A Web Interface for Petri Nets with Transits and Petri Games [⋆]

Manuel Gieseking[1(✉)], Jesko Hecking-Harbusch[2], and Ann Yanich[1]

[1] University of Oldenburg, Oldenburg, Germany
`{gieseking,ann.yanich}@informatik.uni-oldenburg.de`
[2] CISPA Helmholtz Center for Information Security, Saarbrücken, Germany
`jesko.hecking-harbusch@cispa.de`

Abstract. Developing algorithms for distributed systems is an error-prone task. Formal models like Petri nets with transits and Petri games can prevent errors when developing such algorithms. Petri nets with transits allow us to follow the data flow between components in a distributed system. They can be model checked against specifications in LTL on both the local data flow and the global behavior. Petri games allow the synthesis of local controllers for distributed systems from safety specifications. Modeling problems in these formalisms requires defining extended Petri nets which can be cumbersome when performed textually.
In this paper, we present a web interface[1] that allows an intuitive, visual definition of Petri nets with transits and Petri games. The corresponding model checking and synthesis problems are solved directly on a server. In the interface, implementations, counterexamples, and all intermediate steps can be analyzed and simulated. Stepwise simulations and interactive state space generation support the user in detecting modeling errors.

1 Introduction

Distributed systems consist of several individual components. Each component has incomplete information about the other components. Asynchronous distributed systems have no fixed rate at which components progress but rather each component progresses at its individual rate between synchronizations with other components. Implementing correct algorithms for asynchronous distributed systems is difficult because they have to both work with the incomplete information of the components and for every possible scheduling between the components.

Petri nets [22,21] are a natural model for asynchronous distributed systems. Tokens represent components and transitions with more than one token correspond to synchronizations between the components. *Petri nets with transits* [9] extend Petri nets with a transit relation to model the data flow in asynchronous

[⋆] This work has been supported by the German Research Foundation (DFG) through Grant Petri Games (392735815) and through the Collaborative Research Center "Foundations of Perspicuous Software Systems" (TRR 248, 389792660), and by the European Research Council (ERC) through Grant OSARES (683300).
[1] The web interface is deployed at http://adam.informatik.uni-oldenburg.de.

distributed systems. *Flow-LTL* [9] is a specification language for Petri nets with transits and allows us to specify linear properties on both the global and the local view of the system. In particular, it is possible to globally select desired runs of the system with LTL (e.g., only fair and maximal runs) and check the local data flow of only those runs again with LTL. A model checker for Petri nets with transits against Flow-LTL is implemented in the tool AdamMC [10].

Petri games [14] define the synthesis of asynchronous distributed systems based on Petri nets and causal memory. With causal memory, players exchange their entire causal past only upon synchronization. Without synchronization, players have no information of each other. For safety winning conditions, the synthesis algorithm for Petri games with a bounded number of controllable components and one uncontrollable component is implemented in AdamSYNT [12][2]. Both tools are command-line tools lacking visual support to model Petri nets with transits or Petri games and the possibility to simulate or interactively explore implementations, counterexamples, and parts of the created state space.

In this paper, we present a web interface[3] for model checking asynchronous distributed systems with data flows and for the synthesis of asynchronous distributed systems with causal memory from safety specification. The web interface offers an input for Petri nets with transits and Petri games where the user interactively creates places, transitions, and their connections with a few inputs.

As a back-end, the algorithms of AdamMC are used to model check Petri nets with transits against a given Flow-LTL formula as specification. Internally, the problem is reduced to the model checking problem of Petri nets against LTL. Both, the input Petri net with transits and the constructed Petri net can be visualized and simulated in the web interface. For a positive result, the web interface lets the user follow the control flow of the combined system and the data flow of the components. For a negative result, the web interface simulates the counterexample with a visual separation of the global and each local behavior.

