Learning environment model at runtime for self-adaptive systems

Moeka Tanabe
Waseda University
Shinjuku-ku
Tokyo, Japan
m-tanabe@fukazawa.info.waseda.ac.jp

Kenji Tei
National Institute of Informatics
Chiyoda-ku
Tokyo, Japan
tei@nii.ac.jp

Yoshiaki Fukazawa
Waseda University
Shinjuku-ku
Tokyo, Japan
fukazawa@waseda.jp

Shinichi Honiden
National Institute of Informatics
Chiyoda-ku
Tokyo, Japan
honiden@nii.ac.jp

ABSTRACT

Self-adaptive systems alter their behavior in response to environmental changes to continually satisfy their requirements. Self-adaptive systems employ an environment model, which should be updated during runtime to maintain consistency with the real environment. Although some techniques have been proposed to learn environment model based on execution traces at the design time, these techniques are time consuming and consequently inappropriate for runtime learning. Herein, a technique using a stochastic gradient descent and the difference in the data acquired during the runtime is proposed as an efficient learning environment model. The computational time and accuracy of our technique are verified through a case study.

CCS Concepts

•**Theory of computation** → *Online learning algorithms;* •**Software and its engineering** → *Model-driven software engineering; Designing software; Software implementation planning;*

Keywords

Self-adaptive, Learning, Gradient descent

1. INTRODUCTION

The demand for self-adaptive systems[12, 2], which can alter their behavior in response to environmental changes and continually satisfy the requirements, has increased. Recent studies[5, 4] have realized self-adaption satisfying the requirements with formal guarantees. A self-adaptive system verifies whether the current behavior model, which discretely represents the execution environment, satisfies the requirements specified as the safety and liveness properties. If the model does not meet the properties, then the system synthesizes a new behavior model that does.

The analysis and synthesis depend on the environment model. When the environment model becomes inconsistent with the runtime environment, it cannot be assured to satisfy requirements. Because the environment is typically uncertain during development, constructing an ideal environment model at development time is difficult. Additionally, valid assumptions made in an environment model during development may become invalid during the runtime because the environment changes. To reduce violation risks, developers may construct an environment model with weak assumptions. Although weaker assumptions cause satisfiable requirements to also be pessimistic, they can be guaranteed. On the other hand, when developers construct an environment model with strong assumptions to guarantee rich requirements, the violation risk increases. Therefore, the environment model should be updated during the runtime to maintain consistency with the real environment.

Existing studies have proposed techniques to learn environment model based on the execution traces of a system. These techniques assume that an environment model is learned offline using the execution traces collected in the training phase. However, the training phase environment may differ from the runtime environment. Additionally, learning in these techniques is time consuming. Hence, existing techniques are inappropriate for runtime learning, but a self-adaptive system must have an environment model with online learning to reflect the environmental changes.

In this paper, we propose an efficient online learning technique to update the environment model using the difference in the execution traces collected during the runtime. We use a stochastic gradient descent for learning an environment model incrementally, reducing the computational time for

SAC 2017,April 03-07, 2017, Marrakech, Morocco

http://dx.doi.org/10.1145/3019612.3019776

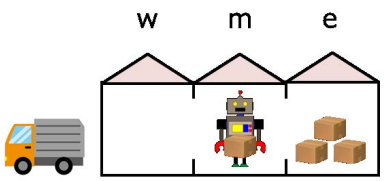

Figure 1: Automated warehouse management system

learning. Then a case study, which employs an automated warehouse management system and compares the results to existing techniques, is implemented to evaluate the computational time and accuracy of our technique.

The rest of this paper is organized as follows. A motivating example and the environment model are described in Section Ⅱ. Section Ⅲ introduces related works about the learning environment model and its challenges. Technical details of differential learning are discussed in Section Ⅳ. Then Section Ⅴ evaluates the method proposed in Section Ⅳ and compares it to an existing technique. Finally, the conclusion and future works are described in Section Ⅵ.

