the factors that influence Petri net understandability and provide a foundation for future research.

**Keywords:** Process Mining · Case Study · Simplicity

Process mining allows its users to automatically discover process models from event logs. Due to the autonomy of process discovery algorithms, it is important to check the quality of the generated models. Four quality criteria are used to do this: Fitness, precision, generalization, and simplicity. While fitness and precision are well-understood, simplicity is often overlooked or handled only implicitly. For our purposes, we consider simplicity to be how easily the model can be understood, reflecting its level of understandability. Unlike other measures, understandability depends on the specific use case and the preferences of the users working with the model. As a result, it cannot be universally defined; instead, it must be tailored to the use case and user base before the evaluation. In this work, we focus on the *understandability* of Petri nets as an indicator for its simplicity.

Existing empirical studies explored process model understandability, focussing on various modeling languages and aspects [1]. Recker et al. examined the differences between languages like BPMN and EPC, analyzing which is easier for beginners to learn [3]. Other studies focus on single modeling languages, investigating whether specific structures improve understandability. For example, Figl et al. analyzed how the design of routing symbols affects the understandability of BPMN models [2]. Reijers et al. introduced a valuable methodology while examining multiple model factors and their correlation with understandability [4]. In their study, participants evaluated eight EPC models with distinct characteristics, answering six Yes/No questions and one open-ended question regarding

model understandability. The authors assigned a score to each model based on the number of correct answers to these questions, which served as the dependent variable in a regression analysis. This analysis revealed significant effects of model factors such as density, average connector degree, and cross-connectivity on the understandability of the models.

Our goal is to adapt the methodology of Reijers et al. to Petri nets and conduct a case study to identify the factors influencing their understandability. Similar to this work, we differentiate between personal factors, such as participants' experience and knowledge, and model-related factors, which can significantly affect Petri net understandability: Experts potentially understand Petri net models better than beginners, which could obscure the relationship between model factors and understandability. To minimize the impact of personal factors, we will invite only students from the University of Augsburg who passed the Process Mining course, ensuring participants have similar knowledge of Petri nets. While additional surveys with other groups are planned, their differing knowledge levels will require separate analysis. We want to analyze 20 model factors, which we will use as independent variables in our correlation analysis. These factors include density, average connector degree, and cross-connectivity, which showed significant effects on the understandability in the study by Reijers et al. and will be validated for Petri nets. Additionally, we selected factors specific to Petri nets, focusing on the reachability graph. These factors are listed in Table 1, where we differentiate between the amount of model elements, quantitative, and qualitative factors.

**Table 1.** Structural model factors that might influence understandability.

| | |
|---|---|
| **Elements** | Amount of nodes, edges, tau/duplicate transitions, connectors |
| **Quantitative** | Avg./Max. node/connector degree, Ratio of parallel transitions, Connectivity, Density, Separability, Sequentiality, Max tokens |
| **Qualitative** | Safe?, End places, deadlock markings, Parallel transitions, Traces |

We designed eight models with significant variation in their factors, and created two slightly different variants for each model. In the case study, we will show one of the variants of each model to the participants. They will then be asked 13 single choice questions on the models' properties, like execution order, exclusivity, repeatability, and concurrency. Afterward, participants will compare all pairs of the eight models and choose the one they find more understandable in each comparison. These pairwise comparisons will generate a ranking of the models based on subjective understandability, allowing for correlation analysis between the correctness of responses and the ranking.

To conduct the case study, we developed a tool called *Petri-Dish* that automates the creation, execution, and evaluation of surveys for Petri nets. The tool automatically imports Petri net models in PNML format and calculates all 20 model factors from Table 1 for each model, simplifying model creation and question formulation. During the survey, participants can adjust the model

layout via an interactive interface, minimizing the layout's effect on understandability. *Petri-Dish* also includes a feature for pairwise model comparisons. For evaluation, the tool consolidates the participants' responses and comparisons and calculates a score based on the correct answers, reducing the overall effort to evaluate the results.

Our case study aims to validate the findings of Reijers et al. for Petri nets and uncover additional factors influencing Petri net understandability. The rankings from pairwise comparisons will serve as an extra indicator of understandability, complementing the survey scores in our correlation analysis. It will be interesting to see whether a significant difference exists between the correlation of model factors and the subjective perception from pairwise comparisons. The *Petri-Dish* tool will streamline the survey process, enabling efficient and accurate evaluations. We expect this approach to produce significant results and establish a foundation for future research on the simplicity dimension.