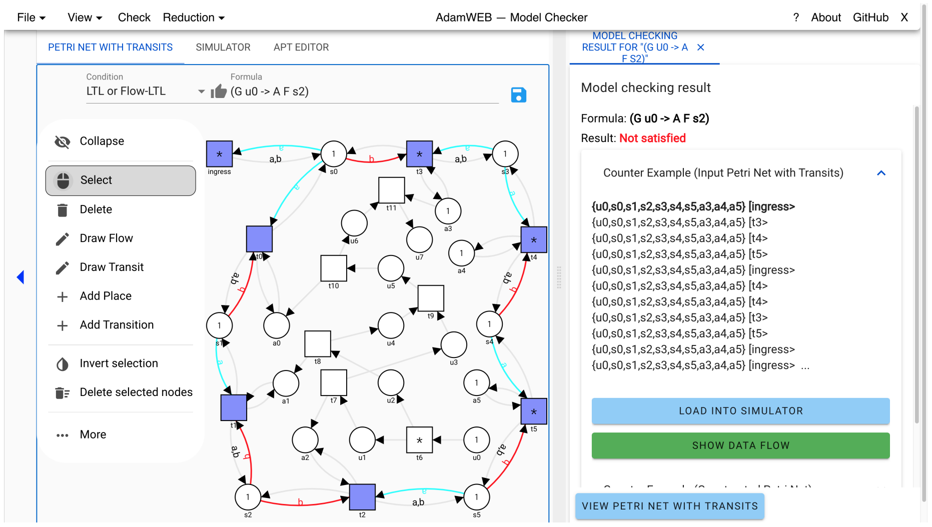
The algorithms of AdamSYNT solve the given Petri game with safety specification. Internally, the problem is reduced to solving a finite two-player game with complete information. For a positive result, a winning strategy for the Petri game and the two-player game can be visualized and the former can be simulated. For a negative result, the web interface lets the user interactively construct strategies of the two-player game and highlights why they violate the specification. These new intuitive construction methods, interactive features, and visualizations are of great impact when developing asynchronous distributed systems.

2 Web Interface for Petri Nets with Transits

The web interface can model check Petri nets with transits against Flow-LTL. We use an example from software-defined networks to showcase the workflow.

[2] AdamSYNT was previously called Adam. From now on, AdamMC and AdamSYNT are combined in the tool Adam (https://github.com/adamtool/adam).

[3] The web interface is open source (https://github.com/adamtool/webinterface) and a corresponding artifact to set it all up locally in a virtual machine is available [16].



Condition: LTL or Flow-LTL
Formula: (G u0 -> A F s2)

Model checking result

Formula: **(G u0 -> A F s2)**
Result: Not satisfied

Counter Example (Input Petri Net with Transits)

```
{u0,s0,s1,s2,s3,s4,s5,a3,a4,a5} [ingress>
{u0,s0,s1,s2,s3,s4,s5,a3,a4,a5} [t3>
{u0,s0,s1,s2,s3,s4,s5,a3,a4,a5} [t4>
{u0,s0,s1,s2,s3,s4,s5,a3,a4,a5} [t5>
{u0,s0,s1,s2,s3,s4,s5,a3,a4,a5} [ingress>
{u0,s0,s1,s2,s3,s4,s5,a3,a4,a5} [t4>
{u0,s0,s1,s2,s3,s4,s5,a3,a4,a5} [t4>
{u0,s0,s1,s2,s3,s4,s5,a3,a4,a5} [t3>
{u0,s0,s1,s2,s3,s4,s5,a3,a4,a5} [t5>
{u0,s0,s1,s2,s3,s4,s5,a3,a4,a5} [ingress>
{u0,s0,s1,s2,s3,s4,s5,a3,a4,a5} [ingress>  ...
```

Fig. 1. Screenshot from the web interface for the model checking workflow.

Workflow for Petri Nets With Transits One application domain for Petri nets with transits are *software-defined networks (SDNs)* [20,4]. The nodes of the network are *switches* which forward *packets* along the edges of the network according to the *routing configuration*. Packets enter the network at *ingress switches* and leave it at *egress switches*. SDNs separate the packet forwarding process, called the *data plane*, from the routing process, called the *control plane*. *Concurrent updates* to the routing configuration are difficult to get right [15].

