1st TAB Report
Formal-methods support for runtime adaptation in role-based systems

Max Korn

Faculty of Computer Science
Technische Universität Dresden
Dresden, Germany
max.korn@tu-dresden.de

1 Introduction

With rising dependence on technology, it is growing ever more important to design systems for long-time deployment. This makes systems able to perform acceptably in a number of different situations critical. This is why there is an ongoing effort to design *self-adaptive systems*, systems that are able to adapt to diverse situations without human input.

The challenges in designing such systems are plenty [9]. In much of literature, the fundamental structure of these systems is defined by the *MAPE-K* loop [17], which requires the steps *Monitor*, *Analyse*, *Plan* and *Execute*, together with a *Knowledge* component, to create a self-adaptation structure.

In my thesis, I research the advantage formal methods, especially methods based on model checking, can have on the varying aspects of self-adaptation and the MAPE-K loop. This ranges from verification of such systems at design time, to utilizing the advantage of formal methods in runtime.

The focus of this report is split in two. After a brief summary of the current research into the field in Section 2, I will introduce an application of formal methods in decision making of adaptive systems in Section 3. After that, Section 4 will introduce a way to reason about the effectiveness of a self-adaptive model at design time. Lastly, Section 5 will conclude my report.

2 Formal Methods in self-adaptive Decision Making

There already has been research using formal methods to design self-adaptive systems[22, 14]. A significant part of that research consists of the verification of system properties at design time[4, 8, 2, 13].

Another way formal methods can improve self-adaptation is in the *Plan* phase of the MAPE-K loop. There are a number of different approaches to employ formal methods in decision making. The base idea is often the same. Being able to optimise our decision making requires an understanding of the consequences different adaptation strategies can have on system performance. Formal methods can be utilized to compute data about these consequences using a formal representation of said system.

This enjoys a number of advantages. Since most of these formal representations can be created from conceptual models used at design, it is easier to interpret the decision making progress for a human. Some[6] approaches capitalise on this, to improve the understanding one has of decision making. After establishing the method, it also takes less effort to implement into a system. All that is needed is a formal representation of the system, as well as the specification requirements. There is no need to develop a custom made heuristic for each system, nor is there a need to train some neural network.

The key challenge for these approaches to decision making lie in the complexity of the representation. Getting accurate results with formal methods requires detailed representations, with sizes often growing exponentially to the system size. This makes the computation of consequences costly, especially if you design a more complex system.

Since self-adaptive systems need to perform decision making often under time pressure, for example in a self-driving car, the formal method approaches need to decrease their decision making time as much as possible. Efforts to this effect often fall into one of two categories.

The first category are *online* approaches. These methods employ techniques to reduce computational effort, like model approximation[1, 7], statistical methods[16, 3], or other methods[15, 5, 12, 21], to reduce the decision making time into an acceptable margin. This leads to flexible decision making , often at the cost of accuracy of the resulting data, and with this the capabilities of the decision making.

The other category contains *offline* approaches[11, 20]. They split the computation of consequences, or at least parts of the computation, from the decision making process. For a set number of assumed contexts, knowledge is computed beforehand and stored, with the decider only accessing the stored knowledge for decision making. While this reduces the decision making time considerably, while not reducing the accuracy of the resulting data, it comes at the cost of other uncertainties. The most considerable problem with these approaches is their static nature. Since they depend on certain assumptions at pre-computation, they get worse at unexpected circumstances, leading in drops to their accuracy and quality of self-adaptation.

3 Offline approach with Probabilistic Model Checking

We analyse self-adaptation with formal methods using an offline approach, which computes probabilities of certain events and expectations of performance critical metrics using *probabilistic model checking*(PMC) on *Markov Decision Processes*(MDP), which are transition systems using both non-deterministic as well as probabilistic transitions.

