Towards Generic Adaptive Monitoring

Thomas Brand and Holger Giese
Hasso Plattner Institute at the University of Potsdam, Germany
{firstname.lastname}@hpi.uni-potsdam.de

***Abstract*—Monitoring is a key prerequisite for self-adaptive software and many other forms of operating software. Monitoring relevant lower level phenomena like the occurrences of exceptions and diagnosis data requires to carefully examine which detailed information is really necessary and feasible to monitor. Adaptive monitoring permits observing a greater variety of details with less overhead, if most of the time the MAPE-K loop can operate using only a small subset of all those details. However, engineering such an adaptive monitoring is a major engineering effort on its own that further complicates the development of self-adaptive software. The proposed approach overcomes the outlined problems by providing generic adaptive monitoring via runtime models. It reduces the effort to introduce and apply adaptive monitoring by avoiding additional development effort for controlling the monitoring adaptation. Although the generic approach is independent from the monitoring purpose, it still allows for substantial savings regarding the monitoring resource consumption as demonstrated by an example.**

I. Introduction

Monitoring is a key prerequisite for self-adaptive software and many other forms of operating software. The observations obtained through monitoring are the basis for detecting and reacting to occurring phenomena that require a reaction or make one beneficial.

However, although the monitoring of rare event-based changes such as adding or removing an architecture element at runtime can be handled quite efficiently in an incremental manner [1], monitoring properties like the number of calls or the number of messages stored in a buffer results in a substantial permanently occurring overhead for data gathering and pre-processing [2]. Consequently, this kind of monitoring for periodically detecting and reacting to phenomena requires examining carefully, which detailed information is needed and feasible to monitor.

If the question what information about the system and its external environment needs to be measured and represented [3] is only answered at development-time then the monitoring needs to cover all future requirements for phenomenon detection. In contrast, adaptive monitoring permits in each situation to monitor only the specifically required details that matter and does not consider the other ones. Consequently, the monitoring efforts can be substantially reduced, if most of the time only small subsets of the complete monitoring data matter.

However, implementing such an adaptive monitoring is a major engineering effort on its own. In the case of self-adaptive systems, the adaptive monitoring also further complicates the development of the underlying MAPE-K loop because the monitoring needs to adapt to the situation specific requirements and changing foci of attention of the analysis, plan, and execute activities.

The paper overcomes the outlined problem with the elaborated generic adaptive monitoring approach, which exploits that runtime models [4] in contrast to ad hoc data structures can be used to systematically observe accesses for monitoring results. Based on those observations our generic adaptation algorithm then adapts the active monitoring configuration to the specific current needs.

Some proposals such as [5, 6] suggest an additional adaptation mechanism, which coordinates the adaptation of the monitoring and other activities together. However, a specific design and implementation for this higher-level adaptation that adapts the underlying parts consistently has to be developed in each case. Other approaches such as [7–12] outline how a specific monitoring task for example, an adaptation strategy employed for the analysis and planning can be extended to also adjust the monitoring activity as required. However, such an extension has to be developed for each specific monitoring effort and such extensions could only be employed if only this single monitoring effort is operating exclusively. In contrast the suggested generic adaptive monitoring does work for all possible changes concerning the monitoring needs and independently of whether a single monitoring effort or multiple ones are executed concurrently.

The paper is structured as follows: In Sec. II we introduce a motivating example and the terminology. Sec. III describes the proposed approach. Characteristics of the approach are evaluated in Sec. IV also with the help of the motivating example. Sec. V reviews the related work concerning adaptive monitoring. Finally a conclusion and outlook of planned future work is presented in Sec. VI.

II. Background

In preparation to discuss the proposed approach, we introduce a motivating example and clarify the used terminology.

A. Motivating example

The potential of the proposed approach to reduce monitoring effort shall be illustrated with the help of an example. The example is inspired by a work on continuous experimentation in software product development [13]. Additionally the example is based on the SEAMS exemplar mRUBiS [14], which mimics an ebay-like auctioning software platform.

