Petri-Nets@Run.Time: Handling Uncertainty during Run-Time Adaptation using Digital Twins

Michael Köhler-Bussmeier¹, Heiko Rölke²

¹Hamburg University of Applied Sciences, Berliner Tor 7, D-20099 Hamburg ²University of Applied Science of the Grisons, Pulvermühlestrasse 57, CH-7000 Chur

Abstract

In this paper we study systems containing uncertainty about quantitative aspects, like delays etc. System design for such systems faces the problem that usually the system's initial configuration is chosen in a sub-optimal way due to these uncertainties. An adaptive system will adjust its configuration whenever it obtains new knowledge about the uncertain aspects.

In our setting we like to derive the optimal configuration from the system's model. Therefore our models have to include a description of the uncertainties as well. To handle the change of knowledge at run-time, we integrate the model into the run-time system: models@run.time

Adaptation of the configuration often requires to evaluate different variations of the model, which arise from the uncertainties. To evaluate the impact of different configurations, we simulate a *digital twin* of the system.

Therefore, we need a formalism that allows for an easy simulation of models. To simplify the approach, we would like to use the same formalism for the simulator and the system model, i.e. we use a formalism that is able to execute itself as a sub-system. As our system model is based on Petri nets we use the reflexive approach of HORNETS, which follow the *Nets-within-Nets* principle.

In this paper, we will specify a HORNET model to handle digital twins during the planning phase of the well-known MAPE-loop (monitor-analyse-plan-execute). A simple scenario based on stochastic workflow Petri nets will serve as a proof-of-concept.

Keywords

Adaptation, digital twin, Hornets, MAPE-loop, models@run.time, nets within nets, uncertainty

1. Uncertainty, Adaption, and Models@run.time

Process models are usually based on quantitative aspects – like duration, delays etc. While the general process structure is often well-understood, there are often uncertainties about the quantitative aspects. Assume that we have to configure the system w.r.t. these quantitative aspects, e.g. we have to define a selection logic that minimizes total delay. Due to the uncertainties in the parameters it is highly likely that the chosen configuration is sub-optimal and has to be adjusted whenever our knowledge about these quantitative aspects improves over the running time via observations.

☆ michael.koehler-bussmeier@haw-hamburg.de (M. Köhler-Bussmeier); heiko.roelke@fhgr.ch (H. Rölke)
 ♦ https://www.haw-hamburg.de/michael-koehler-bussmeier (M. Köhler-Bussmeier);

- D 0000-0002-3074-4145 (M. Köhler-Bussmeier)
- © 0 2023 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).
 - CEUR Workshop Proceedings (CEUR-WS.org)

PNSE'23, International Workshop on Petri Nets and Software Engineering, 2023

https://www.fhgr.ch/personen/person/roelke/ (H. Rölke)

Therefore we advocate to follow the models@run.time approach and use the system model as a part of our running system that describes the knowledge about it (K). This knowledge is used in the following way: We monitor our system at run-time (M); observations are analysed whether we have to correct our estimation of the quantitative parameters (A); whenever we adjust, this may lead to (re-)planning of our configuration (P); the new configuration is rolled out in the execution step (E). These four steps constitutes the phases of the well-known MAPE-K execution loop [1].



Figure 1: MAPE-K Loop: monitor-analyse-plan-execute

Adaptation of the configuration often requires to evaluate different variations of the model ranging over the regions of uncertainty. Here, we evaluate by simulating variations of a *digital twin* of the system. So we need a formalism that allows for an easy simulation of models. We advocate a reflexive approach, i.e. we use a formalism that is able to execute itself as a sub-system.

As we model our processes with Petri nets (including time and stochastic operations), we follow the *nets-as-tokens* approach, also known as Nets-within-Nets [2]. Since our processes have an explicit structure, relying on usual operators (like sequence, and-split, or-choice, etc.) we like to use a formalism that allows to modify also the structure of the net-tokens at run-time. This formalism is called HORNETS [3].

The paper has the following structure: In Section 2 we introduce our basic process model: probabilistic Workflow Nets (PWFN), which are a special variant of workflow nets [4]. Section 3 describes how we incorporate adaption of PWFN into our systems using a MAPE-like run-time loop. Section 4 then shows how we could specify the adaptation concept within the formalism of HORNETS – a special nets-within-nets approach. The HORNET-model uses the PWFN as a *digital twin* to evaluate variants of our PWFN to enable the planning part of the MAPE loop. Especially, we elaborate more on this evaluation in the context of huge parameter spaces, especially with respect to the *exploration vs exploitation* trade-off. In Section 5, we evaluate our approach in a case study. Here, we study the model of a course of study and its success. The contribution closes with a summary, ongoing research, and a conclusion.

