First TAB-Report

Ilja Shmelkin
ilja.shmelkin@tu-dresden.de
Faculty of Computer Science
Technische Universität Dresden
Dresden, Germany

ABSTRACT

Self-adaptive Systems (SASs) are one way to address the ever-growing complexity of software systems by allowing the system to react on changes in its operating environment. In today's systems, self-adaptation is typically realized with a control loop, for which the MAPE-K feedback loop is a prominent example. Research uses the notion of patterns to describe the distribution and decentralization of individual control loop components or control loops and their underlying managed subsystems. While there are some well-accepted standards about which components a managed subsystem has to implement so that it can interact with the control loop, research still lacks detailed investigation of control mechanisms, protocols, message formats and trade-offs for communication within and across control loops. This paper provides an overview of ongoing research in those topics and how it relates to the doctoral project. For this, first, related work that is used as a baseline is presented briefly. Then, the process of deriving uniform communication interfaces and communication protocol requirements will be shown. Furthermore, a list of interesting communication protocol semantics will be highlighted. After that, past and upcoming contributions, as well as the first research results of the project, will be announced. Finally, the remaining tasks of the doctoral project are scheduled in a timeline.

CCS CONCEPTS

• **Computer systems organization** → **Distributed architectures**;
• **Networks** → *Network performance evaluation.*

KEYWORDS

self-adaptive systems, MAPE-K, SNMP, control loop, benchmarking

1 INTRODUCTION

Self-adaptive systems (SASs) can adapt themselves by reacting on changes in their operating environment, their goals, failure of individual components, resource demand, and many more. Typically SASs consist of multiple components that interact with each other in a way that is defined during system design time. Most of them use *control loops* to reason about and organize adaptations of an underlying *managed subsystem* of which the MAPE-K (Monitor, Analyse, Plan, Execute, and Knowledge) control loop is one prominent example [3]. The authors use the terms *autonomic manager* to refer to the (intelligent) control loop and *managed resources* for the system which comprises the application logic and provides the actual SASs functionality. Other researchers use different terms (e.g. *architecture layer*, *reflective subsystem* or *adaptation engine* for the autonomic manager and *system layer*, *base-level subsystem* or *core function* for the managed resources) to make the same distinction [5, 8, 12]. As Weyns et al. [13] state, it makes sense to avoid confusion by sticking to the terms *managing subsystem* (i.e., the control loop) and the *managed subsystem*. Further, they clarify the distinction between a *distributed* and a *decentralized* SAS from which the former refers to the physical distribution of an SAS across multiple nodes which are connected through a computer network, and the latter defines how control decisions within an SAS are made, based on information communicated between individual components of the control loop. To better understand the decentralization aspect of SASs, the authors introduce *patterns* that describe the communication paths between controlling components of the SAS, whereby the components that communicate can be individual parts of a control loop (e.g. only monitoring components communicate with other monitoring components), individual control loops with each other, nested loops [6] or even sub-loops where the interaction is not bound to the logical sequence of the MAPE-K control loop [10]. While there are some well-accepted standards about which components a managed subsystem has to implement (e.g. *sensors* and *effectors*, however, different names exist) so that they can interact with the managing subsystem [13], the community still lacks detailed investigation of control mechanisms, protocols, message formats and trade-offs for communication *within* the control loop or *between* control loops in cases where the SAS is distributed and/or decentralized [11, 13]. This opens multiple research questions, from which the following three will be solved during the doctoral project:

(1) How can we define uniform communication interfaces and protocols for distributed self-adaptive systems that rely on the MAPE-K control loop?
(2) When multiple distributed MAPE-K loop components of the same type have to interact with each other to create a control decision, how do they synchronize?
(3) How can we use the monitoring knowledge from earlier MAKE-K iterations to enhance the overall adaptation performance?

The process of solving (1), as well as the first results, will be presented in this paper. To solve (2) in the future, the role concept as well as leader election algorithms will be used. For solving (3), a feature called *direct adaptation* will be introduced in the future.

The notion of patterns for self-adaptive systems provides a good basis to further assess open research tasks and, therefore, will be used extensively in the remainder of this paper, which is structured as follows. Section 2 summarizes the idea of patterns in SASs, introduces three concrete examples (centralized, decentralized and distributed) to the reader, and shows which kind of data is transmitted in managing subsystems. Based on that, interface descriptions will be presented and later used to derive communication protocol requirements for self-adaptive systems. The section is concluded by

using those requirements and a list of interesting protocol semantics to compare multiple protocols. Section 3 introduces one specific scenario which is then used to create a benchmarking system that can be exploited to compare the quantitative performance of different communication protocols in a self-adaptive environment. With that, the first results of the doctoral project will be presented. Section 4 summarizes thoughts on the open research questions (2) and (3) which will be covered in later contributions and concludes this paper.