2. BACKGROUND

2.1 Motivating example

We explain our research using an automated warehouse management system as an example. In this system, a robot transfers products from the storage area to the shipping area in a warehouse. Figure 1 depicts the warehouse configuration. Area w is the shipping area for shipping preparations, area m connects areas w and e, and the area e is where products are stored. The robot can execute three actions: (i) `move.{e, w}`: move toward the area e, w, (ii) `pickup`: pick up a product, (iii) `putdown`: put down a product. This system has some functional requirements that must be satisfied during execution. For example, "a robot must transfer a product from area e to w", "a robot must not execute `pickup` while holding a product", and "a robot must move toward area w while holding a product".

2.2 Environment model

We use the Labelled Transition System (LTS)[4] to describe the behavior of the environment. LTS is defined as a tuple $E = (S, A, \Delta, s_0)$. S is a finite set of states. A is a set of actions, while $\Delta \subseteq (S \times A \times S)$ is a transition relation. s_0 is the initial state of E. Transitions, which are labeled with names of actions $a \in A$, are controllable and monitorable. An environment model alternates between executing a controllable action (signal from the system to the environment) and receiving a monitorable action (signal from the environment to the system). Figure 2 shows an example of an environment model.

Controllable actions are `move.{e,w}`, `pickup`, and `putdown`, while monitorable actions are `arrive.{e,m,w}`, `pickupsuccess`, and `putsuccess`. Figure 2 shows that the environment model alternates between a controllable action and a monitorable action. For example, when a robot is at area w (i.e., `arrive.w` is monitored), and then the system

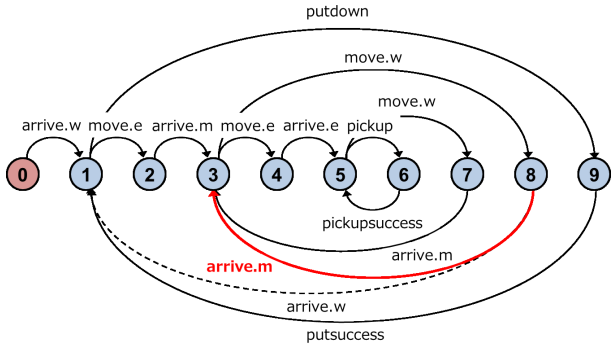

Figure 2: Environment model of example system

may execute `move.e`. It is expected that the robot will arrive at area m (i.e., `arrive.m` is monitored).

2.3 Environmental changes

The behavior of a system is determined to satisfy the given requirements in an environment model assumed at the design time. However, the requirements will be violated if the environment model becomes inconsistent with the actual runtime environment. In this case, the system should learn a new environment model and change its behavior to satisfy the requirements under the new environment model. Because self-adaptive systems can cope with environmental changes quickly by executing these processes during the runtime, such a system must accurately recognize and model the environmental changes.

The motivating example is the case where a physical change occurs (e.g., areas w and m become disconnected by obstacles). This scenario is depicted in Figure 2, where the red arrow is a new transition and the dotted arrow is an obsolete one. If the system cannot recognize this environmental change, the system following the old specifications will execute controllable action `move.w` repeatedly. Then the requirements will not be satisfied.

The system can re-synthesize a specification that satisfies the requirements in the new environment. The purpose of this study is to create an accurate runtime environment model in order to realize self-adaptive system, guaranteeing the requirements are satisfied.

3. RELATED WORKS AND PROBLEMS

3.1 Existing works on the learning environment

Previous works have proposed learning environment models. The environment is modeled as a Markov decision process (MDP) in [7]. All states and transitions are prepared at the design time. Each transition has a reward which is the degree of satisfaction of the non-functional requirements. Although this existing research deals with maximizing the reward, our research deals with functional requirements that must be satisfied.

In [11, 8], the environment is learned through reinforcement

learning by agents. In reinforcement learning, the agent explores and exploits knowledge of the environment. Our research, which deals with guarantees of requirements satisfaction, does not perform reinforcement learning since some requirements may become unsatisfied when a system explores the knowledge.