The separation of data and control plane and updates to the routing configuration can be encoded into Petri nets with transits [9]. Using this encoding, we demonstrate the workflow of the web interface for model checking an asynchronous distributed system with data flows. The packets of the SDN are modeled by the data flow in the Petri net with transits. The data flow relation as an extension from Petri nets to Petri nets with transits is depicted as colored and labeled arcs. In Fig. 1, the web interface presents the resulting Petri net with transits $\mathcal{N}$. First, we use the tools on the left to create for each switch a place si with $i \in \{0, \dots, 5\}$ and add a token (cf. outer parts of $\mathcal{N}$). Then, we create transitions for the connections between the switches and for the origin of packets in the SDN (cf. transition *ingress* in the top-left corner) and link them with flows in both directions. Additionally, we create local transits between the switches corresponding to the forwarding of packets. They are displayed in light blue and red and are identified by the letters. This constitutes the *data plane*.

Next, we define the *control plane*, i.e., which forwarding is activated. Each transition to forward packets is connected to a place ai with $i \in \{0, \dots, 5\}$ which has a token when the forwarding is configured initially (cf. places $a3$, $a4$, and $a5$) and no token otherwise (cf. places $a0$, $a1$, and $a2$). For the concurrent update, we create places ui with $i \in \{0, \dots, 7\}$ and transitions ti with $i \in \{6, \dots, 11\}$ with corresponding flows (cf. inner parts of $\mathcal{N}$).

Transitions for the forwarding are set as weak fair, i.e., whenever a transition is infinitely long enabled in a run, it also has to fire infinitely often, indicated by the purple color of the outer transitions. Transitions for the update do not require fairness assumptions. A satisfied Flow-LTL formula is $A\,F\,s5$ specifying that all packets eventually reach switch $s5$. An unsatisfied formula is $(G\,u0 \Rightarrow A\,F\,s2)$ requiring for runs, where the update is never executed, that all packets are taking the lower-left route. The fairness assumptions and a maximality assumption, i.e., whenever some transition can fire in a run some transition fires, are automatically added to the formula. In the screenshot, a counterexample for the unsatisfied formula is displayed on the right. The first packet takes the upper-right route via transitions $t3$, $t4$, and $t5$ and the update never starts.

Features for Petri Nets with Transits. AdamMC [10] is a command-line model checking tool for Petri nets with transits and Flow-LTL [9]. The model checking problem of Petri nets with transits against Flow-LTL is solved by a reduction to Petri nets and LTL. The web interface allows displaying and arranging the nodes of the Petri net from the reduction and the input Petri net with transits. Automatic layout techniques are applied to avoid the overlapping of nodes. A physics control, which modifies the repulsion, link, and gravity strength of nodes, can be used to minimize the overlapping of edges. Heuristics generate coordinates for the constructed Petri net by using the coordinates of the input Petri net with transits to obtain a similar layout of corresponding parts.

For a positive result, the web interface allows visualizing the data flow trees for given firing sequences of the nets. For a negative result, the counterexample can be simulated both in the Petri net with transits and in the Petri net from the reduction. The witness of the counterexample for each flow subformula and the run violating the global behavior can be displayed by the web interface. This functionality is helpful when developing an encoding of a problem into Petri net with transits to ensure that a counterexample is not an error in the encoding. The constructed Petri net can be exported into a standard format for Petri net model checking (PNML) and the constructed LTL formula can be displayed.

3 Web Interface for Petri Games

The web interface can synthesize local controllers from safety specifications. The workflow is showcased for a distributed alarm system given as a Petri game.

Workflow for Petri Games We demonstrate the workflow of the web interface for the synthesis of asynchronous distributed systems with causal memory from safety specifications. Petri games separate the places of an underlying Petri net into *system places* and *environment places*. Tokens on system places are *system players* and tokens on environment places are *environment players*. Each player has *causal memory*: only upon synchronization with other players, they exchange their entire causal past. For safety specifications, the system players have to avoid that a bad place is reached for all behaviors of the environment players.

