



**HAL**  
open science

# Mochy : a tool for the modeling of concurrent hybrid systems

Loïc Hélouët, Antoine Thébault

► **To cite this version:**

Loïc Hélouët, Antoine Thébault. Mochy : a tool for the modeling of concurrent hybrid systems. Petri Nets 2023 - 44th International Conference on Application and Theory of Petri Nets and Concurrency, NOVA University Lisbon, Jun 2023, Lisbonne, Portugal. pp.1-11, 10.1007/978-3-031-33620-1\_11 . hal-04311218

**HAL Id: hal-04311218**

**<https://inria.hal.science/hal-04311218>**

Submitted on 28 Nov 2023

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.



Distributed under a Creative Commons Attribution 4.0 International License

# Mochy : a tool for the modeling of concurrent hybrid systems

Loïc Hérouët and Antoine Thébaud

Inria, Rennes, France

**Abstract.** This paper introduces MOCHY, a tool designed for the modeling of concurrent systems with variants of stochastic, timed and hybrid Petri nets. Beyond modeling, the tool serves as a platform for fast simulation, and can be used for statistical verification of properties, controller testing, and learning of control rules. The targeted models are variants of stochastic and timed nets where tokens can be continuous quantities depicting trajectories of moving objects. The architecture of the tool is designed to be as adaptive as possible, and allow the redefinition of objects behaviors or transitions firing through the refinement of a few semantic rules. The framework also allows for the integration of controllers. For any model variant, MOCHY can perform fast simulation, and perform statistical verification, evaluate some quantitative properties of a model, or learn control rules for reachability or quantitative objectives.

## 1 Introduction

This paper introduces a new tool called MOCHY, tailored for the modeling of systems with timed variants of Petri nets, and for fast simulation. The origin of MOCHY stems from the need of fast simulation tools to test traffic management policies for metro networks [Kec19]. We rapidly came to the conclusion that transport networks had so many specificities that time Petri nets, timed Petri nets, or most of their variants were not adapted to the design of such models. First of all, even if models such as TPNs are Turing powerful, and can hence simulate most systems, using this expressive power in practice forces to loose the graphical and concurrent nature of nets and results in complex models that are hard to simulate, and cannot be understood by humans. A way to circumvent these issues were to tune existing models to obtain ad-hoc variants of Petri nets. However, this was not satisfactory either, because every transport network comes with its own traffic management policy, i.e. a light form of control that is used to mitigate effects of incidents and delays, that affects the semantics of the model. With this additional constraint, every transport networks can have its own ad-hoc semantics, and is hence a new kind of model. An example of variation point for instance is whether a metro network follows a fixed block policy allowing at most one train in each track segment, or a moving block policy that allows several trains in a segment provided they preserve safety headways.

The main principles of MOCHY are the following: we consider timed models that can depict trajectories of objects in a bounded environment, such as trains

40 on a track, cars on a road lane, manufactured objects on a conveyor,... The  
41 simulation scheme of the tool is designed to be as generic as possible. To reach  
42 this objective, the semantics of a model is given in a few generic operational rules  
43 depicting how the state of a network transforms upon occurrence of a discrete  
44 event, or when time elapses. These rules are repeated within a simulation loop,  
45 that may use a controller to select the next actions or delays allowed. Controllers  
46 also maintain a memory that can be used for further decisions.

47 This paper is organized as follows: we first describe the general architecture  
48 of the tool, the common features of models that can be simulated by MOCHY,  
49 the rules used by its adaptive semantics, and the way runs of MOCHY models  
50 are simulated. We then explain how MOCHY has been used to model a metro  
51 line in Rennes, and to learn a controller which aim is to help a metro recovering  
52 from a bunching situation. We conclude with related work, and perspectives on  
53 the future development of the tool.