In [6, 3], the environment is modeled as a Petri net, which can reproduce a given execution log. The model is learned offline using a process mining. The environment is modeled as a logic program in [13, 10]. NoMPRoL is proposed to estimate the plausibility of each rule of the logic program. Learning is based on the execution traces obtained by the test runs of the system and performed offline by gradient descent. In contrast to these works, learning is performed during the runtime in our research.

3.2 GD-based Learning

As mentioned above, it is possible to learn a plausible environment model representing the interaction of the environment and system using NoMPRoL[13]. Here, we describe how to apply the gradient descent (GD) used in NoMPRoL to our study and learn the environment model.

First, three inputs of learning are explained: (i) some action sets, (ii) rules R, and (iii) threshold ζ. A system records the controllable and monitorable actions in an execution trace. Action set is a set of three actions extracted from the execution trace. These are monitorable, controllable, and monitorable, which are defined as the pre-condition, action, and post-condition, respectively. R, which is a set of the pre-condition, action, and post-conditions that may be observed after the action, is expressed as

$$< \textit{pre-condition}, \textit{action}, \textit{post-conditions}\{\alpha, \beta, \gamma, ...\} >$$

ζ is the value used to decide whether to add to the environment model. R and ζ are specified at the design time.

GD is an optimization algorithm to find the parameter that minimizes the function by evaluating its gradient. Equation 1 defines the error function, which shows the error of the execution trace and the rules. The post-condition for each rule $r \in R$ has an observation probability. The observation probabilities are estimated by learning, and the environment model is constructed with the rules which include the post-conditions exceed a certain value of the estimated observation probability. Learning is performed by adjusting the parameters of the ratio for the estimated observation probability and minimizing the error function based on Equation 1 and 2.

$$MSE(\mathbf{p}) = \frac{1}{X_c} \sum_{j=1}^{X_c} (1 - P(x_j|B_c))^2 \tag{1}$$

$$P(x_j|B_c) = \frac{\sum_{\{b \in B_c, b|=x_j\}} \theta_b}{\sum_{\{b \in B_c\}} \theta_b} \tag{2}$$

c is the set of a pre-condition and an action. B_c is a set of post-conditions in the rule r_c which contains c, and X_c is the number of times c is monitored. x_j is a monitored post-condition in j-th, and θ_b is the ratio of the estimated observation probability of the post-condition $b \in B_c$. Each parameter is repeatedly updated by Equation 3 until the value of Equation 1 converges.

$$\mathbf{p}_{t+1} = \mathbf{p}_t - \eta \nabla MSE(\mathbf{p}_t) \tag{3}$$

$\mathbf{p}_t$ is a vector with a value of $\theta_{\{b \in B_c\}}$ in time t, and η is the learning rate.

3.3 Problem in existing methods

To reflect the runtime changes in the environment model, runtime learning must be performed. However, in the existing method using GD, all of the action sets included in the given execution trace is used, and the computation is repeated until the value converges. The number of computations of Equation 2 increases due to the increase in the number of post-conditions, rules, or trace lengths, enlarging the learning time. In the example of the automated warehouse management system, the number of rules increases when there are more areas. Currently, it can take a long time (10 seconds) to learn. Consequently, this method of learning each time the action set is obtained during the runtime is unrealistic.

4. DIFFERENTIAL LEARNING

We propose a method for differential learning in order to establish an efficient learning environment model from the data obtained during the runtime. Reducing the amount of data and the learning time by differential learning using a stochastic gradient descent (SGD)[1] realizes effective runtime learning.

4.1 Difference from existing method

Figure 3 shows how GD and SGD deal with the data obtained during the runtime. In the GD method, learning is performed based on the action sets obtained for a fixed period in the past. The parameters are updated repeatedly until the error function converges. In contrast, in the SGD method, learning is performed based on new action set obtained at the learning time, and the parameters are updated only once. The rules are updated incrementally.