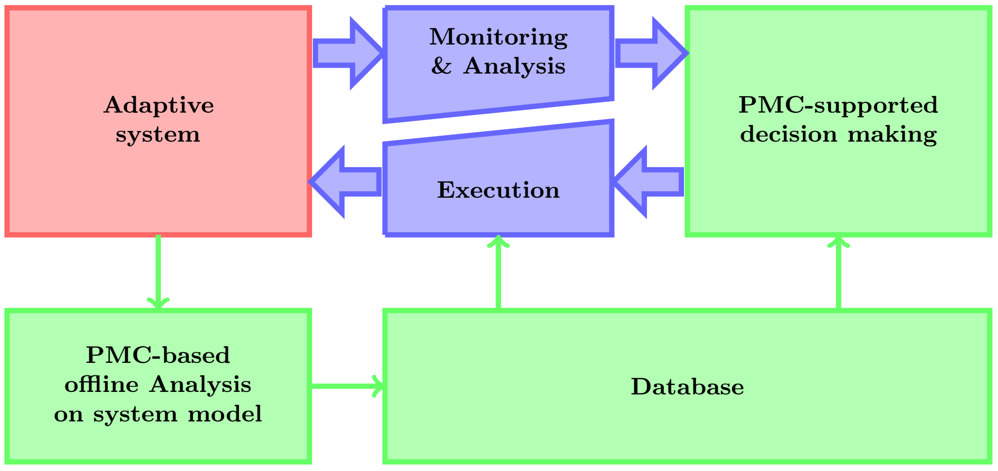

Adaptive system
Monitoring & Analysis
Execution
PMC-supported decision making
PMC-based offline Analysis on system model
Database

Fig. 1: Self-Adaption Concept

The concept, as shown in figure 1, uses an expensive PMC-based Analysis of the model of the adaptive system to fill a database with data about the ramifications of certain adaptation steps. This data can then be used to influence our decision making. This functions as follows.

For this to work, we require two things. A MDP representing our system, and data points to compute on said model. First, let us take a look at the system representation.

Our system representation is split into four parts. The core, the context, the adaptation, and the skeleton. In the core part, we model the system under consideration without any self-adaption. All parts that would be controlled by the decider, or are dependent on the context are left as non-deterministic decisions. These decisions are then controlled by the other parts.

The context consists of multiple environments, models defining behaviour of the outside factors. Exactly one environment can be active at any point in time. This means only this environment can influence the behaviour of the system.

The adaptation consists of multiple adaptation plans, all specifying a certain way to control the system. Like environments, exactly one plan can be active at any time, defining the way the system self-adaptation acts.

The skeleton implements some additional system parts allowing observation of the system.

To define what data to collect for our decision scheme, we need to first make sure what the objective of the decider should be. This means formulating a *global objective*, which the decider should try to optimise. We define this objective, by splitting it into a number of *local objectives*, which set short term metrics we can compute using PMC. These local objectives then need to be specified using a Temporal Logic. We use *Probabilistic Computational Tree Logic*(PCTL), to specify probabilities and expectations over certain events as our local objectives. It is important to note that we always need a definitive endpoint for the computation of expectations, which is reachable with a probability of 1. In all other cases, said expectation is defined as infinite.

To compute our local metrics and requirements, we use the model checker Prism[18], as well as the feature-oriented modelling language ProFeat[10], to create an feature based representation of our system. This allows us to gather results over every possible feature combination, which in our model are defined by every combination of plan, environment as well as system internal variable values observable by the decider. This way, we obtain results for state in the model, we could potentially need a decision from. We subsequently fill the database of our decider by creating one row for every combination, and filling the row with the result of all local objectives, as seen in Table 1.

observable state	context	adaptation	property 1(Pr)	property 2(Exp)	...
state 1	environment 1	plan 1	0.98	37.8	...
state 1	environment 1	plan 2	0.68	59.2	...
state 1	environment 1	plan 3	0.43	12.7	...
state 1	environment 2	plan 1	0.42	35.4	...
state 1	environment 2	plan 2	0.99	37.9	...
state 1	environment 2	plan 3	0.79	27.4	...
state 2	environment 1	plan 1	0.01	120.6	...
⋮	⋮	⋮	⋮	⋮	⋮

Table 1: Example of the resulting database

The decider now reasons only about this data. For an decision, we use the global objective to create a sorting order of our local objectives, which we call a *strategy.* After removing all rows not matching the current situation, we use this strategy to identify the optimal row for our global objective, and use the plan specified in this row as instruction for further adaptation.