In our scenario the software manufacturer of mRUBiS provides the software as a service in the cloud for online shop tenants. Experiments are conducted on the productive

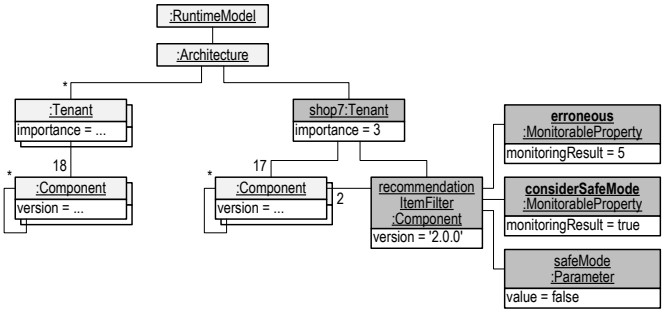

Fig. 1: Simplified mRUBiS runtime model

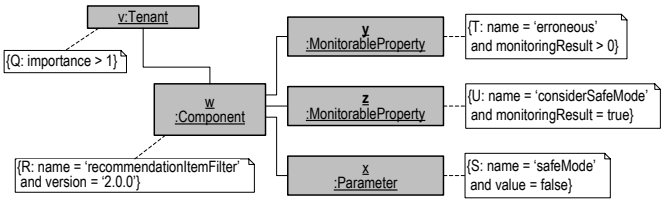

Fig. 2: Query in the form of a graph pattern to search mRUBiS shops that require the safe mode

mRUBiS system to generate additional feedback. The software manufacturer uses the resulting data to evolve the software in small increments. This is supported by very short release cycles and by independently deployable mRUBiS components.

The manufacturer now wants to gain feedback on how a new version of a software component behaves with real data and customer traffic before making it generally available. After comprehensive tests in test environments the manufacturer selects tenants who agreed to take part in experiments and deploys the new component version to their shops.

Conducting the experiment must not have a noticeable negative impact on the shops. To avoid increasing response times, the software manufacturer wants to utilize adaptive monitoring for a focused data collection. Additionally, in the case of errors caused by the new component version, the system shall quickly self-adapt to a safe mode. The decision to switch to safe mode and abort the experiment for a particular tenant shall be based on monitoring results.

B. Runtime model

In this paper we consider graph-based runtime models as the data structure to query and access monitoring results. A runtime model is defined as *an abstraction of a running system that is being manipulated at runtime for a specific purpose* [4]. Fig. 1 depicts the simplified mRUBiS runtime model. The actual model is based on the Eclipse Modeling Framework, which is a model-driven engineering technology.

The runtime model is maintained through monitoring. In order to reduce the monitoring effort our approach allows keeping only selected fragments of the runtime model up-to-date. The rest of the model that is currently not of interest then only represents an older or incomplete state of the corresponding system parts.

C. Monitorable properties

The data structure - in our case the runtime model - holds representations of *monitorable properties* (MP), through which the monitoring results can be accessed. An MP is determined by a property and a monitorable object.

A *property* can be measured such as an exception count or be examined such as the status of a component. A property also defines how the monitoring result for an MP is determined, for example, the size of the moving time window for the exception count so that the property would only hold the number of exceptions during the last hour.

The *monitorable object* determines for which entity the property shall be observed. It answers questions like for which specific component shall the exceptions be counted.

If an MP has a current *monitoring result* depends on whether the monitoring for that particular MP is active or inactive. To produce monitoring results, a *monitoring instrument* gathers pieces of data and might further process them. For example, this can comprise filtering, transforming, enriching and aggregating them. For those tasks and the presentation of the monitoring results the monitoring instrument might consist of several maybe even distributed parts.

Next characteristics of MPs are discussed that are relevant for the proposed approach. One of those characteristics is whether the MP is *inexpensive* or *expensive* regarding the production of monitoring results. This can be considered in terms of time and processing effort. Time-wise the essential question is how long it takes from triggering the monitoring activation until a monitoring result is available. This latency depends for example, on how long data needs to be gathered first. Effort-wise the required data processing is crucial as for example, complex aggregation operations can take a significant amount of time.

