Adding Self-improvement to an Autonomic Traffic Management System

Christian Krupitzer, Julian Otto, Felix M. Roth, and Christian Becker
Information Systems II, University of Mannheim, Germany
Email: {christian.krupitzer, julian.otto, felix.maximilian.roth, christian.becker}@uni-mannheim.de

Alexander Frömmgen
Multimedia Communications Lab
Technische Universität Darmstadt, Germany
Email: froemmgen@dvs.tu-darmstadt.de

***Abstract*—Autonomic Computing and self-adaptive systems are a response to the increasing complexity required to cope with changing environments and varying system resources. However, the complexity of the adaptation logic itself increases with the available information in particular for distributed systems. This leads to uncertainty at runtime resulting in incompleteness in the representation of adaptation goals, models, or rules. Self-improvement which changes the adaptation logic at runtime through meta-adaptation addresses the uncertainty issue.**

In this paper, we present and discuss a self-improvement case study for an autonomic traffic management system. We adapt *parameters* of the adaptation logic through rule learning as well as the *structure* of the adaptation logic, e.g., from central to decentralized control. We show that the resulting implementation enables continuous self-improvement of the system even in situations that have not been taken into account at design time.

I. Introduction

Autonomic Computing tries to handle the increased complexity in the development and maintenance of today's information systems. Autonomic Computing systems are ubiquitous in various domains, such as data center and cloud computing environments, as well as cyber-physical systems and autonomous robotics. They manage resources that deliver functionality to users and backend systems, and the adaptation logic or autonomic manager, respectively [1]. The adaptation logic implements a feedback structure, e.g., by following the MAPE model with monitoring, analyzing, planning, and executing parts [1]. Analyzing and planning may be subsumed as reasoning or decision [2]. Using this control cycle, the adaptation logic addresses self-* properties, such as self-configuration, self-optimization, self-healing, and self-protection [1].

Established concepts to reason about adaptations use models, rules, goals, or utility functions in the control cycle [2]. However, uncertainty at runtime can lead to incompleteness or obsolescence of goals, rules, or models as well as non-optimal utility functions. An additional self-* property denoted as *self-improvement* was proposed to tackle these issues [3]:

> *Self-improvement of the adaptation logic is the adjustment of the adaptation logic to handle former unknown circumstances or changes in the environment or the managed resources.*

A recent analysis of 12 approaches for self-improvement shows two major shortcomings of existing approaches [3]. The approaches (i) integrate either structural or parameter adaptation of the adaptation logic and (ii) focus either on reactive or proactive reasoning. In this paper, we present a case study for self-improvement. We add self-improvement based on meta-adaptation of the adaptation logic to an existing autonomic traffic management system. Our approach combines modules for parameter as well as structural self-improvement and integrates proactive as well as reactive reasoning, showing the compatibility of these concepts.

The remainder of the paper is structured as follows: Section II presents related work in the areas of traffic management systems and self-improving systems. Afterwards, we describe the use case and the design of the self-improvement layer. Section IV and V present the corresponding implementation. Section VI discusses the results of a system evaluation. Finally, Section VII concludes the paper and presents future work.

II. Related Work

Within this work, we target the domain of advanced traffic management systems in the area of *Intelligent Transportation Systems* and add self-improvement to an existing system. Hence, related work is split into two parts: (i) the area of traffic management systems and (ii) approaches to self-improvement.

In the literature, various similar or related concepts can be found, thereof: Intelligent Vehicle / Highway Systems, Intelligent Cars and Automated Highways Systems, Automated Vehicle / Highway Systems, and Smart Highways. One recent example for such systems is the EU project *Cooperative ITS Corridor* which includes traffic control management and warnings through vehicle-to-infrastructure communication [4]. Baskar *et al.* [5] provide a comprehensive overview for the field. Due to space constraints, we focus on autonomic traffic management systems, hence, traffic management systems that include a self-adaptive component. In [6], the authors propose an approach for traffic routing which integrates self-adaptation. Vromant *et al.* present an approach for coordination of control loops and show the feasibility to support self-healing in the traffic monitoring system [7]. Within the Organic Computing domain, different approaches for traffic management are presented. Tomforde *et al.* propose a self-learning traffic lights system, which optimizes single intersections [8] and provides a distributed control to connect intersections [9]. This approach is integrated into a system for resilient traffic management [10]. Gershenson proposes a solution for self-organizing traffic lights [11] based on multi-agent simulations.