Naturally, the decider will obtain no metrics once an environment appears, that is not in the environment assumptions. For this, the Analyse step needs to make sure to find the assumption that fits best to the current environment. This introduces uncertainty, which will in turn affect how well the system can hold to the global objective.

4 Simulating self-adaptive systems

Quantifying how well the decider is able to reach the global objective is a significant problem. While most industry deciders have to be tested on the running system, or in some cases at least simulated in a test environment which adequately represents the real circumstances, performance data always requires a lot of work to gather. Only after that are we able to judge the performance of the approach in question.

Using formal methods to judge system performance is a good approach, to test the system at design time. Since we already utilize a formal model of our system to generate the knowledge of our decider, it makes sense to use this same model to reason about the performance of our approach.

We used PMC based *statistical* model checking[19] to implement such an analysis tool. This model checking technique takes a set number of randomly chosen paths on the given transition model (here MDP), and then analyses probabilities as the percentage of paths fulfilling the property. Depending on the number of paths chosen, this approach can significantly reduce the time it takes to compute results, at the cost of accuracy. Using this technique, we are able to implement the decider into the analysis, without affecting the system representation. This is done by limiting the randomly chosen paths to the ones that align with the one that would have been chosen by the decider, by implementing said decider into the path choosing process.

We integrated this approach into the statistical model checking process implemented in Prism. The resulting plugin allows the decider at previously defined states in the system, to modify a current variable, therefore altering the path currently analysed to one the decider approves of.

The combination of this technique with the decider creation specified in the previous chapter leads to the following MAPE-K loop:

- **The system** is now replaced with a system model, similar to the one used in the decision process. In fact, to streamline the simulation process, we use the same model as in the previous section, with some structural changes. These changes are limited to the skeleton, s.t. environments and adaptation plans can be changed at runtime. Every change of an environment is followed by a point, where the decider can act, and change an adaption plan.
- The **Monitor and Analyse** steps are now contained in the current state of the system. Since the value of all observable variables as well as the active environment are encoded in the current state, we always posses perfect knowledge of the current context. This effect can be mitigated, so that we can see the performance without perfect prediction of the current environment, by introducing variables to the environment that do not exist in the pre-processing.
- The **Plan** step is executed in the decider. The point at which decisions can be made is directly after an environment could be exchanged.

- The **Execute** step is now realised by changing which of the adaption plans is currently active. The change is initiated immediately after the decider has selected a new plan.

The resulting framework needs only the system model, a specification of possible environments and plans, as well as a strategy and its corresponding local objective, to automatically create a decider as specified in the previous section, as well as a model we can use to test the performance of said decider using the Prism plugin. This gives us the ability to test for a various amount of systems the ability of the decider to hold to a global objective. Additionally, we can also utilize this to compare with heuristic deciders, by encoding them into Prism and assigning them as one of the possible plans.

For a first analysis of decision making, we examined the behaviour of our decider on the model of the following system.

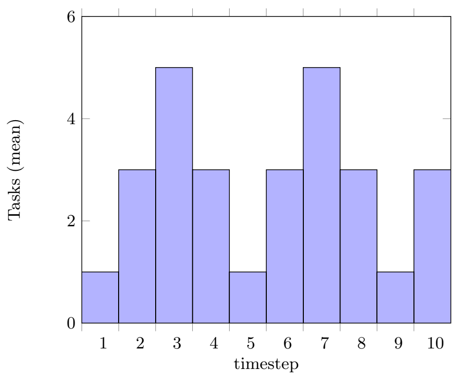
Example 1. Given is a load scheduling system of a computing complex. Its goal is to schedule the appropriate amount of hardware for every time of day, s.t. we do not waste energy in the process, but also do not let a task wait too long. The model assumes the following properties:

- every day consists of 10 *timesteps*
- at every timestep, the system gets between $1-6$ distinct *tasks*, all of which come with the same computational load requirements.
- the system is able to schedule the number of used cores, the frequency, and the usage of hyper-threading as needed.
- At every timestep, the system is able to use this scheduled hardware to compute $1-6$ tasks.
- Tasks can be taken over to the next timestep.
- There can only be 6 tasks scheduled at most. All task acquired at that point will be rejected, at the cost of an *Service-Lifetime-Agreement violation* (SLA-violation).
- *energy costs* of scheduled computational power grows exponentially (enough hardware for one task costs one, for two it costs two, for three it cost four etc.)