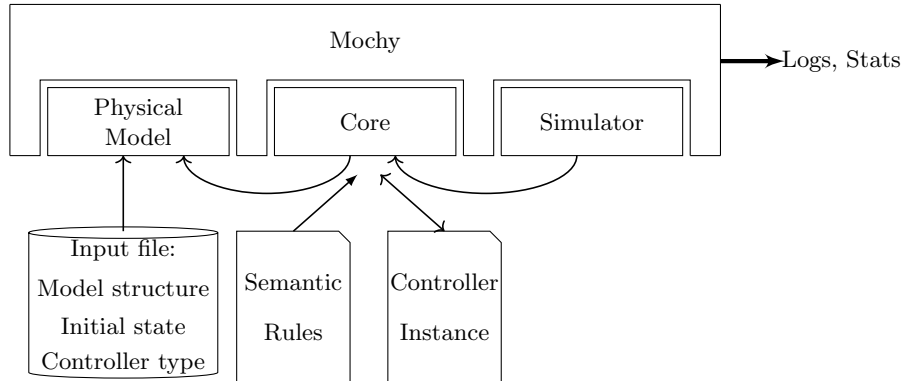
## 54 **2 Mochy Description**

55 MOCHY is designed to be as modular and adaptable as possible. Its architecture  
56 decouples semantics, interfaces, simulation scheme and control. This architecture  
57 allow redefinition of a semantics attached to a particular project, modification  
58 of a controller to guide choices of delays and actions during simulation, etc.  
59 This approach was proved particularly useful when considering models for metro  
60 networks: several train management policies have been implemented by simply  
61 changing the controller part of the tool.

### 62 **2.1 Architecture**

63 Mochy's architecture is composed of four main parts : an interface, and three  
64 modules describing a Physical Model, a Controller and a Simulator loop. The  
65 Physical Model contains the data structures needed to describe the Petri net  
66 variant that will be simulated, its initial configuration (mainly initial contents  
67 of its places) a net type, which will be used to load a class implementing the  
68 semantics rule of the net and a controller if needed. The physical model is loaded  
69 from an input project file. The Core part is composed of semantics rules and of  
70 a controller. This part is composed of classes that are loaded once the type of  
71 net is known from the input project file. The controller and semantic classes  
72 define the effects of possible actions and delays on configurations. The Simulator  
73 part implements a simulation loop, i.e. it handles semantics rules and control to  
74 animate a fixed number of semantics steps set from the interface. The interface  
75 allows interaction with these modules, by displaying current configuration of a  
76 loaded model, and providing access to the simulation functions of the tool and  
77 to standard functions (load, save...).

78



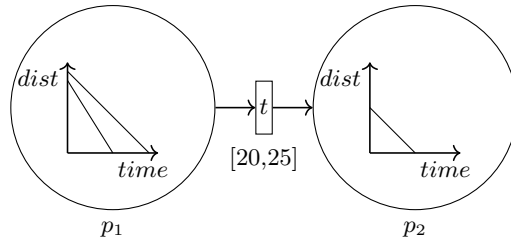
**Fig. 1.** Mochy architecture

79 **The Physical Model** The goal of MOCHY is to provide a generic tool for  
 80 the analysis and simulation of Petri nets variants. The models used as input for  
 81 MOCHY share some common characteristics, namely notions of events represented  
 82 by transitions of a net, notions of resources represented by places, and  
 83 the flows of resources consumed/produced by transitions, and timing information.  
 84 The common elements of MOCHY models are hence close to those of a  
 85 stochastic timed Petri-Net. The variation points are the contents of places, the  
 86 way they evolve over time, the firing rules of transitions, and the way a firing  
 87 of a transition affects place contents. The contents of places can be simple to-  
 88 kens, or more complex objects evolving in a multi dimensional space. We have  
 89 specialized this generic description to address models for transport networks.

90 *Structure of nets* The models used by MOCHY are variants of Petri nets with  
 91 time. They share common features, such as the notions of places, transitions,  
 92 flows, and time intervals. **Places** are contents holders for quantities that may  
 93 evolve over time. For instance, they can be used as containers for tokens, i.e.  
 94 integral numbers that are not affected by time elapsing, but only by discrete  
 95 transition firings. Conversely, places can represent a physical space such as track  
 96 portion with boundaries, where objects have trajectories. **Transitions** represent  
 97 classes of events. As usual in Petri nets, they have a preset, i.e. a set of places  
 98 depicting resources needed for an occurrence of the transition, a postset, i.e. a set  
 99 of places depicting resources impacted by the firing of a transition. A transition  
 100  $t$  can be triggered upon conditions that depend on time, and on the contents of  
 101 places in the preset and in the postset of  $t$ . These conditions and the effect of a  
 102 transition firing vary depending on the semantics rules.