Within this paper, we focus on self-improvement or evolution of the adaptation logic, respectively, as a form of

continuous improvement of self-adaptive systems. In our understanding, self-improvement offers continuous improvement of self-adaptive systems as it adjusts the adaptation logic to formerly unknown situations. Therefore, it handles situations where the current configuration of the adaptation logic performs inadequately. In [3], Krupitzer *et al.* present a comprehensive analysis and comparison of 12 existing approaches for self-improvement of the adaptation logic. The authors evaluate them based on a taxonomy of self-adaptation from [12]. The comparison indicates that almost all of the approaches add an external, centralized component for controlling the adaptation of the adaptation logic. The approaches react on different, often application-specific triggers, e.g., changes (i) in the context, (ii) in the managed resources, or (iii) triggered by the user(s), i.e., change of goals. Some approaches offer proactive adaptation, however, none of them *combine* reactive and proactive adaptation. Further, the approaches mainly focus on *either* structural or parameter adaptation. For a detailed discussion of the approaches, the reader is referred to [3]. In this paper, we will add a flexible layer for self-improvement which integrates structural and parameter meta-adaptation for system self-improvement as well as reactive and proactive adaptation to our autonomic traffic management system.

III. Autonomic Traffic Management System

In this section, we present the existing autonomic traffic management system and motivate the necessity for self-improvement.

The autonomic traffic management system controls digital traffic signs installed on a highway. The adaptation logic which controls the signs and additional self-driving vehicles is implemented with the FESAS framework [13]. The adaptation logic uses interval-based analysis and planning integrated with event-condition-action (ECA) rules that map speed limits to the current traffic flow. The system uses an interval-based policy for reducing the likelihood of adaptation oscillation, e.g., compared to rules for all possible values. Traffic cameras measure the amount of vehicles and corresponding procedures calculate the current traffic flow. As high utilization in a traffic flow with heterogeneous velocities can lead to traffic jams [14], such situations are analyzed to react accordingly by adjusting speed limits which homogenizes the traffic flow. The two lower parts of Figure 1 show the existing components, i.e., the highway and the traffic management system that represents the adaptation logic. Note that the highway is separated in sections, and each section is managed by a dedicated adaptation logic to optimally adapt the traffic flow using a static rule base. However, as traffic patterns might change abruptly due to congestion or accidents, this decentralized setting might be harmful with regard to the adaptation goal. A switch of the highway section mapping and the communication pattern in the adaptation logic, e.g., from a decentralized setting to a coordinated pattern (as presented in [15]) would enable region-wide planning with single components for analysis and planning. The lower part of Figure 1 illustrates such an example of structural adaptation as a response to an accident.

We use the *SUMO* traffic simulator to simulates the vehicles, the behavior of self-driving vehicles as well as the infrastructure with its traffic signs and traffic cameras. *TraCI*, an addition to SUMO, allows to change simulation parameters during the simulation and, therefore, to manage the traffic signs. The adaptation logic's sensors receive data through a socket-based connection to TraCI. Vice versa, the effectors of the adaptation logic send instructions to TraCI using sockets. TraCI adapts the simulation parameters accordingly, e.g., by setting a speed limit. Besides adaptations of the speed limit, the system supports hard shoulder release, re-routing suggestions, and platooning. However, within this paper, we focus on adaptations of the speed limits.

IV. Adding Self-improvement

Based on a MAPE cycle, we added a layer for self-improvement on top of the existing adaptive system.[1] This external approach (cf. [17]) and the loose coupling between adaptation logic and self-improvement layer increases the reusability and maintainability of the self-improving functionality compared to alternative designs. Additionally, this split offers the possibility to adapt the managed resources *during* adjustment of the adaptation logic. In contrary to IBM's reference model for Autonomic Computing ([1], [18]), the self-improvement layer does not orchestrate the functionality of the adaptation logic. Instead, it is responsible for adapting the adaptation logic. Figure 1 visualizes our system model.

The succeeding section presents the self-improvement layer. To distinguish the MAPE components of the adaptation logic and the ones of the self-improvement layer, we add the abbreviation *SIL* to the self-improvement layer's MAPE components. We implemented adaptation logic and self-improvement layer in Java using the FESAS framework ([13], [19]) as it offers interfaces to (i) add/disable MAPE components and (ii) change the connection between MAPE components (and hence the distribution pattern) at runtime.