## References

1. Dikici, A., Turetken, O., Demirors, O.: Factors influencing the understandability of process models: A systematic literature review. Information and Software Technology **93**, 112–129 (2018). https://doi.org/https://doi.org/10.1016/j.infsof.2017.09.001, https://www.sciencedirect.com/science/article/pii/S0950584916302889
2. Figl, K., Recker, J., Mendling, J.: A study on the effects of routing symbol design on process model comprehension. Decision Support Systems **54**(2), 1104–1118 (2013). https://doi.org/https://doi.org/10.1016/j.dss.2012.10.037, https://www.sciencedirect.com/science/article/pii/S0167923612003119
3. Recker, J., Dreiling, A.: Does it matter which process modelling language we teach or use? an experimental study on understanding process modelling languages without formal education. ACIS 2007 Proceedings - 18th Australasian Conference on Information Systems (01 2007)
4. Reijers, H.A., Mendling, J.: A study into the factors that influence the understandability of business process models. IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans **41**(3), 449–462 (May 2011). https://doi.org/10.1109/TSMCA.2010.2087017

# Modelling and Analysis of Microservice Based Systems by Composing Petri Nets

Görkem Kılınç Soylu[1] and Luca Bernardinello[2]

[1] Izmir University of Economics, Turkey
gorkem.soylu@ieu.edu.tr
[2] University of Milano-Bicocca, Italy
luca.bernardinello@unimib.it

## 1 Microservice-based systems

The rapid advancements in cloud technology and the growing demand for continuous digital transformation have led to the rise of modern applications that are responsive, flexible, and reliable. To meet these needs, new software design paradigms have emerged, notably Microservice-based Architecture (MSbA) [9, 4]. MSbA involves breaking down an application into loosely coupled, independently deployable services, known as microservices. This modular approach enhances agility, scalability, and fault tolerance in applications [1, 5].

While the Microservice-based Architecture (MSbA) offers numerous advantages for software engineering, such as flexibility and scalability, it also introduces significant challenges due to its complex and distributed nature. These challenges, particularly in areas like modelling, testing, and verification, are highlighted in [10, 11].

## 2 Utilising Petri nets for modelling microservice-based systems

Formal methods, such as Petri nets [8, 7], offer a robust solution to these challenges by providing a rigorous and systematic approach to modelling and analyzing microservice-based systems. These methods are beneficial because they enable the identification of potential errors or vulnerabilities in the system design before implementation, thus saving time and resources. Formal methods ensure system correctness, provided the application adheres to the model, and assist in generating test cases by computing reachable states and identifying critical ones.

In [6], a case study is performed to demonstrate the suitability and usefulness of Petri nets to model and analyse microservice based systems. In the study, a Banking-as-a-Service (BaaS) application was modelled via 1-safe Petri nets, a class of Petri nets in which each place can have at most 1 token. The model was then analysed to verify some structural and behavioural properties. The study follows an approach based on abstraction and composition. A microservice based software system can include many microservices working together and

communicating in different ways. However, the internal details of a service are not known by the other services. Thus the internal details of a specific service can be abstracted from the overall view while focusing on the communication and collaboration among the services in the whole system.

The case study draws a sketch for a compositional method while showing the use of Petri nets for modelling and analysis of microservice based systems to verify some structural and behavioural properties. While the case study provides valuable insights and demonstrates the practical applicability of 1-safe Petri nets in this context, our objective is to extend and refine this work and provide a more detailed and sound framework for the abstraction and composition of microservice models.

## 3 A compositional approach based on regions

The approach we propose here rests on a notion of abstraction and composition of Petri nets, based on regions. A *region* of a labelled transition system is a subset of states, $r$ such that, for each label, there is a fixed crossing relation: for a given label $x$, either all arcs labelled by $x$ enter $r$, or all those arcs leave $r$, or no arc crosses the border of $r$ (see [2]).

The marking graph of a 1-safe Petri net is a labelled transition system, where markings correspond to states. The set of markings in which a given place is marked is a region; the marking graph can have regions which do not correspond to actual places; those regions are *potential places*, in the sense that one can add new places to the net, corresponding to such regions, without changing the overall behaviour of the net.

In particular, there can be regions corresponding to the logical disjunction of existing places, seen as propositions. Exploiting this fact, one can define a notion of abstraction of a net, or of a part of a net, by keeping some "coarse" places, and masking some more detailed ones.