103 The core structure of a net is hence a tuple  $\mathcal{N} = (P, T, A, I)$ , where  $P$  is  
 104 a set of places,  $T$  is a set of transitions representing events in a system,  $A \subseteq$   
 105  $(P \times T) \cup (T \times P)$  is a set of arcs connecting places to transitions, and transitions  
 106 to places. Map  $I : T \rightarrow \mathbb{Q} \times \mathbb{Q} \times DF$  associates a time interval  $[\alpha(t), \beta(t)]$  and a  
 107 distribution function  $f_t : [\alpha(t), \beta(t)] \rightarrow [0, 1]$  to each transition  $t \in T$ .

108 **Place contents** Standard timed variants of Petri nets (Time Petri nets, timed  
109 arc Petri nets, stochastic nets...) manipulate tokens that are put in places, and  
110 moved by each transition firing. In the models addressed by MOCHY, we allow  
111 for the definition of more complex place contents. For instance, we have used  
112 MOCHY to design models for metro networks, where trains can move at several  
113 speeds on a track segment as soon as they respect some safety headways. In one of  
114 the studied models, called *trajectory nets*, some places represent track segments,  
115 and their contents are trajectories of trains in a track portion depicted by *space-*  
116 *time* diagrams (see [Kec19] for a complete description of this model). Figure  
117 2 shows an example of a configuration of a trajectory net. Place  $p_1$  contains  
118 a space-time diagram with two train trajectories, depicting how the remaining  
119 distance to arrival evolves over time for each train. As one can imagine, as time  
120 elapses, the remaining distance to arrival decreases, which modifies the contents  
121 of places. When the remaining distance of a trajectory is zero, this trajectory  
122 is moved to another place by firing of a transition. The duration of the newly  
123 created trajectory is sampled from an interval attached to the transition that  
124 will consume it, and its initial distance is obviously the size of the track segment  
125 represented by the place entered. The semantics of the model also enforce safety  
126 of trains by allowing only trajectories preserving a sufficient headway between  
127 trains. This type of model was successfully implemented by instantiating the  
128 high-level semantics rules specified below.



**Fig. 2.** A simple Trajectory Net.

## 129 2.2 Semantics

130 The current state of a model is called a *configuration*. Configurations are denoted  
131 by  $C_1, C_2, \dots$ . The semantics of models designed with MOCHY are defined in  
132 terms of *timed moves* of the form  $C_i \xrightarrow{\delta} C_{i+1}$  which describe how a system  
133 evolves when a certain amount of time  $\delta$  elapses, and *discrete moves* of the  
134 form  $C_i \xrightarrow{ev} C_{i+1}$ , that describe how a system evolves when a discrete event  
135  $ev$  (usually the firing of a transition) occurs. We assume a sampling semantics,  
136 that is when a transition gets enabled, the time before its urgent firing is chosen  
137 according to the current configuration and never changed. In the context of  
138 time Petri nets for instance, this corresponds to sampling a duration  $\delta_t$  within a

139 interval  $[\alpha, \beta]$  for every newly enabled transition  $t$  when using a discrete move,  
 140 and considering  $t$  as urgent  $\delta_t$  time units later if it was not disabled before.

141 In a configuration  $C$ , a transition is *urgent* if it **has to** fire or be disabled  
 142 before some time elapses. In urgent semantics timed moves are forbidden in a  
 143 configuration  $C$  if  $C$  has urgent transitions. The notion of urgency may vary  
 144 from a variant of a model to another. In trajectory nets [Kec19], transitions are  
 145 urgent if a trajectory of an object has reached a border of the physical space  
 146 depicted by a place, and a transition is ready to move this object to another  
 147 part of the net. To allow for the specification of many models, the semantics of  
 148 a model in MOCHY is given by redefinition of two configuration transformation  
 149 rules and two functions to test place contents or check values of clocks:

- 150 –  $R_1(C, \delta)$  : depicts the transformation occurring in a configuration  $C$  when  $\delta$   
 151 time units elapse
- 152 –  $R_2(C, ev)$ : depicts the transformation occurring in a configuration  $C$  when  
 153 event  $ev$  occurs (mainly firing of a transition)
- 154 –  $F_1(C)$  : returns the time that can elapse before a transition becomes urgent
- 155 –  $F_2(C)$  : returns the list of transitions that are fireable in  $C$ .

### 156 2.3 Simulation

157 Configurations and transformation rules differ for every type of model, but provid-  
 158 ing functions  $R_1, R_2, F_1, F_2$  is sufficient to implement the simulation loop  
 159 proposed in Algorithm 1 next page. Creating a model with a new semantics in  
 160 MOCHY hence boils down to coding these rules and functions. Then simulation  
 161 of a model with MOCHY consists in repeatedly deciding which transition to fire  
 162 or which delay to elapse.