2 TOWARDS STANDARDIZED COMMUNICATION IN SELF-ADAPTIVE SYSTEMS

Ever since SASs have been identified as an opportunity to tame the growing complexity in software systems, control loops were a popular approach to engineer them [1, 4, 7]. One prominent example of a control loop is MAPE-K, which was originally introduced by IBM [3]. The loop consists of four components and a knowledge repository, whose functions will be briefly introduced in the following:

Monitoring. Collection of metrics and performance data from the underlying managed subsystem and the execution environment. Synchronisation with other monitoring components in decentralized or distributed SASs.

Analysis. Algorithmic processing of monitored data to gain knowledge about the state of the underlying subsystem. Decision making if the underlying subsystem has to be adapted.

Planning. Creation of an *adaptation plan* that consists of one or multiple activities that cause the underlying subsystem to change its behaviour according to the prior calculated adaptation decision.

Execution. This component has a direct connection to the underlying managed subsystem. It executes the prior created adaptation plan.

Knowledge. Ideally, in each MAPE-K sequence iteration, every component gathers information about the underlying subsystem or the loop itself that can be used in future iterations. This information is referred to as *knowledge* and can be present either as an information database (i.e., knowledge repository) or implicitly within each component, however, we stick to the prior definition.

2.1 Patterns for Distributed Control-loops

Large scale systems frequently orchestrate multiple subsystems so that they work towards a common goal. Therefore, modelling large systems with only one control loop often is not sufficient. To better understand recurring architectures of SASs, Weyns et al. [13] introduced a set of patterns with a simple graphical representation as reusable solutions for systems with multiple MAPE-K loops. The patterns describe how individual control loop components (M, A, P, E, K) or whole loops interact with each other. Furthermore, the author's distinguish between the pattern as an *abstract representation* and its specific *instance*, as vividly depicted in figure 1. The former describes abstract groups of MAPE-K components, how they interact with each other and the underlying managed subsystem. The latter represents one specific configuration for a particular SAS.

Although MAPE-K was originally introduced as a hierarchical approach, where each level of the hierarchy contains instances of all five MAPE-K components, in practice, numerous different approaches exist. Their complexity and associated challenges grow with the grade of distribution and decentralization, however, we believe that most of them can be classified as a member of one of the following groups. As an example, each group will be represented by an abstract pattern together with one concrete instance. Note that, as all components can send data to a knowledge repository, it was omitted in the figures.

2.1.1 Fully centralized. All MAPE-K components of an SAS, that is member of this group, are contained in one single node or server. All control decisions are coordinated between components in one single node or server. There is no need to replicate or partition data over a computer network. We call such SAS *fully centralized* in view of control decisions and not distributed.

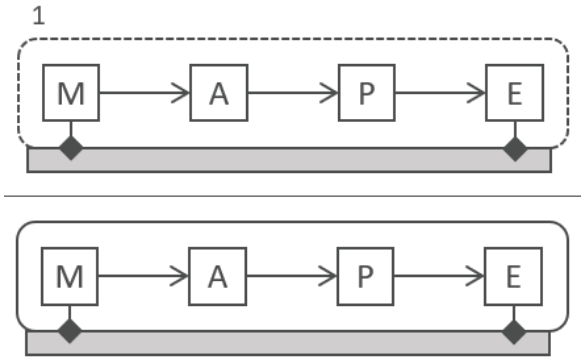

Figure 1: Abstract (dotted line) pattern for a centralized SASs and its single possible instance (solid line).

2.1.2 Distributed with centralized components. Members of this group have components which are *physically distributed* across multiple nodes and therefore require communication across a computer network. Depending on the actual pattern that is present, data may be *partitioned* or has to be *replicated* across the network. Next, centralized and *hybrid* MAPE-K components can exist (as seen with Analysis, Planning and Monitoring), as depicted in figure 2. As shown, two nodes (i.e., managing subsystems) each monitor an underlying subsystem, however, metrics and performance data get transmitted to another node with centralized components (M, A, P), which analyzes the metrics and performance data from the whole system and calculates an adaptation plan. This plan is distributed back to the managing subsystems, where it is executed by the respective components.

2.1.3 Fully distributed. An SAS is fully distributed when each of its nodes contains all components of the MAPE-K sequence, as depicted in figure 3. Individual communication links between the components show how control decisions are made. Therefore, in