2. The Basic Process Model: PWFN

As our basic process model we use the well-known Workflow Nets [4]. A workflow net is a Petri net N = (P, T, F) with a set of places P, a set of transitions T, and a flow relation $F \subseteq P \times T \cup T \times P$ with a designated place for the beginning (with an empty preset) and one for the end of the workflow (with an empty postset). Additionally, all nodes must lie in between these special places. We assume that the reader is familiar with the basic notions of Petri nets.

A Probabilistic View on Conflicts Our model contains probabilities for conflicting transitions, like e.g. in xor-choices. In the following, we use Free-Choice nets (FCN) [5], i.e. we have that whenever two transitions are in conflict for some token then they have the same preset: ${}^{\bullet}t_1 \cap {}^{\bullet}t_2 \neq \emptyset \Longrightarrow {}^{\bullet}t_1 = {}^{\bullet}t_2$ because for FCN the notions of structural and dynamic conflict coincide. The *conflict set* of a transition is $C(t) = ({}^{\bullet}t)^{\bullet}$. The conflict sets constitute a partition $\mathcal{T} = \{T_1, \ldots, T_n\}$ of T.

The choice between different transitions in the conflict set annotated by a stochastic distribution $\rho_{C(t)}: C(t) \to \mathbb{R}^{\geq 0}$, i.e. we have $0 \leq \rho_{C(t)}(t) \leq 1$ and $\sum_{t \in C(t)} \rho_{C(t)}(t) = 1$.

Semantics of Probabilistic Choices The basic semantics of parametric workflow nets (enabled transitions, successor markings etc.) is given the standard way – ignoring the additional stochastic information. However, we can extend the reachability graph into a Markov chain: Each marking m enables a set of transitions $En_m(t)$ that is the disjoint union of k conflict sets T_i due to the free-choice property: $En_m(t) = T_1 \cup \ldots \cup T_k, T_i \in \mathcal{T}$. For each conflict set T_i we have a stochastic distribution ρ_{T_i} , which we can convert into a stochastic distribution over $En_m(t)$ via normalisation:

$$\rho_{En_m(t)}(t) = \frac{1}{k} \cdot \rho_{T_i}(t) \quad \text{for the unique } T_i \text{ s.t. } t \in T_i$$

Note, that our model is quite similar to Probabilistic Timed Automata [6]. The main difference is that for SWFN we allow concurrency as we like to model *multi-party* interaction.

Probabilistic Workflow Nets: Internal vs External Choice Each choice for a conflict set is either internal or external: $\mathcal{T} = \mathcal{T}_{int} \uplus \mathcal{T}_{ext}$. For the internal choice the system itself controls between the alternatives, while for the external choice the environment does.¹

The distribution of environmental choices are uncertain. A concrete assignment of distribution functions to the set of external choices is called a *scenario* and is denoted as α . We have $\alpha : \mathcal{T}_{ext} \to (\bigcup \mathcal{T} \to \mathbb{R}^{\geq 0})$ where $\alpha(C) : C \to \mathbb{R}^{\geq 0}$ for each $C \in \mathcal{T}_{ext}$ and undefined otherwise. From this we have to distinguish the *internal* parameters, which we denote by $\beta : \mathcal{T}_{int} \to (\bigcup \mathcal{T} \to \mathbb{R}^{\geq 0})$ where $\beta(C) : C \to \mathbb{R}^{\geq 0}$ for each $C \in \mathcal{T}_{int}$ and undefined otherwise. Let *Conf* denote the set of all configurations.

The two distribution assignments could be combined into one single an assignment $\rho : \mathcal{T} \to (\bigcup \mathcal{T} \to \mathbb{R}^{\geq 0})$: We define $\rho(C) := \alpha(C)$, if $C \in \mathcal{T}_{ext}$ and $\rho(C) := \beta(C)$, if $C \in \mathcal{T}_{int}$.

¹In the case of coloured WF-Nets the internal choice could be expressed more appropriate by a function which maps the actual binding to a selection of one alternative.

Given a scenario α and a configuration β we denote the resulting Petri net as $N_{\alpha}(\beta)$. In our setting we control the distribution of internal choices and we have to configure them in way that optimises our model wrt. some evaluation criteria.² Whenever N is a workflow net we call $N_{\alpha}(\beta)$ a probabilistic workflow net (PWFN).

Example Figure 2 shows the workflow to perform a study of *n* courses in parallel. The external uncertainty is the failure probability α of examinations. Students react on the examination failure rate (given as an external parameter) by adjusting their probability β to take additional preparation time and skip the examination (the internal parameter).