Abstraction and composition based on regions, for elementary net systems, are defined and studied in [3], where it is shown that some behavioural properties are preserved, or reflected, by abstraction or composition, in particular with respect to invariants and bisimulation relations. We plan to extend those results to 1-safe Petri nets, which are a generalization of elementary net systems.

The approach presented in [6] can then be made more general and flexible. A specific microservice can be modelled in detail by a 1-safe Petri net, where any assumptions about the behaviour of the environment of the service are modelled without details, but still respecting behavioural features of implementations. This "environment" may include other microservices, users, and other components of the real system. It is then possible to explore the behaviour of one, or more, microservices with respect to different assumptions about the operating environment, and formalize precisely when a given implementation of the environment satisfies those assumptions.

## References


1. Alshuqayran, N., Ali, N., Evans, R.: A systematic mapping study in microservice architecture. In: 2016 IEEE 9th International Conference on Service-Oriented Computing and Applications. pp. 44–51. IEEE (Nov 2016). https://doi.org/10.1109/SOCA.2016.15
2. Badouel, É., Bernardinello, L., Darondeau, P.: Petri Net Synthesis. Texts in Theoretical Computer Science. An EATCS Series, Springer (2015). https://doi.org/10.1007/978-3-662-47967-4, https://doi.org/10.1007/978-3-662-47967-4
3. Bernardinello, L., Monticelli, E., Pomello, L.: On preserving structural and behavioural properties by composing net systems on interfaces. Fundam. Informaticae **80**(1-3), 31–47 (2007), http://content.iospress.com/articles/fundamenta-informaticae/fi80-1-3-03
4. Bonér, J.: Reactive Microservices Architecture. O'Reilly Media, Inc. (2016)
5. Di Francesco, P., Lago, P., Malavolta, I.: Architecting with microservices: A systematic mapping study. Journal of Systems and Software **150**, 77–97 (2019). https://doi.org/https://doi.org/10.1016/j.jss.2019.01.001, https://www.sciencedirect.com/science/article/pii/S0164121219300019
6. Kılınç Soylu, G., Demirörs, O.: An exploratory case study: Using Petri nets for modelling microservice-based systems. In: 49th Euromicro Conference on Software Engineering and Advanced Applications, SEAA 2023, Durres, Albania, September 6-8, 2023. pp. 254–261. IEEE (2023). https://doi.org/10.1109/SEAA60479.2023.00047, https://doi.org/10.1109/SEAA60479.2023.00047
7. Murata, T.: Petri nets: Properties, analysis and applications. Proc. IEEE **77**(4), 541–580 (1989). https://doi.org/10.1109/5.24143
8. Petri, C.A.: Kommunication mit Automaten. Ph.D. thesis, Technischen Hoschule Darmstadt (1962)
9. Thönes, J.: Microservices. IEEE Software **32**(1), 116–116 (2015). https://doi.org/10.1109/MS.2015.11
10. Ünlü, H., Bilgin, B., Demirörs, O.: A survey on organizational choices for microservice-based software architectures. Turkish J. Electr. Eng. Comput. Sci. **30**(4), 1187–1203 (2022). https://doi.org/10.55730/1300-0632.3843, https://doi.org/10.55730/1300-0632.3843
11. Ünlü, H., Kennouche, D.E., Kılınç Soylu, G., Demirörs, O.: Microservice-based projects in agile world: A structured interview. Inf. Softw. Technol. **165**, 107334 (2024). https://doi.org/10.1016/J.INFSOF.2023.107334, https://doi.org/10.1016/j.infsof.2023.107334

# Renew Suite: A Tool to Compose Individual Renew Tool Sets Based on a Plugin Concept*

Marcel Hansson, Daniel Moldt, and Laif-Oke Clasen

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

**Keywords:** Renew · Modularization · Composition · Plugin Architecture · Petri Nets · Tool Set.

Tools are becoming increasingly complex. One solution for large tools is decomposing it into smaller feature sets or components. The plugin concept is one of the conceptual solutions for customizing individual components [7] (see e.g. plugins for browsers or software development environments). At language level, this is addressed by the concept of modularization. In Java, this is the Java Platform Module System (JPMS) [9], which was introduced with Release 9.

Work on a tool environment for Petri nets has been underway in Hamburg for over 25 years: Renew[1] [6]. The strength of Renew lies in the modeling, simulation and analysis of Petri net models. Other modeling techniques can also be covered. As a growing software system, Renew requires both the individual application-related components, which can be put together individually (as plugins), and modularization using JPMS [3,5,8], which makes it possible to establish a clean software architecture.