---

#### Algorithm 1 Rule-base Operational semantics for MOCHY

---

```

set a number of steps  $n$ 
set an initial configuration  $C_0$ 
 $i \leftarrow 1$ 
while  $i \leq n$  do
   $L = F_2(C)$ 
  if  $L = \emptyset$  then
     $\delta = F_1(C)$ 
    Use rule  $R_1(C, \delta) : C \xrightarrow{\delta} C'$ 
  else
    Use controller to choose  $t \in L$  and update controller's memory
    Use rule  $R_2(C, t) : C \xrightarrow{t} C'$ 
  end if
   $i++$ 
end while

```

---

163 Let us detail the role of controllers during during simulation. When sev-  
 164 eral transitions are fireable, a controller is used to choose which transition fires.

165 Controllers can be any program making such choice. The most basic controllers  
166 chose randomly a transition among firable ones. More involved controllers can  
167 be equipped with memory, with a schedule to follow, etc. and can implement  
168 complex strategies. We detail in Section 3 a controller designed to fix speed and  
169 dwell time of metros to recover from bunching situations.

## 170 2.4 Inputs-Outputs

171 MOCHY takes as input **project files** that contain : the class of model used by  
172 the project (it is a compiled Java class that implements the semantics rules), a  
173 description of the structure of an instance of the loaded model: the places, transi-  
174 tions, and flow relations of the net, the time intervals attached to transitions and  
175 the associated distributions. Depending on the specialization of the model, the  
176 project file can also provide pointer to additional features : a controller, sched-  
177 ules, etc. ). This approach allowed for the modeling of metro networks equipped  
178 a traffic management algorithm designed to adhere to a timetable.

179  
180 Once a file is loaded, users can play with the specification in an interactive  
181 manner, or run a simulation for several steps, following the simulation procedure  
182 of Algorithm 1. At each stop of the simulator, the contents of places is displayed.  
183 The net can be reset to its initial configuration at every stop. During simulation,  
184 the tool generates **logs**, remembering dates and transitions fired, and a selection  
185 of a few variables for further statistics computed from the saved logs.

## 186 2.5 User Interface

187 Figure 3 describes the main window of MOCHY. It is divided in three horizontal  
188 panels. The top of the window shows the structure of the simulated net, and its  
189 current configuration. The middle of the window in a control panel for simula-  
190 tion, that allows launching a simulation for a given number of steps, letting time  
191 elapse, or firing a single transition. The bottom part of the interface contains  
192 two consoles to display simulation logs, warnings and statistics such as the mean  
193 duration of a simulation after an extensive simulation campaign, or any measure  
194 of a performance indicator.

195

## 196 3 Case Studies

197 MOCHY has been tested successfully on several models of metro networks, with  
198 various semantics. In this section, we present an first experiment that successfully  
199 used MOCHY to test accuracy of a traffic management algorithms, and a second  
200 one that allowed to train a controller to avoid train bunching.