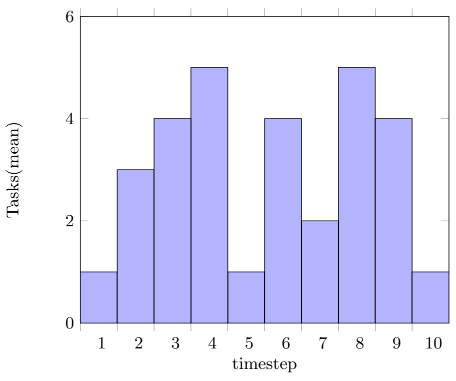
Our global objective can be summarised as 3 goals:

- Finish all received tasks by the end of the day.
- Make sure there are no rejected tasks (ie. make sure that there are no SLA-violations).
- Reduce energy cost as much as possible.

As environments we use different load distributions throughout the day, where every timestep gets a probability distribution over the number of tasks that can be given at that point in time. The plans specify how the hardware is allotted until the next time the decider is active. This means we use every possible combination of allotment over every timestep till then. Since the computational overhead would grow too massive to compute, should we specify this



(a) First Environment



(b) Second Environment

Fig. 2: Environments

for all timesteps, we guarantee that the decider will work at least every third timestep (meaning we can specify plans for up to three steps into the future).

To evaluate our approach, we have run extensive tests on a variety of parameters. These include:

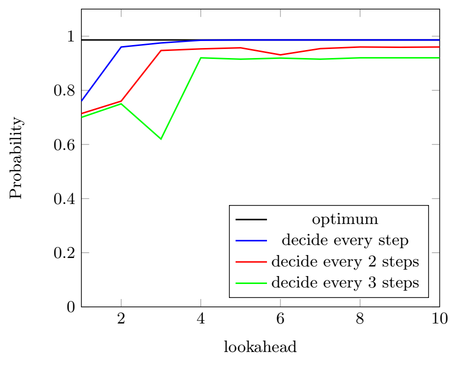
- different queries for the decider
- different probabilistic and deterministic environments
- time-spans until the decider activates again
- limited lookahead of the decider (i.e. allowing the local objective computation only knowledge of x timesteps)
- factors of noise changing environment behaviour at runtime (to simulate unknown environments)

First, let us look at the seldom case, where we are able to specify our global objective using only one local objective. For this we look at the following environment. This environment assigns tasks at every timestep using a discrete normal distribution bound by the limits of the assignable tasks, with a variance of 1 and a mean dependent on each timestep as seen in figure 2a.

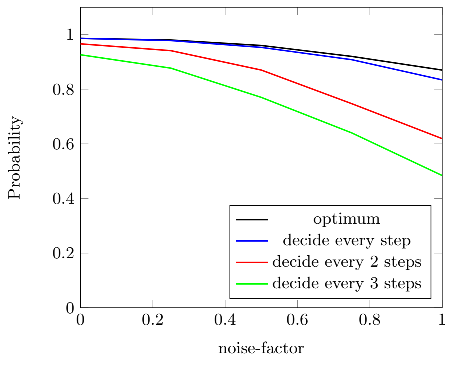
`Budget:` $Pr(\neg\texttt{sla-violation}\,\mathbf{U}((\texttt{budget} >= 0) \wedge \texttt{no-task-left} \wedge \texttt{end-of-day}))$

The formula `Budget` describes the likelihood of satisfying objective 1 and 2, while keeping the accrued costs in a previously defined budget (90 for this example). Using this local objective, we can define the global objective as simply maximising this probability.