We make the assumption that the failure probability decreases with the number t of extra preparation courses: $\alpha_t = \alpha^{(li \cdot t)}$ where α is the basic failure probability and li > 0 is some form parameter (the learning impact).



Figure 2: The Stochastic Workflow Net for the Examination Process

The Multi-Party Setting Note, that the distinction between internal and external choices assume a single agent acting in its environment. In a multi-party setting we have different agents $G = \{ag_1, \ldots, ag_n\}$ where each agent controls only a part of the workflow. This extension is similar to that from LTL/CTL to ATL [8]. Each agent selects its own configuration β_i while the environment α consists of the configurations of all the other agents, i.e. $\alpha = (\beta_1, \ldots, \beta_{i-1}, \beta_{i+1}, \ldots, \beta_n)$.

Therefore, each agent observes the actions taken by the other agents to predict their underlying selection logic, i.e. the decision tree and the succeeding random variables. Once an agent

²Note, that this formulation has a analogy to game models [7] as we have one player, the environment α , an a player β for the model and we try to chose an β that wins the game.

has an acceptable fair prediction of the behaviour of others, he can try to optimize its own selection process, i.e. its own strategy, to maximise its own benefit. So, on one hand, the agent likes to exploit his own knowledge to maximise the benefit; but, on the other hand, he has to explore the set of actions to obtain a better estimation of the other agents' logic. In general, an agent has to balance these two processes of exploration and exploitation. For more details cf. [9, 10].

Adding Colour: The Adapt-OR We like to discuss an extension for coloured tokens in PWFN. Assume a situated choice: Agents observe a sensor input $\vec{x} = (x_1, \ldots, x_n)$ and have to select one action from a set $\{a_1, \ldots, a_m\}$. Assume that agents learn to discriminate different situations via a decision tree. Assume further, that for each different situation the agent selects the action based on an internal decision logic. We assume that the decision logic is unknown from the outside, so we model it by a random variable.

This selection process is modelled as the Petri Net fragment of Fig. 3. The fragment shows a block where an agent observes the given situation by the sensor values x. Here, the variable x is compared against the constant 10. Depending on x the agent considers himself to be in *situation*₁ or in *situation*₂. Each situation leads to a choice between action a_1 and a_2 , but whenever the agent considers himself to be in *situation*₁ the decision differs from that in *situation*₂. We assume that from an internal perspective the action is chosen in a deterministic way, but from an observer's perspective the decision process can be described as a random selection in the ratio of 20 to 80 in *situation*₁, while it is 40 to 60 in *situation*₂.



Figure 3: Adapt-OR = XOR + Decision Tree + Probabilistic Decision Logic

3. Adaption of PWFN

In general we assume that our system is not exactly N_{α} – it can be any model $N_{\alpha'}$ with $\alpha \approx \alpha'$, where $\alpha \approx \alpha'$ denotes that α and α' are (in some sense) similar. We assume that a level of uncertainty induces a set of similar settings for α , i.e. the set $C(\alpha) = \{\alpha' \mid \alpha \approx \alpha'\}$. Usually, a higher uncertainty in α increases the size of $C(\alpha)$. We assume that $C(\alpha_0)$ is weighted, i.e. we have a weight $p: C(\alpha_0) \to \mathbb{R}^{\geq 0}$ such that $\sum_{\alpha \in C(\alpha_0)} p(\alpha) = 1$.³ We can extend this function to a function $p: Conf \to \mathbb{R}^{\geq 0}$ to all configurations by setting $p(\alpha) = 0$ for $\alpha \notin C(\alpha_0)$. When we speak of similar scenarios α and α' this is meant w.r.t. this probability function.

3.1. Analysis of Execution Data w.r.t. Change of Uncertainty

New observations lead to a new estimation of α . So, we write α_t where t denotes the time. Observations also lead to a recalculation of the uncertainty. Thus we have a time dependent definition of similarity: $C_t(\alpha)$.

New observations lead to (1) a new estimation α_{t+1} and (2) to a new calculation of uncertainty, which induces a new similarity class $C_{t+1} := C(\alpha_{t+1})$. To decide whether we have to adapt we have to compare C_t and C_{t+1} . We define the overlap of the current class C_t with its successor C_{t+1} as the weighted sum of squares:

$$overlap_t := 1 - \sum_{\alpha \in Conf} p_t(\alpha) \cdot (p_{t+1}(\alpha) - p_t(\alpha))^2$$
(1)

The sensitivity wrt. the overlap is controlled by a meta-parameter δ that defines how eagerly the system responds:

- Whenever overlap_t ≥ δ, i.e. these two classes have a high overlap, then is reasonable that β_t is also good configuration for C_{t+1}, i.e. β_{t+1} := β_t.
- Whenever $overlap_t < \delta$, we trigger the adaption phase, i.e. we have to recompute β_{t+1} for C_{t+1} .