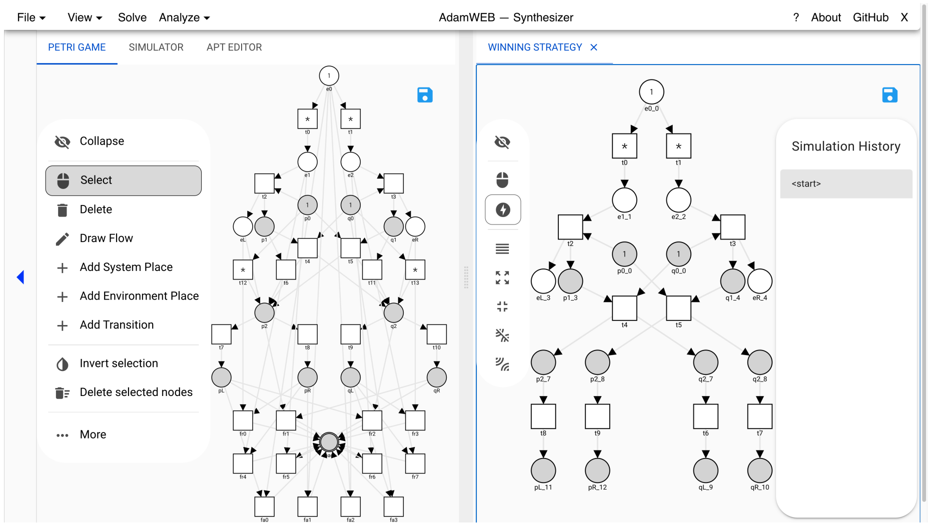


Fig. 2. Screenshot from the web interface for the synthesis workflow.

We want to obtain two local controllers of a distributed alarm system that should indicate the location of a burglary at both controllers. In Fig. 2, the web interface presents the resulting Petri game on the left and the winning strategy for the alarm system on the right. The burglar is modeled by an environment player and each component of the distributed alarm system by a system player. Environment players are on white places and system players on gray ones. We create five environment places $e0$, $e1$, $e2$, eL, and eR. The place $e0$ has a token, $e1$ and $e2$ serve for the decision to burgle a location, and eL and eR for actually burgling the location. Each component $x \in \{p, q\}$ of the alarm system has one system place $x0$ with a token, two system places $x1$ and $x2$ to detect a burglary and inform the other component, and two system places xL and xR to sound an alarm with the position of a burglary. We create rows of transitions for the environment player deciding where to burgle (first row), for the components detecting a burglary (second row), for the communication between the components (third row), and for sounding the alarm at each location (fourth row).

At last, we use transitions fai with $i \in \{0, \ldots, 3\}$ and frj with $j \in \{0, \ldots, 7\}$ connected to the bad place bad to define that the implementation of the distributed alarm system should avoid false alarms and false reports. A *false alarm* occurs if the burglar did not burgle any location but an alarm occurred, i.e., in every pair of places $\{e0\} \times \{pL, pR, qL, qR\}$. A *false report* occurs if a burglary happened at a location but a component of the alarm system indicates a burglary at the other location, i.e., in every pair of places $\{e1, eL\} \times \{pR, qR\}$ and $\{e2, eR\} \times \{pL, qL\}$. We add transitions and flows to bad for these cases.

The web interface finds a winning strategy (depicted on the right in Fig. 2) for the Petri game described above. Each component locally monitors its location ($t2$, $t3$) and simultaneously waits for information about a burglary at the other location ($t4$, $t5$). When a burglary is detected at the location of the component then it first informs the other component ($t4$, $t5$) and then outputs an alarm for the current location ($t7$, $t8$). When a component is informed about a burglary at the other location, it outputs an alarm for the other location ($t6$, $t9$).

Features for Petri Games AdamSYNT [12] is a command-line tool for Petri games [14]. The synthesis problem for Petri games with a bounded number of system players, one environment player, and a safety objective is reduced to the synthesis problem for two-player games. A winning strategy in the two-player game is translated into a winning strategy for the Petri game. Both can be visualized in the web interface. Here, the web interface provides the same features for visualizing, manipulating, and automatically laying out the elements as for model checking. It uses the order of nodes of the Petri game to heuristically provide a positioning of the strategy and allows simulating runs of the strategy. The winning strategy of the two-player game provides an additional view on the implementation to check if it is not bogus due to a forgotten case in the Petri game or specification. For an unrealizable synthesis problem, the web interface allows analyzing the underlying two-player game via a stepwise creation of strategies. This guides the user towards changes to make the problem realizable.