4.2 Overview

Algorithm 4.1 is the algorithm for differential learning. The inputs of learning are (i) obtained action set at the learning time $< pre_o, a_o, b_o >$, (ii) rules R, and (iii) threshold ζ. When new action set is obtained, differential learning is performed. First, the rule to be learned that has the same pre-condition and action as the obtained action set is extracted in line 2. For the extracted rule, the observation probability of each post-condition b is estimated by SGD in line 4. Based on the result of the computation and input ζ, the post-conditions with the high estimated observation probability are adopted in the environment model from lines 6 to 12. $sum(r.B)$ is the total value of $\theta_{b \in B}$. When $b.rule$ is true, b is adopted, and when $b.rule$ is false, b is rejected.

4.3 SGD-based Learning

SGD is a kind of gradient method. When one data is read, the parameters are updated by computing gradients using only the read data. In a normal SGD, all of the given data

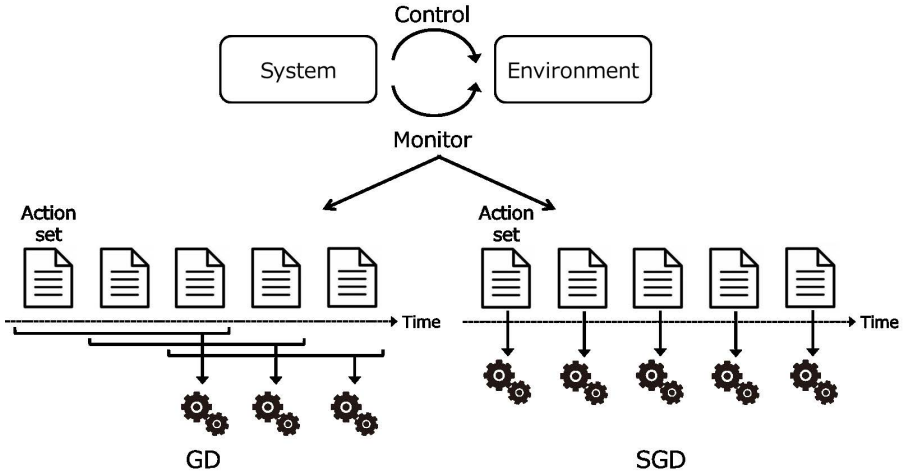

Figure 3: GD and SGD

Algorithm 4.1 Differential learning

Input: R, ζ, $< pre_o, a_o, b_o >$(Obtained action set)
Output: updated R

```
for all r ∈ R do
  if r.pre == pre_o and r.a == a_o then
    for all b ∈ r.B do
      θ_b = θ_b − η ∂MSE/∂θ_b
    end for
    for all b ∈ r.B do
      if θ_b/sum(r.B) ≤ ζ then
        b.rule ← false
      else
        b.rule ← true
      end if
    end for
  end if
end for
return R
```

are arranged randomly. Then one data is selected and used for the computation of the gradient. The computation data is selected in order along the time series in this study. In addition, another method where the computation is repeated with the data arranged randomly after finishing the computation. In this study, this secondary method is not implemented.

The error function is defined as Equation 4. The parameter **p** is estimated by minimizing this function, which is equivalent to Equation 1 in the previous chapter.

$$MSE(\mathbf{p}) = (1 - P(x_j|B_c))^2 \tag{4}$$

$P(x_j|B_c)$ is computed based on Equation 2, and the updated **p** is computed based on Equation 3 in the previous chapter. The estimated observation probability of each post-condition is computed based on the estimated **p**. The computed probabilities are compared to the given ζ, and whether to adopt to the environment model is determined.

We show a specific learning example with the automated warehouse management system. When the following action set is obtained

$$< arrive.m, move.w, arrive.m >$$

the following rule is extracted for learning

$$< arrive.m, move.w, \{arrive.m, arrive.e, arrive.w\} >$$

This rule has the same set of pre-condition and action as the obtained action set. The post-conditions of $b1, b2, b3$ have the parameters of θ_{b1}, θ_{b2}, θ_{b3}. The parameters are updated based on the computed gradient by Equation 3. When the computed values are $\theta_{b1} = 1.4$, $\theta_{b2} = 0.2$, $\theta_{b3} = 0.4$, each estimated observation probability becomes