As shown in Figure 1, the self-improvement layer consists of the four MAPE elements: (i) SIL Monitor, (ii) SIL Analyzer, (iii) SIL Planner, and (iv) SIL Executor. The self-improvement layer is loosely coupled with the adaptation logic through a *proxy* element which captures (i) the structure of the adaptation logic, (ii) information about the MAPE components as well as (iii) data collected from the managed resources about their internal state as well as their external context. Further, the self-improvement layer changes (i) the structure of the adaptation logic or (ii) parameters of the MAPE components, e.g., by adding new rules using the proxy. The connection between self-improvement layer and adaptation logic is managed by the SIL Sensor and the SIL Effector. In the following, we present the functionality of the different elements.

The **SIL Sensor** receives the information from the proxy. The **SIL Monitor** is started regularly and triggers an adaptation cycle for self-improvement with an adjustable frequency. As the data should be available for arbitrary analysis, including sophisticated prediction models, and its format might change due to the integration of new sensors, it is pre-processed by the monitor and written into a MongoDB database. Additionally, procedures for handling noisy sensor information may be integrated at this point. Specified service interfaces enable the developers to access the data. Further, the SIL Monitor builds a graph of the adaptation logic's structure – the so called

[1] This approach extends a previously sketched concept in [16].

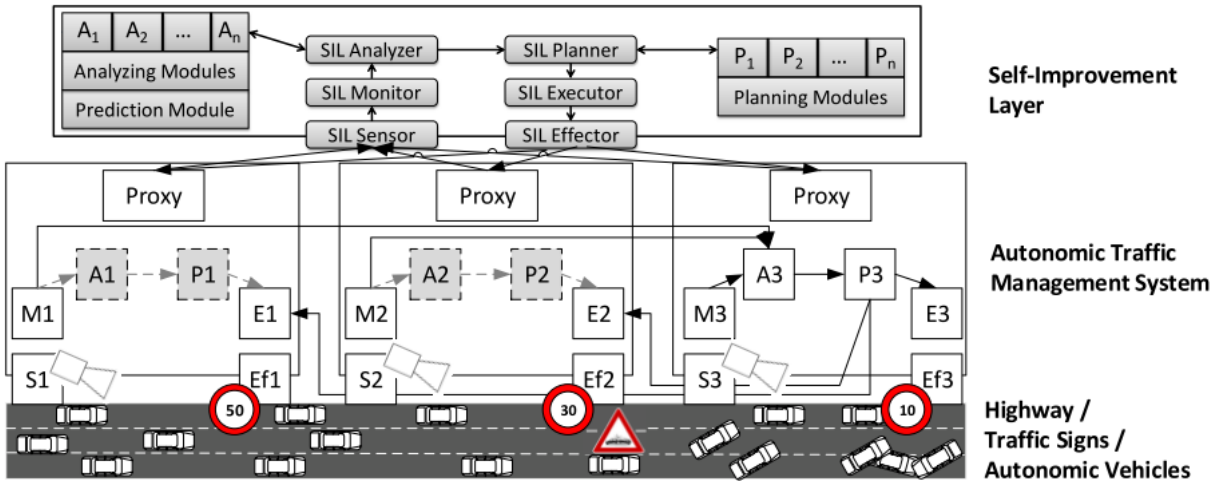

Figure 1. This figure shows the system after a structural adaptation was triggered by the self-improvement layer in response to an accident. The generated cooperative adaptation logic implements a funnel of speed limits so that vehicles approach the accident's location slowed down. Sensors are connected to traffic cameras, effectors to traffic signs. Further, the dashed elements in the adaptation logic indicate its structure before the adaptation.

`ALGraph` – that can be used for the reasoning of structural adaptations of the adaptation logic.

The **SIL Analyzer** uses the data from the managed resources to reason about their performances and characteristics. It integrates a prediction module that can perform different operations for analyzing the collected data from the managed resources. We decided to leverage the *WEKA* [20] machine learning framework, as it offers a huge variety of established classification and clustering algorithms. Whereas the use of the prediction is out of the box, some minor customization for interpreting the results can be necessary depending on the approach. Encapsulation of generic elements and clearly defined interfaces to the customization parts support the reuse of our implementation in future systems.