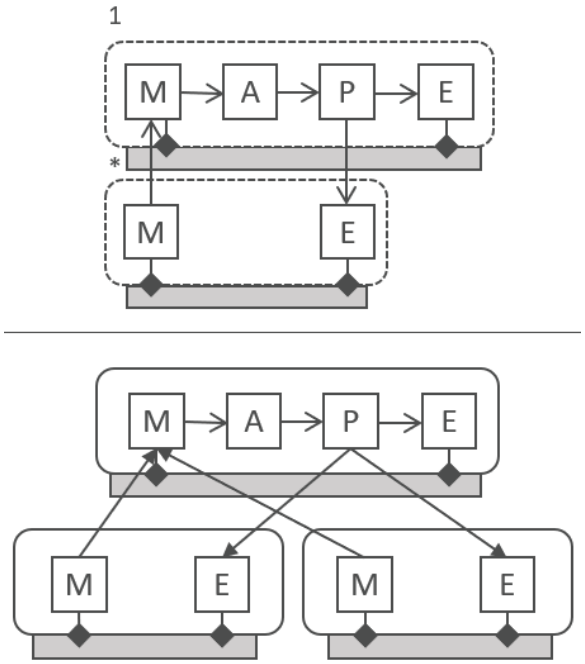

Figure 2: Abstract (dotted line) pattern for a distributed SAS with hybrid monitoring and centralized analysis and planning components and one possible instance (solid line).

the example below, metrics and performance data are *replicated* between the monitoring components and planning decisions between the planning components respectively. When, as seen in figure 3 with the analysis components, no data is communicated between respective components, we denote this data as *partitioned*.

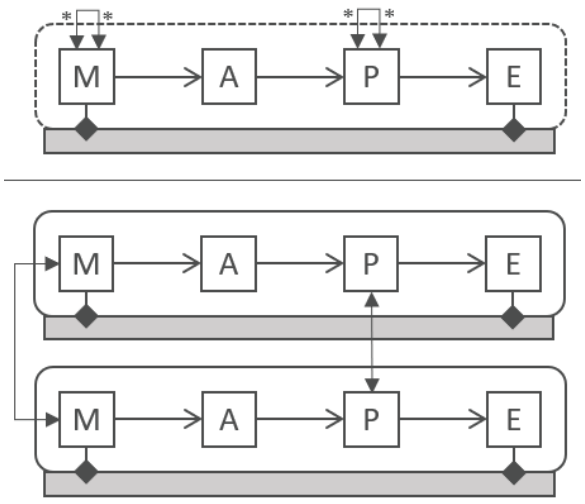

Figure 3: Abstract (dotted line) pattern for a distributed SASs and one possible instance (solid line).

While the dataflow in fully centralized and not distributed MAPE-K loops is rather simple (i.e., a local database), distribution and decentralization introduce multiple challenges: (1) When a MAPE-K loop is distributed, data, that is necessary to compute control decisions, has to be transmitted between nodes. While this is a known challenge in the community of SASs [11, 13], yet no definition of proper communication interfaces exists. (2) When multiple distributed loop components of the same type have to interact with each other to create a control decision (see monitoring and planning in figure 3), how do they synchronize? (3) Should data be fully replicated between all nodes or is it sufficient to elect one node that accumulates all data and calculates a control decision globally? (4) To minimize data traffic, is it sufficient to create local control decisions and calculate a global control decision based on that? (5) How do we secure the communication between MAPE-K components in distributed and decentralized cases?

2.2 Data traffic in distributed control-loops

To solve challenges (1), (5) and partially (3), the exact data, which is transmitted throughout the control loop has to be identified for each component. As depicted in figure 4, different control data has to be present in each step throughout a control loop iteration. Note, that encryption and authentication are not mandatory in a fully centralized SAS. However, we strongly encourage to use them for all communication links by default in fully centralized, as well as in distributed and decentralized cases. The following explanation bases on the incoming and outgoing data of each control loop component:

Monitoring.

- Incoming: Either via pull or push, the monitoring component collects metrics and performance data from the underlying managed subsystem(s) (IF M.2) or the environment (IF M.3). In decentralized scenarios, it may collect the data from another monitoring component (IF M.0). Based on the recurring nature of the control loop, the gathered data can be classified as a time-series. Regarding the control loop as a whole, the monitoring component accumulates the biggest portion of all processed data.
- Outgoing: Either via pull or push, the data transmitted by the monitoring component can either be time-series metric data, if the receiver is another monitoring component (IF M.0) or a *query result* if the receiver is an analysis component (IF M.1).

Analysis.

- Incoming: Either via pull or push, the analysis component can receive two types of data. First and foremost, it can receive query results from a monitoring component (IF M.1), which are calculated based on the time-series metrics gathered from the managed subsystem. Next, it can receive control decisions from another analysis component (IF A.0), in cases where the control loop is decentralized. Either way, the data has a rather small volume.
- Outgoing: Analysis components communicate control decisions based on analyzed query results, however, the receiver can be either another analysis component (IF A.0) in cases

where the analysis components are decentralized or a planning component (IF A.1) as by the sequence of a MAPE-K control loop.

Planning.