Due to the numerous variants, there are incompatible components that cannot be used simultaneously. This applies to the commands, the interface arrangements, the formal foundations and the application-related use (in particular redundancies).

To address the static and dynamic composition of our Renew components, we introduce a new concept to Renew: Suite[2] [10]. With the help of the Suite concept, the above-mentioned aspects are addressed by selecting individual components and composing them into a concrete tool variant, a suite. The development followed a prototyping approach [11] and was carried out by students as part of normal teaching projects at our department, in particular under the guidance of the first author. In addition to the static composition, dynamic composition is possible, by adding and removing components at runtime. A conceptual basis is the use of the plugin architecture described in [2,4]. However, this Petri net-based model cannot be used directly by users, as complex models would have to be implemented. To simplify matters, the details of the implementation were hidden from users and restricted to purely plugin-based use.

However, the plugin architecture used in Renew still required the use of complex configuration files and their management. Each plugin has its own con-

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

[1] www.renew.de

[2] https://en.wikipedia.org/wiki/Software_suite

figuration files with various properties. These properties are often hidden for the usual user despite being potentially useful for their use case. The SUITE concept and the associated plugins provide users with a way to configure these predetermined properties.

The management of dependencies involve a great deal of effort, which ultimately cannot be meaningfully taken over by users. The SUITE concept and the associated plugins can be used to create pre-configurations that replace the previous manually composed and configured RENEW variants for individual usage scenarios. Thanks to the inherent dependency management that is adopted in this way, the composed plugins are compatible and usable. Changes can also be made during runtime using the underlying JPMS, making it easy to switch between different tool configurations. The JPMS allows Java code to be cleanly unloaded at runtime and therefore also plugins.

By means of accompanying descriptions, users get familiar with the services of the respective suites and the configuration options. A graphical user interface is available for experienced users so that they can configure RENEW by creating their own suites.

With regard to the technical basics, it should be noted that the combination of plugins from an application and architecture perspective as well as modules or layers relating to JPMS is of central importance for the implementation. The components of SUITE are three plugins: the underlying framework, the GUI and the creation and management of suites. A headless use is possible, allowing SUITE to configure RENEW for various applications, like container or server applications. RENEW SUITE supports any location of the plugins in the system and can therefore easily integrate plugins from MULAN, a Petri net-based multi-agent architecture [1]. It should be emphasized that non-modularized plugins, such as those still available in MULAN, can also be integrated into a suite. However, the advantages of the JPMS regarding a clean unloading of a plugin, do not apply then.

The possibility of creating plugin configurations that has been implemented so far already makes it possible to significantly streamline the RENEW user interface. Since plugins themselves can be fairly large and can provide several different functionalities at once, the option of deactivating individual functions of plugins via the SUITE concept will be implemented in the further course of development. An initial release of the SUITE concept is planned for an upcoming release of RENEW.

## References

1. Cabac, L.: Modeling Agent Interaction Protocols with AUML Diagrams and Petri Nets. Diploma thesis, University of Hamburg, Department of Computer Science, Vogt-Kölln Str. 30, D-22527 Hamburg (Dec 2003)
2. Cabac, L., Duvigneau, M., Moldt, D., Rölke, H.: Multi-agent concepts as basis for dynamic plug-in software architectures. In: Proceedings of the International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS 2005). pp. 1157–1158 (2005), https://doi.org/10.1145/1082473.1082671