Figure 3a shows, that a decider, trying to maximise this probability, will converge this probability to its theoretical optimum 0.986 (computed using regular model checking without decider) with a rising lookahead. This is easier done with a decider that returns often, since this gives it more flexibility in optimising its query.



(a) by Lookahead



(b) by noise-factor (Lookahead 10)

Fig. 3: Optimizing `Budget`

Since our approach of testing self-adaption comes with perfect knowledge about the current state of the system or environment, we cannot talk about the reliability of environment knowledge. But this is a critical aspect of our offline approach, since it represents one of its weaknesses. This is why we introduced a noise-factor into the model, a variable that is only sampled at runtime, which can move the mean of environment distribution by $\pm 2 \times$ noise-factor with uniform probability. This approach will also change the optimal value reachable.

Figure 3b shows the effect this noise-factor has on the `Budget`. We see that both the result of the decider, as well as the theoretical optimum fall with rising noise. But the decider falls faster, making its performance worse the more noise we introduce. This effect is strengthened with the distance between decisions.

For a deployed self-adaptive system, this would mean, that to gain the optimal result, you have to make sure to compute data for as much environment assumptions as possible, and make sure that the knowledge of the decider gets updated with new and unknown environments regularly.

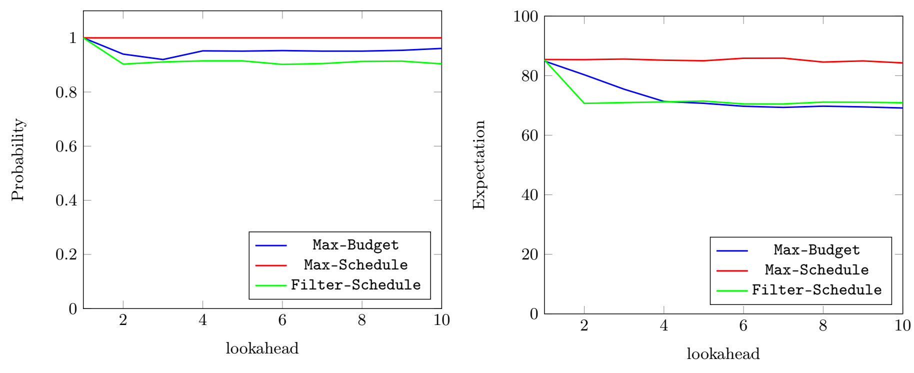
Once we need to split our global objective into multiple local objectives, there no longer exists a explicit optimum we can compare ourselves to. This means, different strategies for self-adaptation need to use varying priorities for local objectives, to lead self-adaptation. This introduces additional challenges. For this, we look at at the second environment in figure 2b, with variance 2.

The probability

$$\texttt{No-Errors:}\ Pr(\neg\texttt{sla-violation}\,\mathbf{U}(\texttt{no-task-left} \wedge \texttt{end-of-day}))$$

expresses the likelihood of reaching the end of the day without any SLA-violations and with no tasks left. Maximising this describes the first two objectives of our decider. The Expectation

$$\texttt{Costs:}\ Exp^{cost}(\Diamond\texttt{end-of-day})$$



(a) `No-Errors` (b) `Costs`

Fig. 4: Different strategies in comparison

describes the expected costs till the end of the day. Minimising this is our third objective. To optimise our global objective, we now need to keep the probability of `No-Errors` as high as possible, while keeping the expected `Costs` as low as possible.

Figure 4 shows different approaches to this optimisation:

– `Max-Budget` tries to maximise `Budget` (with the same budget of 90). Since this formula incorporates all objectives, it gets an all around good performance in local objectives.
– `Max-Schedule` tries to maximise `No-Errors`, and then minimise `Costs` (meaning we sort the database with these columns in order). This leads to an optimal result for the first two objectives, at the cost of the third objective.
– `Filter-Schedule` is more lax than the previous approach. By first limiting the table to rows with `No-Errors` above a certain probability (here 0.9), and then minimising `Costs`, we get a decider that is able to reach better costs at much lower lookahead as `all-in-one`, at the expense of the first two objectives.