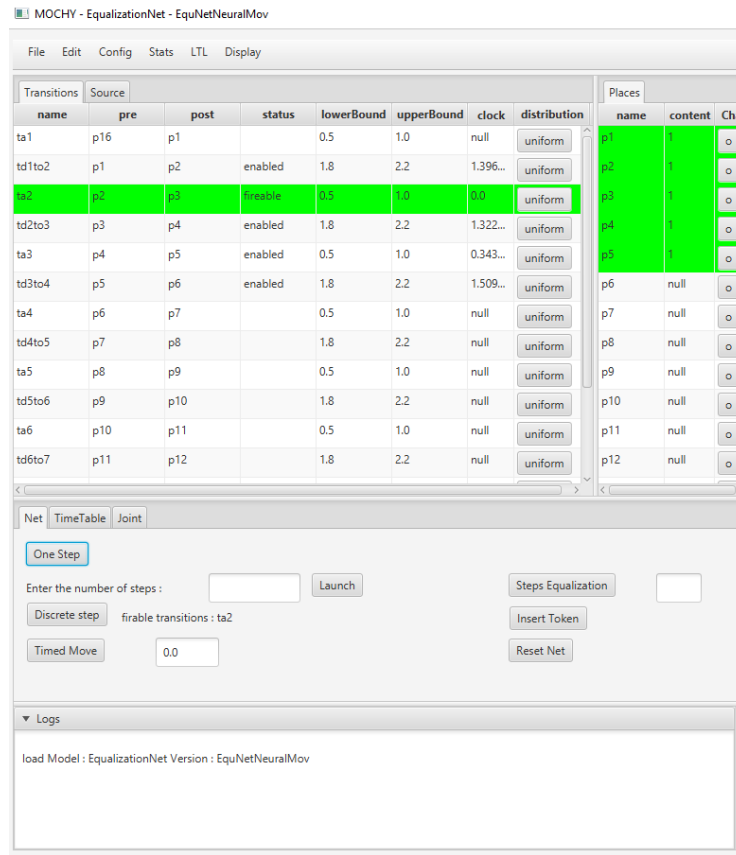


Fig. 3. The main window

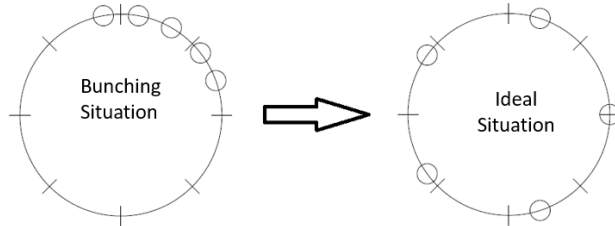
201 **A Metro line in Rennes** The Metro line A in Rennes has 15 stations and a  
 202 length of 9 kilometres. The model developed with MOCHY uses a controller  
 203 that implements a traffic management policy which goal is to give dwell duration  
 204 and speed advices to trains in order to adhere to a given timetable. The timetable  
 205 describes planned operations for 4 hours, and contains dates for more than 3000  
 206 events. The net model is composed of 56 places, 63 transitions. MOCHY showed  
 207 good performance for the simulation of this metro network : simulation of 10  
 208 runs (i.e. sequences of at least 3000 discrete moves) can be performed in 1.5  
 209 minutes on an average laptop.

210 **Regulation by Equalization** The good performance of the tool allows for its  
 211 use for applications that require intensive simulation campaigns, such as statistical  
 212 model-checking or learning techniques. We have used MOCHY to train a neural  
 213 network in charge of controlling a metro network to recover from a bunching



214 situation. Bunching is a situation where all trains are not well distributed on a  
 215 network, causing long periods without service in stations, followed by arrival of  
 216 many trains in a short amount of time. This situation is depicted in Figure 4.

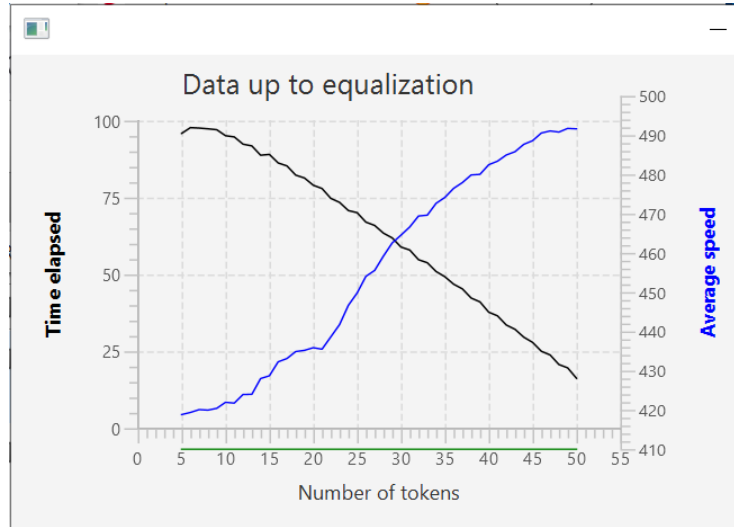
217 We have considered a simple network, namely a loop of 20 kilometers, mod-  
 218 eled by a net with 60 places and 60 transitions equipped with a controller that  
 219 aims at equalizing distances between trains in the network (see [HFT22] for  
 220 more details on the experiment). Several traffic management algorithms have  
 221 been tested. For each of them, the decision taken was to choose an appropriate  
 222 speed and dwell time for trains stopped at a quay. The first tested approach was  
 223 based on optimization of a quadratic function considering distances of a train  
 224 wrt to its predecessor and its successor. The second approach tested was a neural  
 225 network, with similar parameters as input, and trained with a genetic learning  
 226 approach [PHU05], selecting mutations improving the controller’s statistics dur-  
 227 ing intensive simulation campaigns realized with MOCHY. The tool allowed to  
 228 simulate runs of duration of up to 2 hours for fleet sizes ranging from 5 to 50  
 229 trains in less than 15 seconds. Both controller types were evaluated w.r.t the  
 230 time needed to return to an equilibrate distribution of trains and to the aver-  
 231 age speed of trains after this equalization. Figure 5 shows the performance of  
 232 a neural network controller trained with MOCHY. The black curve represents  
 233 the average time needed to recover from the worst possible bunching situation,  
 234 and the blue line the average speed (in meters/min, with an objective of a 500  
 235 m/min) once the equalization is performed.