$$b1 : \frac{\theta_{b1}}{\theta_{b1} + \theta_{b2} + \theta_{b3}} = 0.7$$

$$b2 : \frac{\theta_{b2}}{\theta_{b1} + \theta_{b2} + \theta_{b3}} = 0.1$$

$$b3 : \frac{\theta_{b3}}{\theta_{b1} + \theta_{b2} + \theta_{b3}} = 0.2$$

The computed probabilities are compared to the given ζ. When ζ is 0.15, $b1$ and $b3$ are adopted into the environment model since $\theta_{b1}(0.7)$ and $\theta_{b3}(0.2)$ are greater than ζ. $b2$ is rejected because $\theta_{b2}(0.1)$ is less than ζ. Therefore, the learned rule is as follows

$$< arrive.m, move.w, \{arrive.m, arrive.w\} >$$

When this rule differs from the rule included in the environment model, the environment model is updated based on the obtained rule.

The updated environment model is composed of deterministic actions and one or more than one non-deterministic actions. In this study, it is possible to generate a specification that considers some non-deterministic post-conditions when the estimated observation probability has a value that exceeds specified threshold.

Table 1: Scenario settings

Settings	Small	Large
Number of areas	3	150
Number of rules	10	1171
Number of post-conditions	26	5229

5. EVALUATION

In this section, we compare the SGD and the GD methods in order to evaluate the learning accuracy and required time. We apply these two methods to a case study based on the example of the automated warehouse management system. Two systems with different scales are prepared for this evaluation. One is a small case (Figure 1), and the other is a large case, which has 10×15 areas. The experiments are carried to investigate the following three research questions:

RQ1: If an environment *changes*, does the method learn a model with a high degree of accuracy?

RQ2: If an environment *does not change*, does the method learn a model with a high degree of accuracy?

RQ3: How much time does each method require for learning?

5.1 Settings

For RQ1 and RQ2, the learning accuracy is evaluated using an evaluation index, which is the error defined as the difference between the actual observation probability in given execution trace and the estimated probability obtained using GD or SGD. A smaller error indicates a higher learning accuracy. For RQ3, the learning time is evaluated. The learning time is defined as the time required to update the end of the parameter from reading the execution trace.

The size of the execution trace to the input in the GD method is 3001 actions for the small case and 300001 actions for the large case. The size is determined from the result of the preliminary experiments, which evaluate the learning accuracy using the GD method with different trace sizes. The learning accuracy converges in the preliminary experiments.

The learning rate η is 0.1. The initial value of parameters $\mathbf{p}$ are 0.5, and the threshold ζ is 0.1. Table 1 shows the scenario settings.

5.2 Results

5.2.1 RQ1: Learning accuracy with environmental changes

First, in the small case, the experiment is performed on the assumption that the change occurs as mentioned in Section 2.3. A similar change is applied to the large case as it becomes impossible to move to a certain location. Figure 4 shows the results of an extracted rule related to the environmental change for the large case. Figure on experimental results for the small case is omitted due to space limitations. For both experiments, the data is acquired for ten controllable actions. The horizontal axis indicates the number of actions performed. An environmental change occurrs when 5001 and 500001 actions are performed in the small case and the large case, respectively. According to the results, the SGD method learns faster than the GD method in the large case. The small case yields similar results; the environmental change is learned and reflected in the model when 5081 actions (SGD) and 6621 actions (GD) are performed. In the large case, the change is reflected when 500100 (SGD) and 530620 (GD) actions are performed. The GD method considers the past action sets of the environment at the learning time. On the other hand, the SGD method only takes into account the newly observed action set. The weight against this newly observed post-condition increases in the SGD method.