As we can see, the results vary heavily depending on the priorities of the strategy. This makes it especially important to take great care in the definition of ones priorities when splitting the global objective.

5 Conclusion

In conclusion, we have shown that formal methods represent very powerful tools to steer decision making in self-adaptive systems. We have also demonstrated how they can be used to improve the design of such system, by providing tools to measure the performance of a potential decider.

Publications. We are currently working on a paper introducing this decision making approach, as well as the verification tool. Additionally, we are analysing the potential of the inclusion of other approaches, like parametric model checking, as well as using this approach with role-based automata. We are also looking into examining the split of global into local objectives, and what guarantees a structured approach to this could give.

References

[1] Mehran Alidoost Nia, Mehdi Kargahi, and Fathiyeh Faghih. "Probabilistic approximation of runtime quantitative verification in self-adaptive systems". In: *Microprocessors and Microsystems* 72 (2020), p. 102943. ISSN: 0141-9331. DOI: https://doi.org/10.1016/j.micpro.2019.102943. URL: http://www.sciencedirect.com/science/article/pii/S0141933118303843.

[2] Paolo Arcaini, Elvinia Riccobene, and Patrizia Scandurra. "Formal Design and Verification of Self-Adaptive Systems with Decentralized Control". In: *ACM Trans. Auton. Adapt. Syst.* 11.4 (Jan. 2017). ISSN: 1556-4665. DOI: 10.1145/3019598. URL: https://doi.org/10.1145/3019598.

[3] D. M. Barbosa et al. "Lotus@Runtime: A Tool for Runtime Monitoring and Verification of Self-Adaptive Systems". In: *2017 IEEE/ACM 12th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS).* 2017, pp. 24–30. DOI: 10.1109/SEAMS.2017.18.

[4] Matthias Becker, Markus Luckey, and Steffen Becker. "Performance Analysis of Self-Adaptive Systems for Requirements Validation at Design-Time". In: *Proceedings of the 9th International ACM Sigsoft Conference on Quality of Software Architectures.* QoSA '13. Vancouver, British Columbia, Canada: Association for Computing Machinery, 2013, pp. 43–52. ISBN: 9781450321266. DOI: 10.1145/2465478.2465489. URL: https://doi.org/10.1145/2465478.2465489.

[5] N. Bencomo, A. Belaggoun, and V. Issarny. "Dynamic decision networks for decision-making in self-adaptive systems: A case study". In: *2013 8th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS).* 2013, pp. 113–122. DOI: 10.1109/SEAMS.2013.6595498.

[6] N. Bencomo et al. "Self-Explanation in Adaptive Systems". In: *2012 IEEE 17th International Conference on Engineering of Complex Computer Systems.* 2012, pp. 157–166. DOI: 10.1109/ICECCS20050.2012.6299211.

[7] R. Calinescu et al. "Dynamic QoS Management and Optimization in Service-Based Systems". In: *IEEE Transactions on Software Engineering* 37.3 (2011), pp. 387–409. DOI: 10.1109/TSE.2010.92.

[8] J. Cámara and R. de Lemos. "Evaluation of resilience in self-adaptive systems using probabilistic model-checking". In: *2012 7th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS).* 2012, pp. 53–62. DOI: 10.1109/SEAMS.2012.6224391.

[9] Betty H. C. Cheng et al. “Software Engineering for Self-Adaptive Systems: A Research Roadmap”. In: *Software Engineering for Self-Adaptive Systems.* Ed. by Betty H. C. Cheng et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 1–26. ISBN: 978-3-642-02161-9. DOI: `10.1007/978-3-642-02161-9_1`. URL: `https://doi.org/10.1007/978-3-642-02161-9_1`.

[10] Philipp Chrszon et al. *ProFeat: feature-oriented engineering for family-based probabilistic model checking.* 2018. DOI: `10.1007/s00165-017-0432-4`.