For the traffic scenario, we implemented reactive and predictive analysis procedures to detect abnormal traffic situations, e.g., traffic jams. We rely on time-series forecasts based on WEKA's *Support Vector Machines for Regression* implementation. These forecasts are provided by the data prediction module based on data collected from the managed resources. Further, they integrate procedures for handling outliers in the data, e.g., due to noisy sensor information. As backup, the current traffic situation is analyzed for reactively detecting unexpected traffic jams – e.g., due to an accident – impacting traffic flow and average speed metrics. The SIL Analyzer informs the SIL Planner about the analysis results, i.e., for the autonomic traffic management system, the current and forecasted traffic situations.

The **SIL Planner** integrates optimization modules for planning of self-improvement. These modules combine different types of adaptation (e.g., parameters of the MAPE components or the structure of the MAPE cycles) and might be suitable for different situations identified by the SIL Analyzer. In order to reuse the SIL Planner in other applications, generic interfaces are used within the SIL Planner for supporting adaptations of algorithms, parameters, or structure of the adaptation logic components. In accordance with literature (e.g., [12], [21], or [22]), these optimization types include most common adaptations for self-adaptive systems[2]. However, developers may add additional types or specialize the existing ones. The SIL Planner can choose at runtime between various modules for planning of self-improvement. Developers can configure different runtime modes: (i) predefined order, (ii) run all and use the fastest response, and (iii) run all and use the one with the highest utility.

After triggering the modules and receiving a plan, the SIL Planner sends the plan to the **SIL Executor**. The SIL Executor derives instructions from the plan for the different sub-systems of the adaptation logic. Using socket-based connections, the **SIL Effector** sends these instructions to the corresponding adaptation logic proxies.

V. Optimization Modules

In the following, we describe the implementation of three optimization modules used for planning self-improvement within the autonomic traffic management system: (i) dynamic parameter adaptation through rule learning, (ii) structural adaptation of the adaptation logic based on a fixed rule set, and (iii) dynamic model-based structural adaptation.

A. Parametric Self-improvement

One of the implemented modules offers an approach for continuously adapting the adaptation logic's rule base with bespoke ECA rules. ECA rules provide a comprehensive representation of adaptation logic [23]. In our example use case, the conditions are traffic flows and average speed of vehicles that are matched with corresponding speed limits. The learning module integrates a simulation of the highway which is constantly started with new parameter combinations to compare different actions for a given condition. As a next step, a utility function is applied to the collected data of the simulation runs. The function assigns a utility value to each simulation result:

[2]In this more technical view, behavioral adaptation is seen as the outcome of adapting parameters or structure (cf. [12]).

$$U_{Highway} := \alpha - \left(\beta * \sum wait + \sum lostTime\right) \quad (1)$$

$\sum wait$ is the accumulated number of simulation steps where the speed of a vehicle was below 0.1 m/s, $\sum lostTime$ is the accumulated number of seconds that each vehicle lost due to driving below its target speed. Both parts of the utility function, `wait` and `lostTime`, grow with the size of traffic jams. These values are logged by the SUMO simulation for every vehicle of the simulation. Simulation runs with different weighting factors β lead to a weighting factor of 7 for a balanced utility function in our highway scenario. However, for having a correlation between a large result and a high utility, the function subtracts the sum of the two parts from a constant factor α. The simulation runs showed that the sum does not exceed 1 000 for the track used in the simulation, therefore, we chose 1 000 for the factor α. It may vary for other tracks. The module then uses the speed limit with the highest utility and constructs a new rule consisting of the average flow per lane as the condition and the speed limit as the action. The SIL Planner sends this rule to the SIL Executor which triggers an update of the rules in the adaptation logic.

B. Structural Self-improvement

In our use case, the structure of the adaptation logic and the highway as managed resource can be modeled as an annotated graph. Accordingly, changes in the adaptation logic and the managed resources lead to changes in the graph. Therefore, we implemented two graph-based modules that offer rule-based and model-based planning, respectively, for structural adaptation of the adaptation logic. They specify when to switch the coordination pattern of the adaptation logic as shown in Figure 1 where a switch from a non-coordinated setting to a regional planner coordination pattern (cf. [15]) was triggered. However, the reasoning approach is different for both modules. In this section, we present a graph-based structural adaptation approach using a static rule base and *the Topology Adaptation Rule Language* (TARL) [24] and an alternative variant using a dynamic rule base implemented with *Neo4j*[3].