3.2. Adaption of Configuration β under Uncertainty

The design goal is to select the optimal configuration β^* for a given environment model N_{α} with respect to an evaluation function. Let us assume for a moment that we have perfect knowledge about α , i.e. there is no uncertainty; then the choice of the optimal configuration β^* is straightforward:

$$\beta^* = \operatorname{argmax}_{\beta} \left(eval(N_{\alpha}(\beta)) \right) \tag{2}$$

However, as our knowledge about the environment model α is rather bounded, we cannot assume, that system is exactly N_{α} – it can be any model $N_{\alpha'}$ with $\alpha \approx \alpha'$.

Major Challenge: We like to select the value for β that maximises the evaluation for a large fraction of the similar models $N(\alpha')$. So, we have to balance both aspects – quantity and quality (or: large fraction and maximal evaluation) – somehow.

In the following, we propose a selection mechanism based on rankings. We define a weighted score to select the best configuration: For each $\alpha \in C(\alpha_0)$ we evaluate the reward of each configuration $\beta \in Conf$ and order them by calculating their rank $rank_{\alpha}(\beta)$ for a fixed α .

Then, we must combine the ranking $rank_{\alpha}(\beta)$ for all the scenarios α into one single evaluation $eval(\beta)$. This combination has to balance *large fraction vs maximal evaluation* somehow.

39

³For example, if similarity is measured with some distance metrics we could use a *K*-nearest neighbours (KNN) algorithm to compute this set $C(\alpha)$ and we define the probability proportional to the distance.

Here, we assign values to ranks: The first rank is assigned 1 point, a second rank γ points, a third rank γ^2 points, etc. For $\gamma \in [0, 1]$ we define the evaluation of β as:

$$eval_{\gamma}(\beta) := \sum_{\alpha \in C(\alpha_0)} p(\alpha) \cdot \gamma^{(rank_{\alpha}(\beta)-1)}$$
 (3)

The weight function *p* guarantees that the evaluation function is always defined:

$$\sum_{\alpha \in C(\alpha_0)} p(\alpha) \cdot \gamma^{(rank_{\alpha}(\beta)-1)} \le \sum_{\alpha \in C(\alpha_0)} p(\alpha) = 1$$

We use this weighted score to select the best configuration:

$$\beta^* = \operatorname{argmax}_{\beta} \left(eval_{\gamma}(\beta) \right) \tag{4}$$

Here, γ is a meta-parameter that controls the balance: Assume e.g. that $\gamma = 1/2$, then one scenario with first rank counts as much as two scenarios with rank two or four scenarios with rank four. For a smaller value, e.g. $\gamma = 1/4$, then one scenario with first rank counts as much as four scenarios with rank two or 16 scenarios with rank four. Therefore, a smaller γ prefers higher rankings in a small spectrum of scenarios over a more broader spectrum with lower rankings. For $\gamma = 0$ we only consider the winning β for a scenario, i.e. the evaluation is proportional to the number of scenarios won by β . On the other hand, for $\gamma = 1$ all β have the same evaluation: $eval_{\gamma}(\beta) = \sum_{\alpha \in C(\alpha_0)} p(\alpha) \cdot \gamma^{(rank_{\alpha}(\beta)-1)} = \sum_{\alpha \in C(\alpha_0)} p(\alpha) \cdot 1 = 1$. In this case the choice of β^* is purely random.

In the following we propose a simulation-based way of approximating this configuration β^* by exploring the run-time model (i.e. the digital twin). The main idea is to sample the value, i.e. to compute the rankings only for *some* configurations and for *some* scenarios.

4. Adaptation of Digital Twins modelled with HORNETS

We use HORNETS [3] to specify the simulation of digital twins as a part of the MAPE-loop. A HORNET is a Petri net, called the system net, that contains Petri nets as tokens, which are called object nets or net-tokens. We will not present the formal theory behind HORNETS, and use them rather informal in the following. Instead we will illustrate the idea giving a simple example taken from [3].

Example We consider a HORNET with two workflow nets N_1 and N_2 as tokens – cf. Figure 4. To model a run-time adaption, we combine N_1 and N_2 resulting in the net $N_3 = (N_1 || N_2)$, i.e. the parallel composition. This modification is modelled by transition t of the HORNETS in Fig. 4. In a binding α with $x \mapsto N_1$ and $y \mapsto N_2$ the transition t is enabled. Assume that (x || y) evaluates to N_3 for α . If t fires it removes the two net-tokens from p and q and generates one new net-token on place r. The net-token on r has the structure of N_3 and its marking is obtained as a transfer from the token on v in N_1 and the token on s in N_2 into N_3 . This transfer is possible since all the places of N_1 and N_2 are also places in N_3 and tokens can be transferred in the obvious way.