- Incoming: Based on calculated control decisions, analysis components send data about whether an adaptation is needed to the planning component (IF A.1). As the sent data spans only the decision and probably additional meta-information the data volume stays small. In decentralized cases, planning components have to actively synchronize each other to find suitable adaptation plans (IF P.0). That, however, is a research field on its own and will not be covered in this contribution.
- Outgoing: The outgoing traffic of planning components is limited to a collective finding of a suitable adaptation plan (IF P.0) as well as the communication of that plan to execution components in the SAS (IF P.1).

Execution.

- Incoming: The execution of adaptations, especially in distributed environments, is a complicated process and still under research. However, simplified, the communication should be limited to adaptation synchronization between multiple execute components in distributed scenarios (IF E.0) and an incoming adaptation plan as by the sequence of a MAPE-K control loop (IF P.1).
- Outgoing: The communication between an execution component and its underlying managed subsystem (IF E.2) depends on numerous factors. This factors can be the operating system of the managed subsystem, the adaptation goal as well as information about the success or failure of the desired adaptation, for example. This part of communication is not in the scope of this paper and should be addressed in another contribution.

Knowledge.

- Incoming: As depicted in figure 4, the knowledge repository can accumulate data about all processes in the MAPE-K feedback loop. Therefore, it can receive data from every other component (IF K.1), which it has to organize in a meaningful way. To gain knowledge, other loop components can query the repository. Next, as knowledge repositories can be decentralized as well, replication or structured partitioning of data between individual knowledge repositories can be a big part of the overall traffic.
- Outgoing: The outgoing traffic is limited to the communication of query results to other loop components and replication or partitioning of data between knowledge repositories that are not specified precisely at the moment.

In summary, we can identify the following interface functions between communicating control loop components:

(1) Requested transfer of data (pull)
(2) Unrequested transfer of data (push)
(3) Bulk-transfer or streaming of data to transmit metrics and performance data
(4) Synchronization between individual components
(5) State-of-the-art encryption and authentication mechanisms to guarantee confidentiality and integrity

With those requirements in mind, suitable communication protocols can be chosen which fit the interface descriptions presented earlier. Note, that the interfaces do not exactly specify the way data is transmitted but rather provide a vague description of the expected functionality. In fact, there may be protocols which differ significantly regarding their semantic properties (e.g., with/without flow control, connection-oriented/connection-less) but still fulfil the requirements. However, different semantics may provide advantages in specific infrastructures, e.g., message queue protocols allow to queue and later retrieve important data when parts of the self-adaptivity logic are not available at a given time, wherein contrast, using the connection-less "*fire-and-forget*" UDP protocol would result in a loss of data, that, in some cases may be beneficial whatsoever. Therefore, no existing protocol fulfils all requirements to the maximum degree. All existing solutions have their pros and cons and thus, an investigation will allow the identification of best candidates that provide the best performance and/or semantics for an interface within a specific self-adaptive system's control layer. In the remainder of this section, five communication protocols will be listed and some of their semantic properties shown. The presence or absence of a semantic property can both benefit different types of self-adaptive systems. This subject as well as the list are ongoing research and, therefore, neither complete nor final. A summary of supported semantics for each protocol is listed in Table 1. In early 2021, a paper will be submitted which discusses each of the semantic properties and how those protocols may benefit SASs in the future.

Table 1: List of semantic properties for communication protocols. Checkmark (supported), cross (not supported), circle (partially supported).

	TCP	UDP	SNMP	gRPC	ZeroMQ
Connection establishment	✓	×	×	✓	✓
Flow control	✓	×	×	✓	✓
Congestion control	✓	×	×	✓	✓
Transmission failure handling	✓	×	✓	✓	✓
Message queueing	×	×	×	×	✓
Authentication	×	×	✓	✓	✓
Encryption	×	×	✓	✓	✓
Broadcast	×	✓	⊙	⊙	✓
Multicast	×	✓	✓	⊙	✓
Synchronous communication (API)	✓	×	✓	✓	✓
Asynchronous communication (API)	✓	✓	✓	✓	✓
Integrity check	✓	✓	✓	⊙	✓
In order processing	✓	×	×	✓	⊙
Licensing	×	×	×	Apache 2.0	GNU LGPL3

3 BENCHMARKING THE COMMUNICATION IN SELF-ADAPTIVE SYSTEMS

To allow the comparison of communication protocols regarding their quantitative performance properties, a benchmarking infrastructure has been created. The infrastructure resembles a real-world scenario in which clients access a website. Depending on the number of accesses, the infrastructure adapts the number of web servers within a load-balancing configuration in a self-adaptive manner, to provide accessibility to the website even when the number of