1) Rule-based Structural Self-improvement using TARL: TARL offers a general topology adaptation model. It describes the topology of the adaptation logic in a rule-based manner, separating monitoring, reasoning (named *adaptation logic* in TARL), and execution of the adaptation. The advantage of TARL rules compared to other approaches is the support for decentralized reasoning. A TARL rule consists of a condition specification and an execution specification. The condition contains (potentially multiple) graph patterns and matches those on the input graph. The corresponding execution statement specifies the changes to the input graph. If a condition matches, the corresponding execution statement is triggered. A first prototype of the TARL execution engine is presented in [24]. In contrast to the previously introduced dynamic rule learner, the current state of the adaptation logic (e.g., its distribution) is relevant for structural adaptation of the adaptation logic. The TARL module compares the adaptation logic's structure with patterns that are defined in the TARL rules. The patterns integrate the current or predicted traffic situation. If the TARL module finds corresponding patterns, it returns the associated adaptation actions of the matching rules. These are handed to the SIL Planner to execute the adaptation logic adaptations.

2) Model-based Structural Self-improvement using Neo4j: We implemented a second variant using the graph database Neo4j (called Neo4j module in the following). The Neo4j module gets informed about the current structure of the adaptation logic. It uses the information of the `ALGraph`, the data captured from the managed resources, as well as the analysis of the predicted and the current traffic situations to create queries using pre-defined but customizable rule templates. These queries change the graph of the adaptation logic's structure which is stored in the graph database. The changes include the addition/removal of MAPE components, changes of their functionality, or the connections between the MAPE components, hence, the adaptation logic's structure. The Neo4j module then compares the current structure of the adaptation logic with the version stored in the database after performing the queries. Differences lead to plan instructions for adapting the adaptation logic, so that it fits the proposed structure.

VI. Evaluation

For evaluating the feasibility of our approach, we performed various simulations of a part of the German highway A8 between the interchange Stuttgart and the junction Stuttgart-Degerloch near Stuttgart airport. We used real traffic datasets from the authors of [25] and the German Federal Highway Research Institute (BASt). During the simulation, the volume of traffic is adjusted hourly to reflect a typical working day. Our simulated highway includes a track of around 10 kilometers. We divided it into 4 sections. In the first scenario, we analyzed the rule learning component. As second scenario, we evaluated how the two modules for structural adaptation cope with changes due to an accident as well as a daily road work. Due to the different time horizons of the modules – continuous rule learning vs. spontaneous structural adaptation – we separated the tests of the runtime optimization modules. As we focus on reducing traffic congestions, we evaluated the average time that vehicles spent in traffic congestions as performance measurement. The waiting time is the time where the speed of a vehicle was below 0.1 m/s. Hence, the lower the number is, the better it is as vehicles spend less time in traffic jams. We calculated the average waiting time for five minute intervals based on all vehicles that started during that interval in a simulation run. To get more reliable values that are not influenced by random deviations, we repeat the simulation runs 50 times. For a final aggregated value, we calculate the average of the waiting time for an interval over all runs. To compare entire settings, the integral of the values is calculated using the Romberg method. For both scenarios, we compare the adaptation logic's performance with a fixed rule set to the self-improvement modules. A subsystem controls one section of the highway independently. In the following, we present the results of the test runs.

A. Baseline Measurements

In order to have baseline measurements for comparing the adaptation logic's performance, we first ran two series of 50 simulations each without an adaptation logic as well as using a fixed rule set for both tracks. The self-improvement layer is inactive. Hence, there are no rules added during the

[3]Neo4j website: https://neo4j.com/

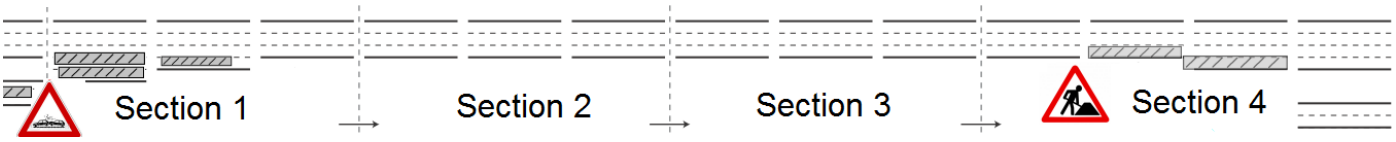

Figure 2. An overview of the highway that is controlled by the system. Because of the construction site in section 4 and an accident in section 1, the gray marked areas are closed in the second setting for evaluation of structural self-improvement.

simulation and structural self-improvement is not active. The adaptation logic set different speed limits. The speed limit is set to unrestricted if the highway is empty. In case of dense traffic, a limit of 120 km/h is set and finally, for stop-and-go traffic and traffic jams the speed limit is reduced to 80 km/h. Dense traffic, stop-and-go traffic situations, and traffic jams are detected using measurements of the amount of vehicles on the track as well as their average speed. However, as traffic jams happen even with the adaptation logic, this indicates that the adaptation logic could be improved though meta-adaptation.