Here, the system net is used to describe the MAPE-loop and all $N_{\alpha}(\beta)$ for all α and β are used as object nets.



Figure 4: Modification of the net-token's structure

4.1. First- and Second-Order Adaption

HORNETS uses algebraic operations on the structure of net-tokens, i.e. the formalism allows to change the net-topology at run-time. This is very useful to model block-structured nets (like e.g. [11]) built upon atomic actions a, sequences: $N_1 \cdot N_2$, parallel composition (i.e. and-split): $N_1 || N_2$, xor-choice: $N_1 \oplus N_2$, etc. Given this possibility it is obvious, that we have two ways to introduce adaptation here:

- 1. First order adaption: change of configuration β
- 2. Second order adaption: change/refinement of N

Obviously, second order adaption has a much bigger impact on the change of the system when compared to the first order changes where the fundamental workflow structure remains unchanged.

HORNET-Model of First-Order Adaption Whenever we adjust our environment model from α_1 to α_2 , we change our run-time model from $N_{\alpha_1}(\beta)$ to $N_{\alpha_2}(\beta^*)$ where β^* is the new configuration adapted to α_2 . The complete system net describing the MAPE-Loop is shown in Figure 5. We adapt by replacing the net $N_{\alpha_2}(\beta)$ with $N_{\alpha_2}(\beta^*)$, i.e. we change the configuration β .

HORNET-Model of Second-Order Adaption For the second-order adaption we also change the net itself, e.g. we may replace an action by a complete workflow or replace an and-split a || b by an sequence a; b etc. We adapt by replacing the net N with the modified net N'. This extends the MAPE-Loop from Figure 5 by the transition modify model structure via (online) model mining. The resulting loop is shown Figure 6.

A modification of the model structure at run-time demands for a *meta-level model* that specifies which transformations are allowed under which circumstances. We study this meta-level specification in ongoing work in the context of our specification framework for multi-agent organisations: SONAR [12]. SONAR organisation models [13] describe a whole possibility space how tasks may be handled within the multi-agent system (MAS). For each task a SONAR-MAS generates a team of agents $G = \{ag_1, \ldots, ag_n\}$ and a workflow net D describing the interaction to handle the task. This mapping is not fixed; usually there is a set of possible teams and the



Figure 5: System Net describing the MAPE-Loop



Figure 6: MAPE-Loop with Structural Adaptation

selection is based on current state of the MAS, including parameters like load of agents etc. Whenever the state changes usually the team formation changes, too, and therefore the same task will generate a different team G' and a different team workflow net D'.⁴ The definition of the meta-level is beyond the scope of this paper.

4.2. Evaluating Digital Twins: Exploration vs. Exploitation

In the following we propose a simulation-based way of approximating the optimal configuration β^* by exploring the run-time model (i.e. the digital twin). The main idea is to compute scores only for a small set of scenarios and a subset of configurations.

When selecting the best configuration β^* to react on the environment α we are facing two problems: Firstly, usually it is not easy to determine the reward of a given configuration directly. Instead of calculating the value, we advocate for a simulation-based approach, i.e. we evaluate

⁴Note, that this naturally induces a *distributed* MAPE-loop. Some authors reserve the term *adaption* for conceptually centralised systems, and speak of *self-organisation* in case of distributed systems. We do not make this differentiation.

runs of $N_{\alpha}(\beta)$. Secondly, in most cases the models are too complex to perform an exhaustive evaluation of the complete configuration space, i.e. usually we have a huge (sometimes even infinite) set of scenarios α and possible configurations β .



Figure 7: MAPE: Simulation of Digital Twins at Run-Time

In this case we have to operate with a *double sampling* method, i.e. we generate a sample set of scenarios and a sample of configurations; for each scenario α we evaluate the configurations from the sample and combine the evaluation of all scenarios into one overall evaluation.

The planning part including this simulation step is shown in Figure 7. Whenever we adjust

our estimation for α we generate a family of similar parameters $\alpha_1, \ldots, \alpha_m$. We generate several configuration candidates β_1, \ldots, β_n . For each scenario α_i we evaluate all these $N_{\alpha_i}(\beta_j), j = 1..n$.

However, the relationship of m and n depend on the uncertainty about α : Whenever our uncertainty about α is high we have to consider a larger number m of similar parameter sets $\alpha_1, \ldots, \alpha_m$. Since we like to keep the evaluation time constant, the product $m \cdot n$ is bounded/fixed. Therefore, a high uncertainty about α results in a large number of parameter sets α_i , which restricts the number of configuration candidates β_j under evaluation to a relatively small number. Conversely, a higher certainty in parameter α allows for a broader exploration of configuration parameters β , i.e. a higher value of n.