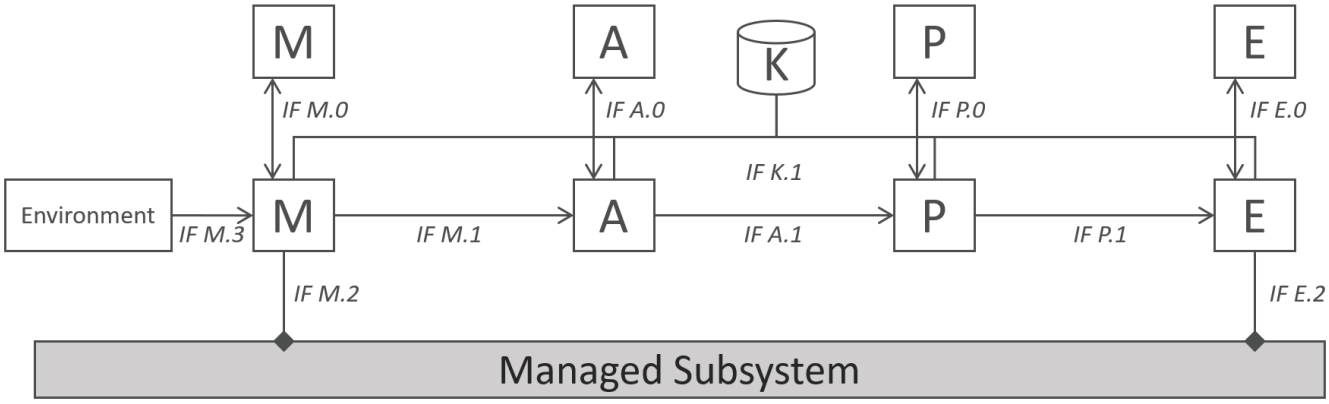

Figure 4: Intra-communication within the MAPE-K control loop based on a centralized and not distributed example.

users peak (which would overwhelm a single server). This scenario was initially introduced by Cheng et al. [2] which used the Rainbow framework [5] to provide that functionality. As the doctoral project focuses on a more general contribution, the infrastructure was recreated with a more generic MAPE-K control loop. The prototype of this infrastructure is described in a contribution for the [redacted for the reason of a double blind review] in more detail. The main idea, however, is to allow to monitor specific performance properties of the communication protocol as well as the overall performance of the control- and managed-system layer while choosing the communication protocol that is used throughout the control layer from a list of options.

To evaluate the performance of the protocols in action, the Apache JMeter program is used to artificially put a load on the web server infrastructure. For each protocol, two experiments are executed. In the first experiment, self-adaptation is disabled to provide comparable results to the reader, however, the remaining procedure is the same. For the second experiment, adaptation is enabled. In both experiments, each of the ten client virtual machines (16 CPU cores and 16GB of RAM each) creates a dynamic number of requests per second (RPS), that in total match the blue line in figure 5 over time, resulting in a peak of 12000 combined RPS with a step-wise increase/decrease from/to 0 RPS. The goal of this procedure is to validate the self-adaptiveness of the implemented system with the expectation that additional servers are incorporated as load increases and deactivated with decreasing load, as well as showing the actual benefit of using the infrastructure to head off intolerable load to other servers. Each experiment is executed in a time window of exactly 90 seconds. Further, to evaluate the performance of each of the protocols for communication between distributed MAPE-K control loop components, the communication effort for the IF M.0 interface, as it transports the largest volume of data, is measured.

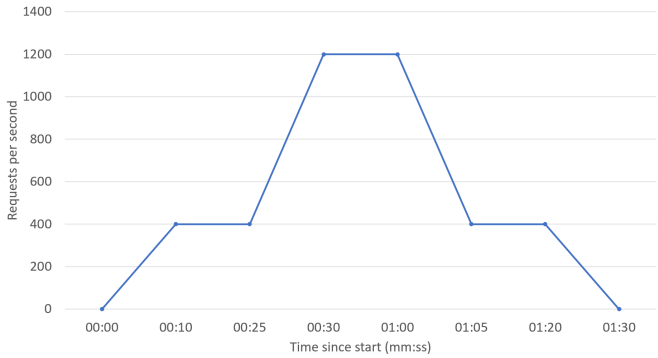

Figure 5: The JMeter load profile for both evaluation experiments.

3.1 Results

To analyze the outcomes of the experiments, multiple metrics were gathered client sided, server sided and within the MAPE-K application-level infrastructure itself. In the following, only some of the collected metrics will be discussed. Also, the following figures only show the use of one specific protocol (SNMP). A full paper that shows the performance of more protocols is planned for the beginning of 2021. Also, the use of SNMP as the communication protocol in self-adaptive MAPE-K control loops is described in detail in a contribution currently under review for the [double blind review] conference. The first experiment (without self-adaptation enabled) shows, that the single web server, which has to process all requests, is completely overwhelmed by the number of requests. Figure 7 shows the number of successful and failed transactions. After a brief period where the plot matches the load profile (figure 5), the resource demand spikes and many requests are dropped, which results in freeing server-side CPU capacity. When the maximum RPS are reached (approximately on timestamp 11:12:50), the web server cannot keep up with the resource demand and is unable to answer the requests, which results in a slowly increasing number of closed connections. Also, the per-second averaged response time (see figure 8) starts to rise. After the application of load ends (timestamp 11:13:45), there are still many requests unanswered, which the web server did already drop. This results in a continuing linear increase of the response time until the JMeter process times out and exits all of its child threads simultaneously around 11:15:15.