B. Scenario 1: Parametric Self-improvement

In the first scenario, we measured the performance of the parametric self-improvement, hence, the rule learner. The recommended speed limits are learned while the system is executed. The system starts with an empty rule base and the rule base evolves over time. Figure 3 shows the measured waiting times. For comparison, we add the results of the baseline evaluation without and with a static adaptation logic.

When looking at the progression of the two curves of Figure 3 in detail, both start with the same performance. At the beginning of the day, the highway is free. Hence, no speed limits are set. As new rules are learned, the performance of the online learner increases. For the first fifteen and the last four hours of the day, the performance of the parametric self-improvement is constantly better than the baseline with a static adaptation logic. This means that for the traffic conditions in this time frame the learning module found better speed limits than the rules of the static adaptation logic. It can also be concluded that traffic jams form later and resolve earlier which means that the traffic remains longer in a flowing state. However, at around 4pm, the fixed rules outperform the rule learner. A reason for that might be that the rule learner cannot react fast enough to the fundamental change in the pattern. The integrals of the baseline without the adaptation logic is 21 308, the one with a fixed rule set decreases to 14 950. Using the rule learner, the integral of the measurement decreases by 1 139 points which is a 7.6 % improvement compared to the fixed rule set and an improvement of 35 % compared to the scenario without adaptation logic. We conclude that aggregated over the day the rule learner outperform the static adaptation logic.

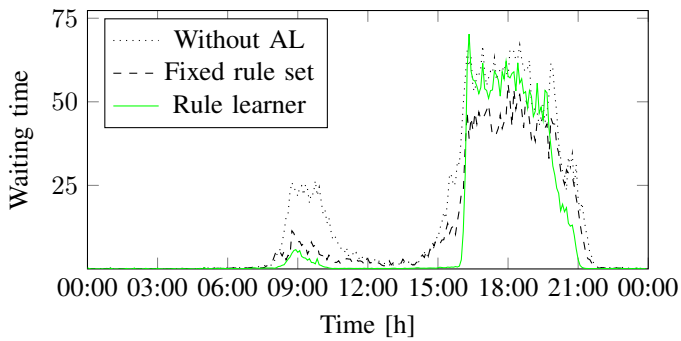

Figure 3. Measurements for parametric self-improvement compared to the baseline measurements. AL stands for adaptation logic.

C. Scenario 2: Structural Self-improvement

It is not preferable to learn rules for a spontaneous, non-durable event as learning rules involves timely and costly simulations. Contrary, structural adaptation reacts fast to changing conditions. Therefore, we introduce additional events that trigger structural self-improvement. Reaction to events is not limited to reactive adaptation only, but includes proactive adaptation as reaction to forecasted events. We simulate an accident on the second highway in section 1. Further, as structural adaptation is event-based, we simulated having a daily road work in the last section of the highway during the morning rush hour between 6am and 11am. The blocked parts of the track are shadowed in Figure 2.

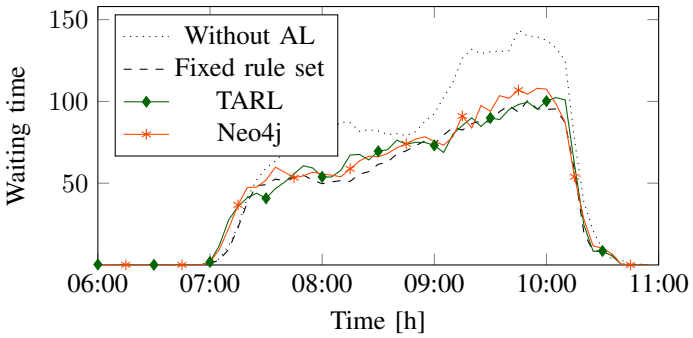

Figure 4. Measurements for the structural self-improvement compared to the baseline measurements. AL stands for adaptation logic.