5. Case Study

In the following we demonstrate our method for the example given in Figure 2. The presented Petri nets are freely available online.⁵ First, we have to describe the (expected) value for a workflow. Each examination comes with some costs and costs increase with the resources spent (e.g. time and money) and the number of additional preparation courses:

$costs = (resources spent) \cdot (no of prep courses)$

We assume that the spent resources increases with the number of failed examinations – to model the increased stress level after a failed examination. Assume that we have m - 1 failures and m is the first successful examination, then the resources are calculated as (where c is some form parameter):

resources spent =
$$m^c$$
, $c > 0$

The *reward* is the inverse of the costs:

$$reward = (costs)^{-1} = (m^c \cdot t)^{-1}$$
(5)

Both values, the number of failures m-1 and the number of extra preparation courses taken t, are random numbers. Throughout the paper, we approximate the expected reward E[reward] by simulating the SWFN $N_{\alpha}(\beta)$.

5.1. Calculating the Expected Reward for $N_{\alpha}(\beta)$

Before starting with the planing part of the MAPE-loop (which is done in the following section) we evaluate the reward obtained for a variation of α and β .

We have the following hypothesis about the model's behaviour: As we assume that the costs increase with the numbers of failed examinations, it seems reasonable to skip a few examinations (i.e. to increase β) in the case of a high failure probability α to increase the chance to succeed. Therefore, we assume that there is an optimal balance between these antagonistic forces.

The workflow net itself is shown in Figure 8; it is an stochastically annotated version (using RENEW syntax) of the Petri net shown in Figure 2. Each token is a tuple [x, t, m, b], where x is the course id, t is the number of extra preparation courses, m is the number of examination tries, and b is a random number used to resolve the choices.

⁵https://github.com/koehler-bussmeier/PNaRT



Figure 8: The Stochastic Workflow Net

reward $\alpha \setminus \beta$	0.1	0.3	0.5	0.7	0.9
0.1	9	205	175	405	279
0.3	12	77	215	402	279
0.5	3	65	183	339	285
0.7	5	35	227	270	263
0.9	3	37	47	77	166

Table 1

The average *reward* value for different α , β variants (for c = 3.0, li = 3.0)

We use RENEW [14] to execute the simulation runs. For completeness the simulating net is shown in Figure 10. It generates for all combinations of values of α and β in $\{0.1, 0.3, 0.5, 0.7, 0.9\}$ an simulation batch of rounds = 50 independent simulation runs, all configured with the same parameter values c = 3.0 and li = 3.0.

For each pair of α and β we calculate the average rewards over the runs. The simulation results are given in Table 1. Here, one can observe, that with increasing failure probability α the maximal reward is obtained for increasing training probability β . Our hypothesis that students should adapt to higher failure probabilities with longer preparation times is supported by our calculation of rewards. Additionally, we observe that an increasing failure probability α reduces the reward in general, i.e. the extra learning time is used to minimise the damage, but cannot compensate the higher use of resources completely.

5.2. RENEW Specification of the MAPE-Loop

For our running toy example we can perform a complete parameter sweep, but in general this is out of reach for real models. Instead, as we explained before with Figure 7, our MAPE-loop selects only some variants of α (the explore phase) and measure how a sample of β values would perform w.r.t. a given α (the exploit phase). Finally, we pick that value of β that performs well in most of the α scenarios, i.e. is a good balance. This could be considered as a *tournament* situation.

Figure 9 shows the RENEW specification of this evaluation method as sketched in Figure 7 above. In the following, we use the value of $\gamma = 0.5$ as our score meta-parameter in (3) to compute $eval_{\gamma}(\beta)$. The Net of Figure 9 shows the system net which contains net-tokens which have the basic topology of the PWFN N from Figure 8 as net-tokens. We use the net-tokens as digital twins to evaluate the parameter settings and select the best value β^* .

We like to evaluate how well this approach operates when compared to a full parameter search as done before. Therefore, we perform this evaluation many times to reduce random effects. This is done by RENEW itself (the corresponding net is given in Figure 11). We evaluate rounds = 500 tournaments of digital twins.

Table 2 shows the evaluation of several tournaments of digital twins. Note, that the tournament is rather restricted as it considers only $m = 2 \alpha$ -scenarios for a sample of only $n = 3 \beta$ -variants. Therefore, we assume a lot of 'noise' in the selection of the tournament's winner β^* when compared to the complete parameter sweep as shown in Table 1.