**Fig. 4.** From a bunching situation to a good distribution of trains in a Metro network.

## 236 4 Conclusion

237 The MOCHY toolbox can be freely downloaded at the following url:<https://adtmochy.gitlabpages.inria.fr/mochy/>. The available packages contain implementa-  
 238 tion of semantic rules for several variants of nets, including waiting nets [HA22]  
 239 and trajectory nets [Kec19] and examples of models for metro networks. Future  
 240 distributions will include a statistical model checker for Signal LTL, and Machine  
 241 Learning techniques to train controllers.  
 242



**Fig. 5.** Time to equalization guided by a neural network for varying fleet sizes.

243 Several tools dedicated to Petri nets and their variants exist (see [PNT] for  
 244 an extensive list). Some of them can handle time, complex firing rules or hybrid  
 245 variables needed to model trajectories of objects. Many tools are dedicated to  
 246 time(d) Petri nets and their variants. Romeo [LRST09] is a verification tool for  
 247 time Petri nets (TPNs). In addition to the standard urgent semantics of TPNs,  
 248 ROMEO allows for the specification of ad-hoc firing rules, read/inhibitor arcs,  
 249 and for parameters synthesis. ORIS [CPV22] is close to ROMEO, but is tailored  
 250 for transient analysis of stochastic timed Petri nets. Tapaal [BS09] targets ver-  
 251 ification of timed arcs Petri nets. It allows inhibitor/read arcs. TINA [BV06] is  
 252 an analyser for TPNs extended with read arcs, inhibitor arcs, open intervals and  
 253 Data. Several other tools can perform simulation of timed and stochastic vari-  
 254 ants of nets. ARP analyzes and simulates nets where transitions are attached an  
 255 interval and a distribution over this interval. Petrisim allows for the simulation  
 256 of Petri nets with delay between token production and consumption.

257 Some tools target net variants with colors, a way to introduce variables and  
 258 data in nets. Alpha/Sim allows for the simulation of stochastic, timed, attributed  
 259 or colored Petri nets. Great SPN is a tool for generalized (colored) Stochastic  
 260 Petri nets, that allows for timed simulation. ExSpect is dedicated to the design  
 261 of business processes and is formally based on colored Petri nets. CPN tools  
 262 allows for the definition of colored High-level nets and nets with time, and can  
 263 simulate them to analyse performance of the modeled systems. PnetLab is a tool  
 264 dedicated to the control of High-level coloured Petri net and allows in addition  
 265 side management of time to test scheduling strategies when transitions are given  
 266 a service time. TimeNET is a tool for the modelling and analysis of stochastic

267 Petri nets with non-exponentially distributed firing times. It supports graphical  
268 modeling of uncolored and colored Petri nets as well as Markov chains. Numerous  
269 performance evaluation and structural analysis algorithms are available as well  
270 as an interactive token game.

271 Modeling of transport systems calls for mechanisms that can encode objects  
272 movements, road/track bounds, safety distances. Obviously, this can be simu-  
273 lated by Turing powerful models such as all colored variants, time Petri nets,  
274 etc. but at the cost of low-level encodings of objects movements, that are more  
275 intuitively captured by continuous or hybrid variants of nets. QPME is a tool  
276 that implements Queuing Petri nets. This type of nets/tools is of great in-  
277 terest for the design and analysis of transport systems, but does not allow for  
278 the modeling of constraints among the moving objects. As for queuing theory,  
279 analyses lead either to optimistic or pessimistic performance evaluation wrt the  
280 actual behavior of a train network. Batch Petri nets [Dem01], or Differential  
281 Petri nets [HMM09], and the tool Hisim can handle mixed discrete and contin-  
282 uous tokens/ transitions, where places contain quantities that evolve according  
283 to differential equations. Discrete transitions firings have the usual semantics,  
284 and continuous firings are allowed when place contents exceed some threshold,  
285 and moves some quantities of token per time unit. Hisim simulates hybrid nets  
286 via a simulation loop that : fires immediate transitions, computes the next event  
287 date, progress time to this date, and iterates. Simulation in MOCHY is based  
288 on a similar simulation loop.