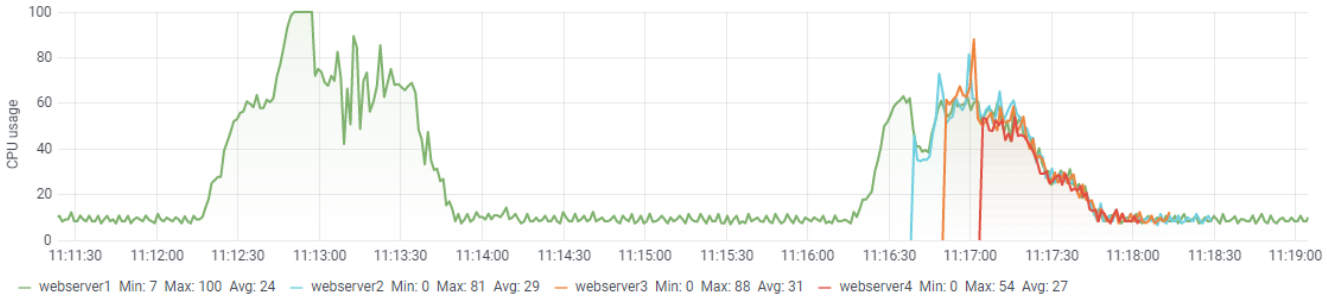

Figure 6: Plot of the CPU usage of all web servers without adaptation enabled (left) and with adaptation enabled (right).

Table 2: Comparison of distributed and centralized monitoring components behaviour without applied load and under load.

	IF M.0 (distributed)	IF M.0 (distributed) under load	IF M.0 (central)	IF M.0 (central) under load
payload message overhead	40% compared to UDP	40% compared to UDP		
number of messages exchanged	119	309		
average packaging time (ms)	96ms	140ms		
average processing time (microsec)			20microseconds	20microseconds
memory consumption	268M	268M per active machine	325M	325M
continuous processor load	9%	9% per active machine	18%+20%	18%+20%
time to add/remove server	no adaptations needed	11,6s		
increase of communication effort	no increase	incremental per added webserver		

The left side of figure 6 shows the CPU usage of the involved web servers for the same experiment. As you can see, the graph shows an increase in CPU usage as the RPS rise. As the CPU reaches 100% usage on 11:12:45, the web server cannot keep up and starts dropping connection requests, which coincides exactly with what we see on the client side figures (figures 7 and 8).

In experiment two, on the right-hand side of figure 6, you can see the same load applied to the infrastructure but with self-adaptation enabled. After the number of requests and the correlating rise of CPU usage reaches a certain threshold, an adaptation is executed, which enables the use of another web server (i.e., 11:16:40, 11:16:52, 11:17:04). As the load balancer distributes requests in a round-robin fashion, each of the web servers should process the same number of requests (i.e., 3000 RPS at max), which is what figure 6 reflects. After the load drops below a certain threshold, the number of servers is decreased (i.e., 11:18:05, 11:18:14, 11:18:30). Again, with figure 9 we can see the number of successful and failed transactions but with adaptation enabled. In experiment two, however, the plot matches the load profile (figure 5) almost exactly. That means, no requests are dropped or responses missing at all. The per second average response time (figure 10) for experiment two ranges between 2 and 22 milliseconds at max, which represents an almost non-perceptible delay. Concluding both experiments, the self-adaptation provided by the MAPE-K infrastructure succeeded in sustain availability of the website by heading off intolerable load to multiple web servers.

But the self-adaptation comes at a cost. As table 1 (for detailed descriptions of the metrics and infrastructure components, see the [double blind review] paper) shows, multiple overheads are introduced into the infrastructure by enabling self-adaptation. Firstly, each enabled web server runs an application which by itself consumes a certain amount of CPU time. With or without any load applied, the application produced a constant amount of 9% CPU usage (per activated server) on the distributed monitoring component servers and 18% (plus 20% within another core and thread) on the centralized monitoring component servers. Next, the distributed monitoring components used 268 megabytes of memory each and the centralized monitoring component 325 megabytes respectively. Furthermore, each of the distributed monitoring components sends a message containing monitoring data approximately once per second, which produces network load. Combined, in idle (i.e., without load applied to the infrastructure), 119 messages were sent during the first experiment by one active web server during a 90 seconds period and 309 messages during the second experiment respectively. Next, SNMP as a communication protocol introduces overhead as well, as it adds additional organizational information into the payload of each sent network packet. Compared to a minimal implementation that uses plain UDP to transport messages, SNMP creates 40% payload overhead. Although the payload overhead stays the same during the experiments, the time of package creation rises with an increasing load applied to the servers, however, this is irrelevant, as with increasing load more and more web servers can be provided. The central monitoring component, however, shows increasing CPU usage with an increase in active web servers. The increase is rather small but indicates a potential bottleneck (but not a realistic one) whatsoever. Furthermore, the time between detecting high load and having an additional server available is 11,6 seconds on average, which has proven to be sufficient during our tests. This time can be reduced significantly by disabling a smoothing mechanism (which was added to reduce overshoot) within the analysis component that causes the adaptation decision to be based on an average of 10 seconds of measured values rather than a single measure.