Of course, we know in advance that these values for exploration and exploitation, i.e. m and



Figure 9: The Explore/Exploit Simulation of the Digital Twins for m = 2 and n = 3

n, are too small to result in reliable results. The setting used here is intended for presentation mainly and not intended to be used as a decision method. Nevertheless, the values are in accordance with the general evaluation given in Table 1.

Let us point out some observations for the tournament results given in Table 2: We see that the best value β^* for all the scenarios α is given as $\beta^* = 0.9$, i.e. agents decide to have a rather high probability to go for an extra training time before taking the examination. This is also

# wins $\alpha \setminus \beta$	0.1	0.3	0.5	0.7	0.9
0.1	41	80	112	123	144
0.3	48	67	117	131	137
0.5	54	60	98	134	154
0.7	66	74	93	120	147
0.9	42	62	85	128	183

Table 2

Tournament for m = 2 scenarios α and n = 3 configurations β (parameters: c = 3.0, li = 3.0). The table denotes the number of wins for β in 500 repetitions.

the best choice for β for $\alpha = 0.9$, but is different from the the best choice of $\beta^* = 0.7$, which has the highest reward value for the scenarios $\alpha \in \{0.1, \ldots, 0.7\}$ (cf. Tab. 1). Secondly, we can observe that at least the frequency of $\beta^* = 0.7$ is very close to that of $\beta^* = 0.9$ for the smaller values of $\alpha \in \{0.1, \ldots, 0.7\}$, while the difference is much bigger for $\alpha = 0.9$.

The reason for this is most likely the nature of the tournament: Since we pick only a small sample of n = 3 values for β , it is likely that the values $\beta = 0.7$ and $\beta = 0.9$ are not both present in a sample. However, they both will outperform the smaller values $\beta = 0.1, 0.3, 0.5$, which leads to a frequency of equal quality for $\beta = 0.7$ and $\beta = 0.9$.

A possible mechanism to overcome this problem will be to iterate the tournament and to increase the selection probability for winning values β to increase the probability that interesting candidates will be in the same tournament. This extension will be evaluated in future work.

6. Conclusion

In this paper we studied a process model, namely probabilistic workflow nets, that allows to model uncertainty. The model distinguished with external and internal choices to express that the system has the ability to adapt to changes in the external environment by changing the probabilistic distribution over internal choices. This adaption mechanism is embedded into our HORNET-model of the MAPE-loop. As the HORNET-model is formulated to support self-modification, our MAPE-loop does not only support the first-order adaption of internal choices only, but also of the complete net topology itself. To formalise this second-order adaption we use a formalism for multi-agent organisation, called SONAR. In this contribution we concentrate on first-order adaption. We have introduced a selection mechanism that defines how the system should select its best configuration β in the presence of uncertainty about the environment α . This mechanism should guarantee a good reward for a large fraction of scenarios compatible with α . Our mechanism uses a scoring approach of rankings to balance these two dimension simultaneously:

In our MAPE-loop model we use our PWFN model as a digital twin during the planning phase. We implement an approximation of our mechanism via a double sampling approach: As we usually cannot evaluate all scenarios and all configurations, we approximate the score value using a m-n-sample of α and β . Our case study of an examination process based on randomisation of failure rates and extra preparation times shows that this sampling approach

34-52

leads to results consistent with the real values even in the setting of very small sample sizes (here: m = 2 and n = 3).

We like to mention some lines of research that arise from our work:

In ongoing work we investigate the second-order adaption in more detail. We study MAS organisations within our SONAR-approach. From the perspective of adaption SONAR provides a mechanism to change the internal configuration, which leads to a change in group formation processes and team interaction processes. More specifically, a change of the team structure, e.g. as a reaction to an imbalance of the agents' work load, generates a different team and also an adapted multi-party workflow. As a consequence, the SONAR-model defines the state space of all second-order adaptations within the MAS organisation (cf. [15] for details).

We are also working on a more elaborated evaluation of the MAPE-loop processes based on our previous work on the analysis of adaption processes [16]. We will consider key-values for an abstraction of the adaption state space. To do so we consider an abstraction of the HORNETS reachability graph, where only types of the net-tokens are considered. The set of types of the net-tokens that occur in the current marking are considered as the system's gene-pool. In our MAPE-loop setting the gene-pool is the set of scenarios, i.e. $\{N_{\alpha}(\beta) \mid \alpha \in C(\alpha_0)\}$. The analysis will allow to evaluate the long-run quality of the adaption process as provided by the MAPE-loop.

In future work we like to extend the MAPE-loop in the following way: There is a close connection of *models@run-time* and Process Mining [17]. Here, we consider the model variants (i.e. the models@run.time) as parts of the planning process, which can be used to adapt the system's configuration. We would like to deduce the process nets and the organisational model using process mining techniques, i.e. we observe the real system and generate the model@run.time from the event logs during the analysis phase of the MAPE-loop.