4 Implementation Details

The server is implemented using the Sparkjava micro-framework [23] for incoming HTTP and WebSocket connections. The client is a single-page application written in Javascript using Vue.js [25], D3 [5], and the Vuetify component library [26]. We constructed libraries out of the tools AdamMC and AdamSYNT and implemented one interface handling both libraries. Common features like the physics control of nodes share the same implementation. All components of the libraries and the web interface [2] are open source and available on GitHub [1].

5 Conclusion

We presented a web interface for two tools: AdamMC, a model checker for data flows in asynchronous distributed systems represented by Petri nets with transits, and AdamSYNT, a synthesis tool for local controllers from safety specifications in asynchronous distributed systems with causal memory represented by Petri games. The web interface makes the modeling and debugging of Petri nets with transits and Petri games user-friendly as it presents visual representations of the input, all intermediate steps, and the output of the tools. The interactive features are a great assistance for correctly modeling distributed systems.

We plan to extend the web interface and tool support to model checking Petri nets with transits against Flow-CTL* [11], to other classes of Petri games with a decidable synthesis problem [13,3], to the bounded synthesis approach for Petri games [7,8,19,18], and to high-level Petri games [17]. As our web interface is open source and easy to extend, we also plan to connect it to other tools for Petri nets like APT [24], LoLA [27], or TAPAAL [6].

References

1. Adam: https://github.com/adamtool/ (2020)
2. AdamWEB: https://github.com/adamtool/webinterface (2020)
3. Beutner, R., Finkbeiner, B., Hecking-Harbusch, J.: Translating asynchronous games for distributed synthesis. In: 30th International Conference on Concurrency Theory, CONCUR. LIPIcs, vol. 140, pp. 26:1–26:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2019), https://doi.org/10.4230/LIPIcs.CONCUR.2019.26
4. Casado, M., Foster, N., Guha, A.: Abstractions for software-defined networks. Commun. ACM **57**(10), 86–95 (2014), https://doi.org/10.1145/2661061.2661063
5. D3: https://d3js.org/ (2020)
6. David, A., Jacobsen, L., Jacobsen, M., Jørgensen, K.Y., Møller, M.H., Srba, J.: TAPAAL 2.0: Integrated development environment for timed-arc Petri nets. In: Tools and Algorithms for the Construction and Analysis of Systems - 18th International Conference, TACAS. Lecture Notes in Computer Science, vol. 7214, pp. 492–497. Springer (2012), https://doi.org/10.1007/978-3-642-28756-5_36
7. Finkbeiner, B.: Bounded synthesis for Petri games. In: Correct System Design - Symposium in Honor of Ernst-Rüdiger Olderog on the Occasion of His 60th Birthday. Lecture Notes in Computer Science, vol. 9360, pp. 223–237. Springer (2015), https://doi.org/10.1007/978-3-319-23506-6_15
8. Finkbeiner, B., Gieseking, M., Hecking-Harbusch, J., Olderog, E.: Symbolic vs. bounded synthesis for Petri games. In: Sixth Workshop on Synthesis, SYNT@CAV. EPTCS, vol. 260, pp. 23–43 (2017), https://doi.org/10.4204/EPTCS.260.5
9. Finkbeiner, B., Gieseking, M., Hecking-Harbusch, J., Olderog, E.: Model checking data flows in concurrent network updates. In: Automated Technology for Verification and Analysis - 17th International Symposium, ATVA. Lecture Notes in Computer Science, vol. 11781, pp. 515–533. Springer (2019), https://doi.org/10.1007/978-3-030-31784-3_30
10. Finkbeiner, B., Gieseking, M., Hecking-Harbusch, J., Olderog, E.: AdamMC: A model checker for Petri nets with transits against Flow-LTL. In: Computer Aided Verification - 32nd International Conference, CAV. Lecture Notes in Computer Science, vol. 12225, pp. 64–76. Springer (2020), https://doi.org/10.1007/978-3-030-53291-8_5
11. Finkbeiner, B., Gieseking, M., Hecking-Harbusch, J., Olderog, E.: Model checking branching properties on Petri nets with transits. In: Automated Technology for Verification and Analysis - 18th International Symposium, ATVA. Lecture Notes in Computer Science, vol. 12302, pp. 394–410. Springer (2020), https://doi.org/10.1007/978-3-030-59152-6_22
12. Finkbeiner, B., Gieseking, M., Olderog, E.: Adam: Causality-based synthesis of distributed systems. In: Computer Aided Verification - 27th International Conference, CAV. Lecture Notes in Computer Science, vol. 9206, pp. 433–439. Springer (2015), https://doi.org/10.1007/978-3-319-21690-4_25
13. Finkbeiner, B., Gölz, P.: Synthesis in distributed environments. In: 37th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS. LIPIcs, vol. 93, pp. 28:1–28:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2017), https://doi.org/10.4230/LIPIcs.FSTTCS.2017.28
14. Finkbeiner, B., Olderog, E.: Petri games: Synthesis of distributed systems with causal memory. Inf. Comput. **253**, 181–203 (2017), https://doi.org/10.1016/j.ic.2016.07.006