We configured the following parameters of the prediction module for traffic forecast: size of the training set (1 000), amount of forecasted time steps (200), and classifier (time series forecast based on WEKA's *Support Vector Machines for Regression* implementation). We performed 25 runs using the Neo4j module and 25 runs using the TARL module. The integral of the baseline without the adaptation logic is 18 128, the one with the static adaptation logic decreases to 13 032. The Neo4j module has an integral of 14 055, the TARL module performs slightly better with 13 713. Confirmed by Figure 4, both approaches for structural self-improvement perform similarly to the static adaptation logic.

Both Neo4j and TARL offer structural self-improvement. The static rules of TARL always connect 3 sections to a region with a central planner, whereas the flexibility in the Neo4j modules enables to connect between 2 and 4 sections. Therefore, the Neo4j module offers more flexibility than the static TARL rules, however, the flexibility comes at the cost of performance. Whereas TARL is designed for resource poor devices, the Neo4j database needs fully-fledged devices. However, both do not

improve the system in terms of waiting times. We use the same rules as the static adaptation logic and, further, set the same speed limit for the whole region. Additionally, the parameters of the prediction modules are not optimized for the planning modules. Therefore, the following optimizations are possible: (i) learn new rules for the situation, (ii) optimize the speed limits with variable speed limits for the sections of a region, and (iii) optimize the parameters for prediction. However, the created regions help to homogenize the traffic flow which increases safety of traveling in situations of dense traffic and helps to save fuel [14]. Additionally, they still improve the traffic compared to the situation without an adaptation logic by more than 23 %. The Neo4j module needs on average 181 ms for the adaptation (minimum: 19 ms; maximum: 1 822 ms) whereas the TARL module has an average processing time of 16 544 ms for the adaptation (minimum: 11 043 ms; maximum: 40 105 ms). As the authors claim in [15], the reaction time increases in higher layers of hierarchical feedback loop structures. Whereas the adaptation logic reacts very fast, the reaction time for self-improvement is larger. However, the changes performed by the self-improvement layer are more substantial. Hence, even the maximum of 40 seconds by the TARL module is acceptable, give the fact, that this is not a real time system. Additionally, the adaptation logic still adapts the highway regardless the self-improvement layer. Further, the authors of TARL provided only a prototype implementation, leaving potential for optimizations [24]. Additionally, TARL is optimized for decentralized settings, which are out of this paper's scope.

VII. Conclusion

In this paper, we presented a case study to add self-improvement to an autonomic traffic management system. We implemented three modules for planning self-improvement as well as an analyzing module for our autonomic traffic management system. All modules are customizable through minor configurations. The evaluation showed that the rule learner for parametric self-improvement increases the system performance. The modules for structural adaptation do not reduce the time wasted in traffic jams. However, as they homogenize the traffic flows, they offer benefits regarding safety aspects and fuel efficiency.

This work provides a first step for understanding self-improvement. For future work, we plan to formalize the design of the self-improvement layer and integrate the customization process into the FESAS IDE [13]. Further, we plan to evaluate the approach in additional applications and in decentralized settings. This will enable us to analyze issues regarding safeness of adapting the adaptation logic, e.g., inconsistent states or lose of data (which is less relevant for the traffic management system) as well as compare issues of decentralized and centralized settings.

Acknowledgment

This work was supported by the Julius-Paul-Stiegler-Memorial-Foundation. The authors would like to thank their former student Fabian Kajzar for his contribution. Parts of this work have been funded by the German Research Foundation (DFG) as part of projects A4 and C2 within the Collaborative Research Center (CRC) 1053 – MAKI.

References

[1] J. O. Kephart and D. M. Chess, "The Vision of Autonomic Computing," *IEEE Computer*, vol. 36, no. 1, pp. 41–50, 2003.

[2] P. Lalanda, J. A. McCann, and A. Diaconescu, *Autonomic Computing - Principles, Design and Implementation*. Springer, 2013.

[3] C. Krupitzer, F. M. Roth, M. Pfannemüller, and C. Becker, "Comparison of Approaches for Self-Improvement in Self-Adaptive Systems," in *Proc. ICAC*, 2016, pp. 308–314.

[4] P. E. Ross, "Thus Spoke the Autobahn," *IEEE Spectrum*, 2015.

[5] L. D. Baskar, B. D. Schutter, J. Hellendoorn, and Z. Papp, "Traffic control and intelligent vehicle highway systems: a survey," *IET Intelligent Transport Systems*, vol. 5, no. 1, pp. 38–52, March 2011.