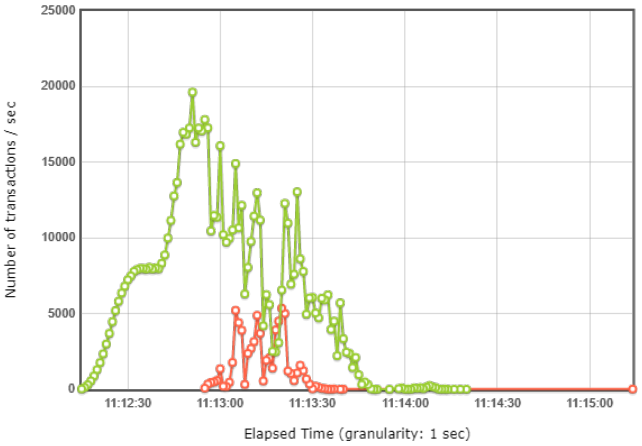

Figure 7: Plot of the successful (green) and failed (red) transactions (request and response) of all the web servers without adaptation enabled.

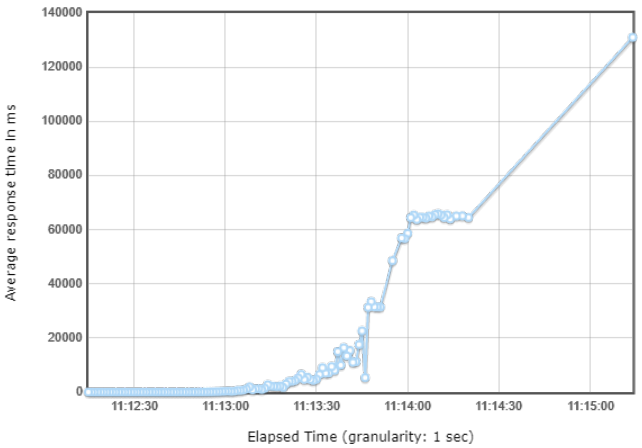

Figure 8: Plot of the response time of all the web servers without adaptation enabled.

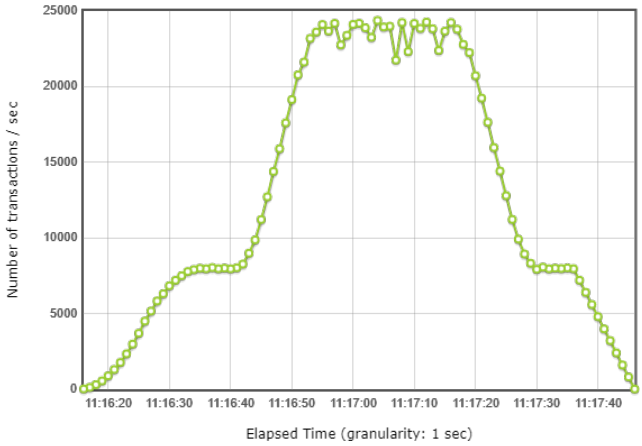

Figure 9: Plot of the successful (green) and failed (red) transactions (request and response) of all the web servers with adaptation enabled.

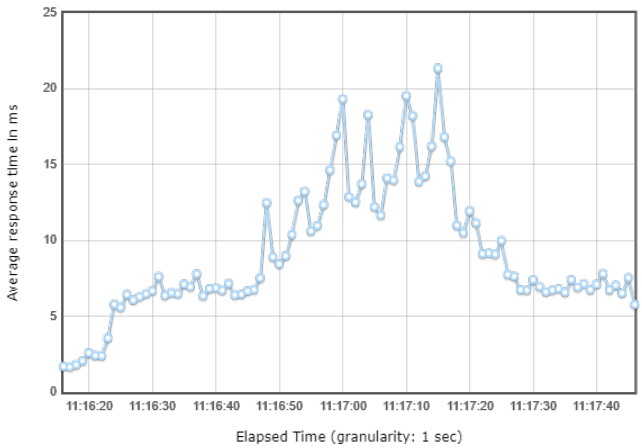

Figure 10: Plot of the response time of all the web servers with adaptation enabled.