15. Förster, K., Mahajan, R., Wattenhofer, R.: Consistent updates in software defined networks: On dependencies, loop freedom, and blackholes. In: IFIP Networking Conference. pp. 1–9. IEEE Computer Society (2016), https://doi.org/10.1109/IFIPNetworking.2016.7497232
16. Gieseking, M., Hecking-Harbusch, J., Yanich, A.: AdamWEB: A Web Interface for Petri Nets with Transits and Petri Games (2020). https://doi.org/10.6084/m9.figshare.13089800
17. Gieseking, M., Olderog, E., Würdemann, N.: Solving high-level Petri games. Acta Informatica **57**(3-5), 591–626 (2020), https://doi.org/10.1007/s00236-020-00368-5
18. Hecking-Harbusch, J., Metzger, N.O.: Efficient trace encodings of bounded synthesis for asynchronous distributed systems. In: Automated Technology for Verification and Analysis - 17th International Symposium, ATVA. Lecture Notes in Computer Science, vol. 11781, pp. 369–386. Springer (2019), https://doi.org/10.1007/978-3-030-31784-3_22
19. Hecking-Harbusch, J., Tentrup, L.: Solving QBF by abstraction. In: Ninth International Symposium on Games, Automata, Logics, and Formal Verification, GandALF. EPTCS, vol. 277, pp. 88–102 (2018), https://doi.org/10.4204/EPTCS.277.7
20. McKeown, N., Anderson, T.E., Balakrishnan, H., Parulkar, G.M., Peterson, L.L., Rexford, J., Shenker, S., Turner, J.S.: Openflow: enabling innovation in campus networks. Comput. Commun. Rev. **38**(2), 69–74 (2008), https://doi.org/10.1145/1355734.1355746
21. Nielsen, M., Plotkin, G.D., Winskel, G.: Petri nets, event structures and domains, part I. Theor. Comput. Sci. **13**, 85–108 (1981), https://doi.org/10.1016/0304-3975(81)90112-2
22. Reisig, W.: Petri Nets: An Introduction, EATCS Monographs on Theoretical Computer Science, vol. 4. Springer (1985), https://doi.org/10.1007/978-3-642-69968-9
23. Sparkjava: http://sparkjava.com/ (2020)
24. University of Oldenburg: APT – Analyse von Petri-Netzen und Transitionssystemen. https://github.com/CvO-Theory/apt (2012)
25. Vue.js: https://vuejs.org/ (2020)
26. Vuetify: https://vuetifyjs.com/ (2020)
27. Wolf, K.: Petri net model checking with LoLA 2. In: Application and Theory of Petri Nets and Concurrency - 39th International Conference, PETRI NETS. Lecture Notes in Computer Science, vol. 10877, pp. 351–362. Springer (2018), https://doi.org/10.1007/978-3-319-91268-4_18