3. Clasen, L.O., Moldt, D., Hansson, M., Willrodt, S., Voß, L.: Enhancement of Renew to version 4.0 using JPMS. In: Köhler-Bußmeier, M., Moldt, D., Rölke, H. (eds.) Proceedings of the International Workshop on Petri Nets and Software Engineering 2022 co-located with the 43rd International Conference on Application and Theory of Petri Nets and Concurrency (PETRI NETS 2022), Bergen, Norway, June 20th, 2022. CEUR Workshop Proceedings, vol. 3170, pp. 165–176. CEUR-WS.org (2022), https://ceur-ws.org/Vol-3170
4. Duvigneau, M.: Konzeptionelle Modellierung von Plugin-Systemen mit Petrinetzen, Agent Technology – Theory and Applications, vol. 4. Logos Verlag, Berlin (2010), http://www.logos-verlag.de/cgi-bin/engbuchmid?isbn=2561&lng=eng&id=
5. Janneck, J.R.: Modularizing a Plugin System Using Java Modules: Application to a Medium-Sized Open-Source Project. Master thesis, University of Hamburg, Department of Informatics, Vogt-Kölln Str. 30, D-22527 Hamburg (Mar 2021)
6. Kummer, O., Wienberg, F., Duvigneau, M., Cabac, L., Haustermann, M., Mosteller, D.: Renew – the Reference Net Workshop (Feb 2023), http://www.renew.de/, release 4.1
7. Mayer, J., Melzer, I., Schweiggert, F.: Lightweight plug-in-based application development. In: Aksit, M., Mezini, M., Unland, R. (eds.) Objects, Components, Architectures, Services, and Applications for a Networked World. pp. 87–102. Springer Berlin Heidelberg, Berlin, Heidelberg (2003)
8. Moldt, D., Johnsen, J., Streckenbach, R., Clasen, L., Haustermann, M., Heinze, A., Hansson, M., Feldmann, M., Ihlenfeldt, K.: RENEW: modularized architecture and new features. In: Gomes, L., Lorenz, R. (eds.) Application and Theory of Petri Nets and Concurrency - 44th International Conference, PETRI NETS 2023, Lisbon, Portugal, June 25-30, 2023, Proceedings. Lecture Notes in Computer Science, vol. 13929, pp. 217–228. Springer Nature Switzerland AG, Cham, Switzerland (2023), https://doi.org/10.1007/978-3-031-33620-1_12
9. Oracle: Project Jigsaw (9 2017), https://openjdk.java.net/projects/jigsaw/, [Online; accessed 19.05.2022]
10. Rugaber, S.: A tool suite for evolving legacy software. In: 1999 International Conference on Software Maintenance, ICSM 1999, Oxford, England, UK, August 30 - September 3, 1999. pp. 33–39. IEEE Computer Society (1999). https://doi.org/10.1109/ICSM.1999.792496, https://doi.org/10.1109/ICSM.1999.792496
11. Wilde, T., Hess, T.: Forschungsmethoden der wirtschaftsinformatik. Wirtschaftsinf. **49**(4), 280–287 (2007). https://doi.org/10.1007/S11576-007-0064-Z, https://doi.org/10.1007/s11576-007-0064-z

# Modelling Cooperating Failure-Resilient Processes

Rüdiger Valk

University of Hamburg
ruediger.valk@uni-hamburg.de

**Keywords:** circular traffic queues, Petri nets, cycloids, failure-resilient systems

## 1 Introduction

Consider a distributed system of a finite number of circular and sequential processes. The processes are synchronized by uni-directional one-bit channels in such a way that they behave like a circular traffic queue when folded together. To give an example, on the top of Figure 1 a circular queue of length eight is represented, containing three traffic items $a_0, a_1$ and $a_2$ in positions $0, 1$ and 2, respectively. Obviously traffic items $a_0$ and $a_1$ cannot proceed until $a_2$ makes a step. In this example however, also $a_2$ is restricted in such a way that only two steps are possible until $a_1$ and $a_0$ made a step. This restriction is symbolically represented by the two arrows starting in position 2. The dashed line is a hint to the circular behaviour of the queue. If this description of the intended synchronisation leaves any doubt, the reader should look at the net $\mathcal{C}_0$ on the left of Figure 1. In this net the steps of traffic item $a_i$ are given by transitions $[t_0, a_i], [t_1, a_i], \cdots, [t_7, a_i]$, connected by places of the form $[s_j, a_i]$. See $[s_0, a_0]$ for an example. Such a place is denoted *forward output place* of $[t_j, a_i]$. In contrast, the places that serve to synchronize the processes are *backward output places* of the form $[s'_j, a_i]$ (see $[s'_2, a_2]$ for an example). $\mathcal{C}_0$ is the net of the cycloid $\mathcal{C}_0(2, 3, 4, 6, M_0)$. The theory of cycloids cannot be introduced in this context, but the results of this contribution can be explained with the help of nets with this structure. Cycloids have been introduced by C.A. Petri [1], the basics of cycloids, in particular *regular* cycloids, can be found in [2], as well as the full paper of this extended abstract in [4].

## 2 Stop-resilient Processes

Considered as cooperating processes cycloids perform a strong synchronization regimen. It is therefore surprising that we can model these processes with a small extension so that they can be stopped or fail without stopping the other processes. As a by-product it is proved that by eliminating the traffic item with the highest index, the cycloid for one traffic item less and one gap more is obtained. Since the extension is defined by a folding of the backward output places only, the forward places and transitions of the processes are not modified

and the *cycloid algebra* [3] can be applied furthermore. To obtain a live system when one traffic item is eliminated we require that at least $\beta > 1$ traffic item are present. A cycloid system $\mathcal{C} = \mathcal{C}(\alpha, \beta, \gamma, \delta, M_0)$ is *regular* if $\beta$ divides $\delta$ and *canonical* if $\beta = \gamma = \delta$.