4 DOCTORAL PROJECT PROGRESS AND UPCOMING CONTRIBUTIONS

The current plan to finish the doctoral project is divided in three sub-tasks, which depend on each other in some details. A more detailed description of the doctoral project was already published at the SEAMS 2020 conference [9]. To provide a functioning basis, the first research question (see introduction) has to be solved in the first place. The current progress on solving this task is represented by the interface descriptions, communication protocol requirements and communication protocol comparison as well as the benchmarking of them that were presented earlier (with SNMP as one representative example) in this paper. The overall idea, the description of one possible benchmarking scenario and first results of the benchmarks for the SNMP protocol were submitted to the [double blind review] conference. A contribution which presents the benchmarking results of all the introduced protocols is planned for the first half of 2021. A significant part of the final dissertation manuscript is aimed towards providing a list of best-practices for building communication in distributed self-adaptive system's control layers.

The role-concept and a basic idea of how it can help to solve research question two was presented in [9] and the [double blind review] paper. However, as the research progresses in 2021, one submission is planned for the second half of 2021 which will describe precisely how roles can be exploited to allow synchronization between MAPE-K components.

In late 2021, one submission is planned on the topic of *direct adaptation*, which is a technique that uses already present knowledge to speed up the adaptation process by skipping the computation-intensive analysis of monitored data when its possible. This aims to solve the task associated with the last research question.

During those periods, a self-written benchmarking platform will be created (a prototype was already used in the [double blind review] paper), that will be extended throughout the whole doctoral projects duration. A demo paper on it is planned for the beginning of 2022. The remaining months of 2022 will be used to finish the dissertation manuscript as well as to defend it.

ACKNOWLEDGMENTS

This work is funded by the German Research Foundation (DFG) within the Research Training Group Role-based Software Infrastructures for continuous-context-sensitive Systems (GRK 1907).

REFERENCES

[1] 1995. Shaw, M.: Beyond Objects: A Software Design Paradigm based on Process Control. ACM Software Engineering Notes 20(1), 27-38. *ACM Software Engineering Notes* 20 (02 1995). https://doi.org/10.1145/225907.225911

[2] Shang-Wen Cheng, David Garlan, and Bradley Schmerl. 2009. Evaluating the effectiveness of the rainbow self-adaptive system. In *2009 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems*. IEEE, 132–141.

[3] IBM Corporation. 2006. An architectural blueprint for autonomic computing. *IBM White Paper* 31, 2006 (2006), 1–6.

[4] Simon Dobson, Spyros Denazis, Antonio Fernández, Dominique Gaïti, Erol Gelenbe, Fabio Massacci, Paddy Nixon, Fabrice Saffre, Nikita Schmidt, and Franco Zambonelli. 2006. A survey of autonomic communications. *ACM Transactions on Autonomous and Adaptive Systems (TAAS)* 1, 2 (2006), 223–259.

[5] David Garlan, S-W Cheng, A-C Huang, Bradley Schmerl, and Peter Steenkiste. 2004. Rainbow: Architecture-based self-adaptation with reusable infrastructure. *Computer* 37, 10 (2004), 46–54.

[6] Karl M Göschka, Lorenz Froihofer, and Schahram Dustdar. 2008. What SOA can do for software dependability. *Supplementary Volume of DSN* 8 (2008).

[7] Jeffrey O Kephart and David M Chess. 2003. The vision of autonomic computing. *Computer* 36, 1 (2003), 41–50.

[8] Mazeiar Salehie and Ladan Tahvildari. 2009. Self-adaptive software: Landscape and research challenges. *ACM transactions on autonomous and adaptive systems (TAAS)* 4, 2 (2009), 14.

[9] Ilja Shmelkin. 2020. Monitoring for control in role-oriented self-adaptive systems. In *Proceedings of the IEEE/ACM 15th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*. 115–119.

[10] Pieter Vromant, Danny Weyns, Sam Malek, and Jesper Andersson. 2011. On interacting control loops in self-adaptive systems. In *Proceedings of the 6th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*. ACM, 202–207.

[11] Danny Weyns. 2017. Software engineering of self-adaptive systems: an organised tour and future challenges. *Chapter in Handbook of Software Engineering* (2017).

[12] Danny Weyns, Sam Malek, and Jesper Andersson. 2012. FORMS: Unifying reference model for formal specification of distributed self-adaptive systems. *ACM Transactions on Autonomous and Adaptive Systems (TAAS)* 7, 1 (2012), 8.

[13] Danny Weyns, Bradley Schmerl, Vincenzo Grassi, Sam Malek, Raffaela Mirandola, Christian Prehofer, Jochen Wuttke, Jesper Andersson, Holger Giese, and Karl M Göschka. 2013. On patterns for decentralized control in self-adaptive systems. In *Software Engineering for Self-Adaptive Systems II*. Springer, 76–107.