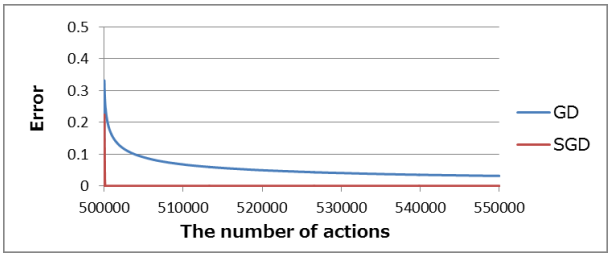

Figure 4: Learning accuracy with an environmental change (large case)

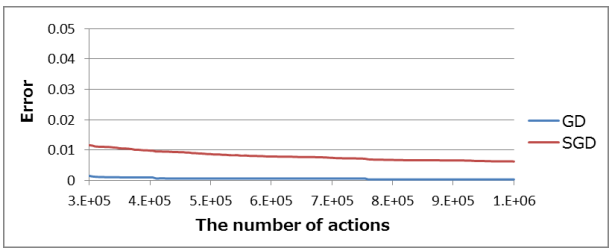

Figure 5: Learning accuracy (large case)

In the example of the small case, we also conducted the experiment on the assumption that changes occur. Because the GD method affects the action sets observed in the past environments, the error value increases. On the other hand, SGD method is not affected by past data, so it can learn quickly with a higher accuracy.

5.2.2 RQ2: Learning accuracy without environmental changes

In this section, the experiment is performed on the assumption that an environmental change does not occur. We evaluated the learning accuracy for the small and the large cases. Figure 5 shows the results for the large case. Due to space limitation, figure on experimental results for the small case is omitted. The data is taken by 10 controllable actions in the small case and by 2500 controllable actions in the large case.

In both cases, the learning accuracy of the GD method is higher than the SGD method. However, the difference in error between the GD method and the SGD method ranges

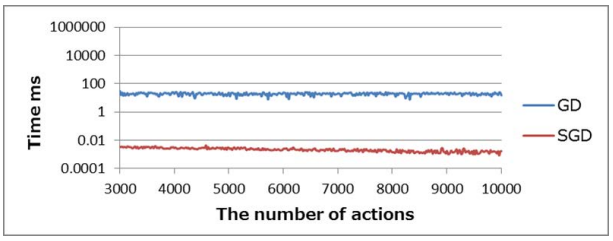

Figure 6: Learning time (small case)

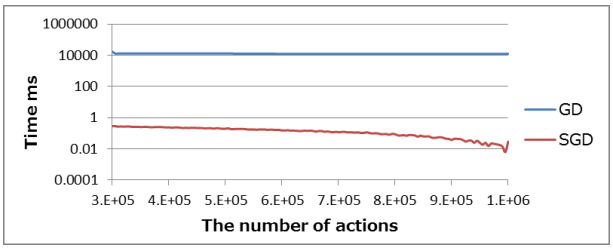

Figure 7: Learning time (large case)

from 0.01 to 0.05 in the small case and is about 0.01 in the large case. Both methods learn the environmental model with almost the same degree of accuracy.

5.2.3 RQ3: Learning time

Figures 6 and 7 show the results of the learning time for the small case and the large case, respectively. The experiment is performed on the assumption that an environmental change does not occur. The data is taken by 10 controllable actions in the small case and by 2500 controllable actions in the large case. The time is the average value obtained by performing 10 experiments.

From the result of the GD method, the large case takes 1000 times longer compared to the small case. On the other hand, the learning time for the large case is only 100 times longer than the small case for SGD. The average increase in the learning time is about 12794 ms in the GD method and about 0.14 ms in the SGD method. Hence, learning in the GD method is much more time consuming than that in the SGD method.

If learning is performed each time to obtain new action set, the learning time in the large scenario is around 10 s, which is unrealistic. In the example of the automated warehouse management system, it is assumed that the learning time reduces the work efficiency of the robot. When using SGD, it is possible to finish learning within 1 ms in the same large case, which should have a negligible downtime and impact on the work efficiency.

The computation time necessary to update the parameters once is represented as $O(n)$ in GD and $O(1)$ in SGD. The frequency of updating the parameters increases due to the increase in the number of rules. Therefore, when the scale of the system is large, the GD method significantly increases the learning time compared to the SGD method.

5.3 Threats to validity

The results of the previous section depend on the learning rate used for the computing gradient or the length of the execution trace for GD. Hence, these factors may significantly influence the results.