**Definition 1.** *For a given regular cycloid system $\mathcal{C} = \mathcal{C}(\alpha, \beta, \gamma, \delta, M_0)$ with $\beta > 1$, process length $p$ and a fixed set $D \subseteq \{0, \cdots, \beta - 1\}$ with $|D| > 1$, called the set of* back indices*, we define the* backward folding $\mathcal{C}_{bf(D)}(\alpha, \beta, \gamma, \delta, [\![M_0]\!]_D)$ *by a relation $\equiv_{bf(D)}$ on the backward places $[s'_i, a_j]$ by*

$$[s'_i, a_j] \quad \equiv_{bf(D)} \quad [s'_r, a_s] \;\Leftrightarrow\; i = r \;\wedge\; \{j, s\} \subseteq D \quad \textit{for} \quad 0 \leq i, r < p \qquad (1)$$

*The folding is extended to markings by $[\![M]\!]_D := \{[\![s]\!]_{bf(D)} | s \in M\}$. If $D = \{0, \cdots, \beta - 1\}$ the folding and the equivalence relation are called* total *and denoted by $\mathcal{C}_{bf}(\alpha, \beta, \gamma, \delta, [\![M_0]\!])$ and $\equiv_{bf}$, respectively.*

The folding is defined on backward output places modelling the channels of the cooperating processes. Therefore by the folding we switch from a message oriented synchronization to a shared variable synchronization mechanism. If $j \in D$ then the process $a_{(j+1) mod\, \beta}$ is sharing its backward input places. It is important to prove that under a mild restriction the backward folding of a cycloid is safe and live.

**Theorem 2.** *The backward folding $\mathcal{C}_{bf(D)}(\alpha, \beta, \gamma, \delta, [\![M_0]\!])$ of a regular cycloid system $\mathcal{C}(\alpha, \beta, \gamma, \delta, M_0)$ with regular $M_0$ and $\alpha + \beta \leq p + 1$ is a live and safe net, i.e. in each reachable marking each place contains at most one token.*

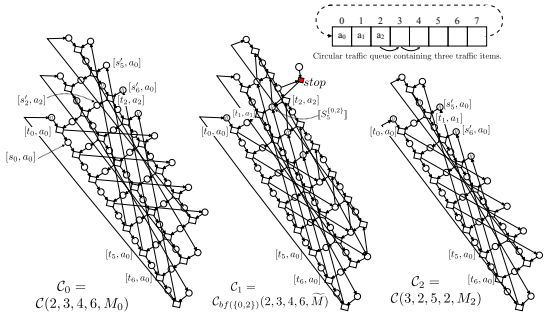


**Fig. 1.** Two cycloids and the bf-folding $\mathcal{C}_{bf(0,2)}(2, 3, 4, 6, \widetilde{M})$ as intermediate step.

Figure 1 shows the cycloid $\mathcal{C}_0 = \mathcal{C}(\alpha, \beta, \gamma, \delta, M_0) = \mathcal{C}(2, 3, 4, 6, M_0)$ and its folding $\mathcal{C}_{bf(\{0,2\})}(2, 3, 4, 6, \widetilde{M})$ as intermediate step to the cycloid $C_2 = \mathcal{C}(\alpha + 1, \beta - 1, p - (\alpha + 1), \beta - 1, M_2) = \mathcal{C}(3, 2, 5, 2)$. This formula holds in general, in particular $p - (\alpha + 1) = \beta - 1$ for $p = \alpha + \beta$ in canonical cycloids. The additional already occurred transition *stop* in $\mathcal{C}_1$ simulates the stop-failure of process 2.

## References

1. Petri, C.A.: Nets, Time and Space. Theoretical Computer Science (153), 3–48 (1996)
2. Valk, R.: Circular Traffic Queues and Petri's Cycloids. In: Application and Theory of Petri Nets and Concurrency. Lecture Notes in Computer Science, vol. 12152, pp. 176 – 195. Springer-Verlag, Cham (2020)
3. Valk, R.: Analysing cycloids using linear algebra (2024), https://arxiv.org/abs/2402.07303
4. Valk, R.: Modelling cooperating failure-resilient processes (2024), https://arxiv.org/abs/2409.18318