289 Fluid-survival-tool [PRHG14] considers Hybrid Petri nets to model systems  
290 with discrete and continuous quantities, and provides tools to compute the prob-  
291 ability to be in a given state at a certain time, or to verify Stochastic Timed  
292 Logic. Time is handled by attaching constant firing times to discrete transi-  
293 tions, distributions on firing times to stochastic transitions, and firing speed to  
294 continuous ones.

## 295 References

- 296 [BS09] K.Y Byg, J.and Jørgensen and J. Srba. TAPAAL: editor, simulator and  
297 verifier of timed-arc petri nets. In *Proc. of ATVA 2009*, volume 5799 of  
298 *LNCS*, pages 84–89, 2009.
- 299 [BV06] B. Berthomieu and F. Vernadat. Time petri nets analysis with TINA. In  
300 *Proc. of (QEST 2006)*, pages 123–124. IEEE Computer Society, 2006.
- 301 [CPV22] L. Carnevali, M. Paolieri, and E. Vicario. The ORIS tool: app, library, and  
302 toolkit for quantitative evaluation of non-markovian systems. *SIGMET-*  
303 *RICS Perform. Evaluation Rev.*, 49(4):81–86, 2022.
- 304 [Dem01] I. Demongodin. Generalised batches petri net: Hybrid model for high speed  
305 systems with variable delays. *Discret. Event Dyn. Syst.*, 11(1-2):137–162,  
306 2001.
- 307 [DK06] I. Demongodin and N.T. Koussoulas. Differential petri net models for indus-  
308 trial automation and supervisory control. *IEEE Trans. Syst. Man Cybern.*  
309 *Syst.*, 36(4):543–553, 2006.

- 310 [HA22] L. Hélouët and P. Agrawal. Waiting nets. In Luca Bernardinello and Laure  
311 Petrucci, editors, *PETRI NETS 2022*, volume 13288 of *LNCS*, pages 67–89,  
312 2022.
- 313 [HFT22] L. Hélouët, E. Fabre, and A. Thébault. Optimization of traffic management  
314 with learning machines. *hal-03777459*, 2022.
- 315 [HMM09] F. Hamdi, N. Messai, and N. Manamanni. Design of switched observer using  
316 timed differential petri nets: A dwell time approach. In *Proc. of European  
317 Control Conference, ECC 2009*, pages 4641–4646. IEEE, 2009.
- 318 [Kec19] Karim Kecir. *Performance Evaluation of Urban Rail Traffic Management  
319 Techniques. (Évaluation de Performances pour les Techniques de Régulation  
320 du Trafic Ferroviaire Urbain)*. PhD thesis, University of Rennes 1, France,  
321 2019.
- 322 [LRST09] D. Lime, O.H. Roux, C. Seidner, and L-M Traonouez. Romeo: A parametric  
323 model-checker for petri nets with stopwatches. In *Proc. of TACAS 2009*,  
324 volume 5505 of *LNCS*, pages 54–57, 2009.
- 325 [PHU05] P.P. Palmes, T. Hayasaka, and S. Usui. Mutation-based genetic neural net-  
326 work. *IEEE Trans. Neural Networks*, 16(3):587–600, 2005.
- 327 [PNT] Petri nets tools database quick overview. [https://www.informatik.uni-  
328 hamburg.de/TGI/PetriNets/tools/quick.html](https://www.informatik.uni-hamburg.de/TGI/PetriNets/tools/quick.html). Accessed: 2023-02-01.
- 329 [PRHG14] B.F. Postema, A. Remke, B.R. Haverkort, and H. Ghasemieh. Fluid sur-  
330 vival tool: A model checker for hybrid petri nets. In *Proc. of Measurement,  
331 Modelling, and Evaluation of Computing Systems and Dependability and  
332 Fault Tolerance - 17th International GI/ITG Conference, MMB & DFT  
333 2014*, volume 8376 of *LNCS*, pages 255–259, 2014.