References

- [1] D. Weyns, Software engineering of self-adaptive systems: An organised tour and future challenges, in: ICAC/SASO, 2017.
- [2] R. Valk, Object Petri nets: Using the nets-within-nets paradigm, in: J. Desel, W. Reisig, G. Rozenberg (Eds.), Advanced Course on Petri Nets 2003, volume 3098 of *Lecture Notes in Computer Science*, Springer-Verlag, 2003, pp. 819–848.
- [3] M. Köhler-Bußmeier, Hornets: Nets within nets combined with net algebra, in: K. Wolf, G. Franceschinis (Eds.), International Conference on Application and Theory of Petri Nets (ICATPN'2009), volume 5606 of *Lecture Notes in Computer Science*, Springer-Verlag, 2009, pp. 243–262.
- [4] W. v. d. Aalst, Verification of workflow nets, in: P. Azeme, G. Balbo (Eds.), Application and theory of Petri nets, volume 1248 of *Lecture Notes in Computer Science*, Springer-Verlag, Berlin Heidelberg New York, 1997, pp. 407–426.
- [5] J. Desel, J. Esparza, Free choice Petri nets, volume 40, Cambridge university press, 1995.
- [6] M. Kwiatkowska, G. Norman, R. Segala, J. Sproston, Automatic verification of real-time systems with discrete probability distributions, Theoretical Computer Science 282 (2002) 101–150. doi:https://doi.org/10.1016/S0304-3975(01)00046-9.

- [7] E. Grädel, W. Thomas, T. Wilke, Automata, Logics, and Infinite Games: A Guide to Current Research, volume 2500 of *Lecture Notes in Computer Science*, Springer, 2002.
- [8] R. Alur, T. A. Henzinger, O. Kupferman, Alternating-time temporal logic, J. ACM 49 (2002) 672–713. doi:10.1145/585265.585270.
- [9] R. S. Sutton, A. G. Barto, Reinforcement Learning: An Introduction, The MIT Press, 2018.
- [10] F. A. Oliehoek, C. Amato, A Concise Introduction to Decentralized POMDPs, SpringerBriefs in Intelligent Systems, Springer-Verlag, 2016.
- [11] E. Best, R. Devillers, M. Koutny, Petri Net Algebra, EATCS Monographs on Theoretical Computer Science Series, Springer-Verlag, 2001.
- M. Köhler-Bußmeier, M. Wester-Ebbinghaus, Sonar*: A multi-agent infrastructure for active application architectures and inter-organisational information systems, in: L. Braubach, W. van der Hoek, P. Petta, A. Pokahr (Eds.), Conference on Multi-Agent System Technologies, MATES 2009, volume 5774 of *Lecture Notes in Artificial Intelligence*, 2009, pp. 248–257.
- [13] M. Köhler-Bußmeier, M. Wester-Ebbinghaus, D. Moldt, A formal model for organisational structures behind process-aware information systems, Transactions on Petri Nets and Other Models of Concurrency. Special Issue on Concurrency in Process-Aware Information Systems 5460 (2009) 98–114.
- [14] O. Kummer, F. Wienberg, M. Duvigneau, J. Schumacher, M. Köhler, D. Moldt, H. Rölke, R. Valk, An extensible editor and simulation engine for Petri nets: Renew, in: J. Cortadella, W. Reisig (Eds.), International Conference on Application and Theory of Petri Nets 2004, volume 3099 of *Lecture Notes in Computer Science*, Springer-Verlag, 2004, pp. 484 – 493.
- [15] M. Köhler-Bußmeier, J. Sudeikat, Balance vs. contingency: Adaption measures for organizational multi-agent systems, in: K. Jander, L. Braubach, C. Badica (Eds.), 15th International Symposium on Intelligent Distributed Computing (IDC'22), volume 1089 of *Studies in Computational Intelligence*, Springer-Verlag, 2023, pp. 224–233.
- [16] M. Köhler-Bußmeier, H. Rölke, Analysing adaption processes of Hornets, in: M. Köhler-Bußmeier, D. Moldt, H. Rölke (Eds.), Petri Nets and Software Engineering 2022, PNSE'22, volume 3170, CEUR, 2022, pp. 80–98.
- [17] W. M. P. van der Aalst, Process Mining: Data Science in Action, 2 ed., Springer, Heidelberg, 2016. doi:10.1007/978-3-662-49851-4.

A. Experimental Set-Up: Nets



Figure 10: Renew Net: Reward Evaluation of α, β Variants

51



Figure 11: RENEW Net: The Experimental Set-Up of the Tournament