[6] J. Wuttke, Y. Brun, A. Gorla, and J. Ramaswamy, "Traffic Routing for Evaluating Self-Adaptation," in *Proc. SEAMS*, 2012, pp. 27–32.

[7] P. Vromant, D. Weyns, S. Malek, and J. Andersson, "On interacting control loops in self-adaptive systems," in *Proc. SEAMS*, 2011, pp. 202–207.

[8] S. Tomforde, H. Prothmann, F. Rochner, J. Branke, J. Hähner, C. Müller-Schloer, and H. Schmeck, "Decentralised Progressive Signal Systems for Organic Traffic Control," in *Proc. SASO*, 2008, pp. 413–422.

[9] H. Prothmann, F. Rochner, S. Tomforde, J. Branke, C. Müller-Schloer, and H. Schmeck, "Organic Control of Traffic Lights," in *Autonomic and Trusted Computing*, 2008, pp. 219–233.

[10] M. Sommer, S. Tomforde, and J. Hähner, "An organic computing approach to resilient traffic management," in *Autonomic Road Transport Support Systems*, 2016, pp. 113–130.

[11] C. Gershenson, "Self-organizing Traffic Lights," *Complex Systems*, vol. 16, no. 1, pp. 29–53, 2005.

[12] C. Krupitzer, F. M. Roth, S. VanSyckel, G. Schiele, and C. Becker, "A survey on engineering approaches for self-adaptive systems," *Pervasive and Mobile Computing Journal*, vol. 17, no. Part B, pp. 184–206, 2015.

[13] C. Krupitzer, F. M. Roth, C. Becker, M. Weckesser, M. Lochau, and A. Schürr, "FESAS IDE: An Integrated Development Environment for Autonomic Computing," in *Proc. ICAC*, 2016, pp. 15–24.

[14] M. Treiber and A. Kesting, *Traffic Flow Dynamics - Data, Models and Simulation*. Springer, 2013.

[15] D. Weyns, B. Schmerl, V. Grassi, S. Malek, R. Mirandola, C. Prehofer, J. Wuttke, J. Andersson, H. Giese, and K. M. Göschka, "On Patterns for Decentralized Control in Self-Adaptive Systems," in *Software Engineering for Self-Adaptive Systems II*, 2013, pp. 76–107.

[16] F. M. Roth, C. Krupitzer, and C. Becker, "Runtime evolution of the adaptation logic in self-adaptive systems," in *Proc. ICAC*, 2015, pp. 141–142.

[17] J. Floch, S. Hallsteinsen, E. Stav, F. Eliassen, K. Lund, and E. Gjorven, "Using architecture models for runtime adaptability," *IEEE Software*, vol. 23, no. 2, pp. 62–70, 2006.

[18] IBM Corporation, "An Architectural Blueprint for Autonomic Computing," Tech. Rep., 2005.

[19] C. Krupitzer, F. Roth, S. VanSyckel, and C. Becker, "Towards Reusability in Autonomic Computing," in *Proc. ICAC*, 2015, pp. 115–120.

[20] G. Holmes, A. Donkin, and I. H. Witten, "WEKA: a machine learning workbench," in *Proc. ANZIIS*, 1994, pp. 357–361.

[21] P. K. McKinley, S. M. Sadjadi, E. P. Kasten, and B. H. C. Cheng, "Composing adaptive software," *IEEE Computer*, vol. 37, no. 7, pp. 56–64, 2004.

[22] M. Handte, G. Schiele, V. Matjuntke, C. Becker, and P. J. Marrón, "3PC: System Support for Adaptive Peer-to-Peer Pervasive Computing," *ACM TAAS*, vol. 7, no. 1, p. Article 10, 2012.

[23] A. Frömmgen, R. Rehner, M. Lehn, and A. Buchmann, "Fossa: Learning eca rules for adaptive distributed systems," in *Proc. ICAC*, 2015, pp. 207–210.

[24] M. Stein, A. Frömmgen, R. Kluge, F. Löffler, A. Schürr, A. Buchmann, and M. Mühlhäuser, "TARL: Modeling topology adaptations for networking applications," in *Proc. SEAMS*, 2016, pp. 57–63.

[25] J. Schlaich and M. Friedrich, "Staumeldungen und routenwahl in autobahnnetzen – teil 1: Analyse von staumeldungen," *Straßenverkehrstechnik*, vol. 14, no. 10, pp. 621–627, 1999.