[11] Antonio Filieri, Carlo Ghezzi, and Giordano Tamburrelli. “Run-Time Efficient Probabilistic Model Checking”. In: *Proceedings of the 33rd International Conference on Software Engineering.* ICSE ’11. Waikiki, Honolulu, HI, USA: Association for Computing Machinery, 2011, pp. 341–350. ISBN: 9781450304450. DOI: `10.1145/1985793.1985840`. URL: `https://doi.org/10.1145/1985793.1985840`.

[12] Vojtěch Forejt et al. “Incremental Runtime Verification of Probabilistic Systems”. In: *Runtime Verification.* Ed. by Shaz Qadeer and Serdar Tasiran. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 314–319. ISBN: 978-3-642-35632-2.

[13] M. Güdemann, F. Ortmeier, and W. Reif. “Safety and Dependability Analysis of Self-Adaptive Systems”. In: *Second International Symposium on Leveraging Applications of Formal Methods, Verification and Validation (isola 2006).* 2006, pp. 177–184. DOI: `10.1109/ISoLA.2006.38`.

[14] Marwa Hachicha, Riadh Ben Halima, and Ahmed Hadj Kacem. “Formal Verification approaches of Self-adaptive Systems: A Survey”. In: *Procedia Computer Science* 159 (2019). Knowledge-Based and Intelligent Information & Engineering Systems: Proceedings of the 23rd International Conference KES2019, pp. 1853–1862. ISSN: 1877-0509. DOI: `https://doi.org/10.1016/j.procs.2019.09.357`. URL: `http://www.sciencedirect.com/science/article/pii/S1877050919315583`.

[15] Chao He. “Runtime Probabilistic Model Checking Based on Incremental Method”. In: *Academic Journal of Computing & Information Science* 3 (2020), pp. 85–95. DOI: `https://doi.org/10.25236/AJCIS.2020.030212`.

[16] M. Usman Iftikhar and Danny Weyns. “ActivFORMS: Active Formal Models for Self-Adaptation”. In: *Proceedings of the 9th International Symposium on Software Engineering for Adaptive and Self-Managing Systems.* SEAMS 2014. Hyderabad, India: Association for Computing Machinery, 2014, pp. 125–134. ISBN: 9781450328647. DOI: `10.1145/2593929.2593944`. URL: `https://doi.org/10.1145/2593929.2593944`.

[17] J. O. Kephart and D. M. Chess. “The vision of autonomic computing”. In: *Computer* 36.1 (2003), pp. 41–50. DOI: `10.1109/MC.2003.1160055`.

[18] M. Kwiatkowska, G. Norman, and D. Parker. “PRISM 4.0: Verification of Probabilistic Real-time Systems”. In: *Proc. 23rd International Conference on Computer Aided Verification (CAV’11).* Ed. by G. Gopalakrishnan and S. Qadeer. Vol. 6806. LNCS. Springer, 2011, pp. 585–591.

[19] V. Nimal. “Statistical Approaches for Probabilistic Model Checking”. MSc Mini-project Dissertation. Oxford University Computing Laboratory, 2010.

[20] Karsten Saller, Malte Lochau, and Ingo Reimund. “Context-Aware DSPLs: Model-Based Runtime Adaptation for Resource-Constrained Systems”. In: *Proceedings of the 17th International Software Product Line Conference Co-Located Workshops*. SPLC ’13 Workshops. Tokyo, Japan: Association for Computing Machinery, 2013, pp. 106–113. ISBN: 9781450323253. DOI: 10.1145/2499777.2500716. URL: https://doi.org/10.1145/2499777.2500716.

[21] Hui Song et al. “Self-adaptation with End-User Preferences: Using Run-Time Models and Constraint Solving”. In: *Model-Driven Engineering Languages and Systems*. Ed. by Ana Moreira et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 555–571. ISBN: 978-3-642-41533-3.

[22] Danny Weyns et al. “A Survey of Formal Methods in Self-Adaptive Systems”. In: *Proceedings of the Fifth International C* Conference on Computer Science and Software Engineering*. C3S2E ’12. Montreal, Quebec, Canada: Association for Computing Machinery, 2012, pp. 67–79. ISBN: 9781450310840. DOI: 10.1145/2347583.2347592. URL: https://doi.org/10.1145/2347583.2347592.