6. CONCLUSION

The purpose of this study is to learn an accurate environment model, which expresses the interaction between the system and the environment monitored in a real runtime environment. We propose a differential learning method for efficient learning of the environment model from the runtime data. Runtime learning is possible using SGD to reduce the amount of inputted data and learning time. We obtained the following results by comparing the existing GD method to the proposed SGD method.

- When the environment changes, the SGD method enables learning of an accurate environmental model quickly.
- When the environment does not change, the accuracy is lower than the existing GD method, but it is possible to learn with nearly the same accuracy via the GD method and the SGD method.
- The SGD method can greatly reduce the learning time.

Future works are necessary. First, this proposed method must be evaluated by applying it to an existing system. Second, an evaluation utilizing other learning rates must be conducted because many methods to calculate the learning rate exist (Adam[9], RMSProp[14], etc.). Finally, the robustness to learn the environment model regardless of the specified action needs to be further investigated.

7. ACKNOWLEDGMENT

The research was partially supported by National Institute of Information and Communications Technology (NICT), JAPAN.

8. REFERENCES

[1] L. Bottou. Online learning and stochastic approximations. *On-line learning in neural networks*, 17(9):9–42, 1998.

[2] R. De Lemos, H. Giese, H. A. Müller, M. Shaw, J. Andersson, M. Litoiu, B. Schmerl, G. Tamura, N. M. Villegas, T. Vogel, et al. Software engineering for self-adaptive systems: A second research roadmap. In *Software Engineering for Self-Adaptive Systems II*, pages 1–32. Springer, 2013.

[3] Z. Ding, Y. Zhou, and M. Zhou. Modeling self-adaptive software systems with learning petri nets. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 46(4):483–498, 2016.

[4] N. D'Ippolito, V. Braberman, J. Kramer, J. Magee, D. Sykes, and S. Uchitel. Hope for the best, prepare for the worst: multi-tier control for adaptive systems. In *Proceedings of the 36th International Conference on Software Engineering*, pages 688–699. ACM, 2014.

[5] N. R. D'Ippolito, V. Braberman, N. Piterman, and S. Uchitel. Synthesis of live behaviour models. In *Proceedings of the eighteenth ACM SIGSOFT international symposium on Foundations of software engineering*, pages 77–86. ACM, 2010.

[6] D. Fahland and W. M. van der Aalst. Repairing process models to reflect reality. In *International Conference on Business Process Management*, pages 229–245. Springer, 2012.

[7] C. Ghezzi, L. S. Pinto, P. Spoletini, and G. Tamburrelli. Managing non-functional uncertainty via model-driven adaptivity. In *Proceedings of the 2013 International Conference on Software Engineering*, pages 33–42. IEEE Press, 2013.

[8] J. E. Godoy, I. Karamouzas, S. J. Guy, and M. Gini. Adaptive learning for multi-agent navigation. In *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems*, pages 1577–1585. International Foundation for Autonomous Agents and Multiagent Systems, 2015.

[9] D. Kingma and J. Ba. Adam: A method for stochastic optimization. In *Proceedings of the 3rd International Conference for Learning Representations, arXiv:1412.6980*, 2015.

[10] D. Martínez Martínez, T. Ribeiro, K. Inoue, G. Alenyà Ribas, and C. Torras. Learning probabilistic action models from interpretation transitions. In *Proceedings of the Technical Communications of the 31st International Conference on Logic Programming (ICLP 2015)*, pages 1–14, 2015.

[11] J. Menashe and P. Stone. Monte carlo hierarchical model learning. In *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems*, pages 771–779. International Foundation for Autonomous Agents and Multiagent Systems, 2015.

[12] M. Salehie and L. Tahvildari. Self-adaptive software: Landscape and research challenges. *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, 4(2):14, 2009.

[13] D. Sykes, D. Corapi, J. Magee, J. Kramer, A. Russo, and K. Inoue. Learning revised models for planning in adaptive systems. In *Proceedings of the 2013 International Conference on Software Engineering*, pages 63–71. IEEE Press, 2013.

[14] T. Tieleman and G. Hinton. Lecture 6.5-rmsprop. *COURSERA: Neural networks for machine learning*, 2012.