Another characteristic is the purpose of the MP. *Configurational* MPs are used to monitor the configuration of the system and cover parameters as well as structural aspects, which are reflected in the runtime model as nodes and edges. Their monitoring results rarely or never change. MPs that are not configurational shall be called *operational* MPs. For example, they indicate the number of exceptions that occurred.

Finally the flexibility characteristic shall be considered. For *sporadically* monitored MPs the production of monitoring results can be deactivated to avoid expensive monitoring effort. Sporadically monitored MPs are a prerequisite for adaptive monitoring and typically operational.

In comparison *continuously* monitored properties are, as the name suggests, always monitored, either because their results need to be checked constantly or because they avoid the latency caused by monitoring activation when short query response times are required. Configurational MPs are typically continuously monitored.

D. Querying monitoring results

Working with monitoring results that are accessed through a data structure involves querying them. With a runtime model this includes searching fragments of the model that match a

structural pattern. This is done by checking configurational MPs. The found fragments are match candidates. Evaluating the operational MPs of the candidates then determines the actual matches for the query result. Checking operational and configurational MPs can happen alternately.

Every match that is found for a query indicates the existence of a phenomenon upon which a reaction is intended. Fig. 2 depicts the query of our example. It is used to detect those tenants for which the experiment needs to be aborted by switching to the safe mode. The query is shown in the form of a graph pattern to make the searched structure of nodes and edges easier to comprehend. Further the constrains, which need to be fulfilled for a match, are specified. For our example a match requires that the safe mode is not active, already (S), the importance of the shop is higher than one (Q), and that the new version 2.0.0 of the recommendation item filter is deployed (R). Further, the safe mode shall only be activated if at least one request was erroneous (T) and the detailed analysis indicates that the safe mode should be considered (U).

Writing the query in expression form allows providing details on how the query shall be executed. The individual variables v to z in the Expr. 1 and 2 represent the nodes in Fig. 2 and Q to U the related constraints. P becomes true if the shown graph structure of the query and the currently investigated runtime model fragment match. For both exressions we consider an evaluation order from left to right. The two expressions differ in the use of boolean operators. Expr. 1 contains only conditional and-operators (&&), whereas Expr. 2 also contains a logical and-operator (&).

For the conditional and-operator && the so called short-circuit evaluation is performed. This means that the right operand of the operator is only evaluated if it still can impact the result. Thus whether an operand and the corresponding MPs get evaluated depends on a preceding MP. For example, consider the operand P in the Expr. 1 and 2. If P is false then the remaining part of both query expressions cannot change the result anymore and thus none of subsequent operands and MPs are evaluated due to short-circuit evaluation.

For the boolean logical and-operator & the standard evaluation is performed. This means its right operand always gets evaluated no matter if it could still change the result. Consequently, also all related MPs get evaluated, too. For example, this is the case in Expr. 2 with the logical and-operator & and the operands T and U.

$$P(v, w, x, y, z) \;\&\&\; Q(v) \;\&\&\; R(w) \;\&\&\; S(x) \;\&\&\; \underline{T(\mathbf{y}) \;\&\&\; U(\mathbf{z})} \quad (1)$$

$$P(v, w, x, y, z) \;\&\&\; Q(v) \;\&\&\; R(w) \;\&\&\; S(x) \;\&\&\; \underline{T(\mathbf{y}) \;\&\; U(\mathbf{z})} \quad (2)$$

Short-circuit evaluation is used for both expressions in their first part that covers the configurational MPs. The intention is to avoid effort on those model fragments that are not and likely will not become query matches soon due to the stable nature of the configurational MPs. In comparison, with the operational MPs that are represented by the bold variables **y** and **z** either standard or short-circuit evaluation is applied.

Because Expr. 1 contains a conditional and-operator between the operational MPs in the underlined expression part, it shall be called *query with short-circuit evaluation*. Expr. 2 is called *query with standard evaluation* because of the absence of a conditional operator in that expression part.

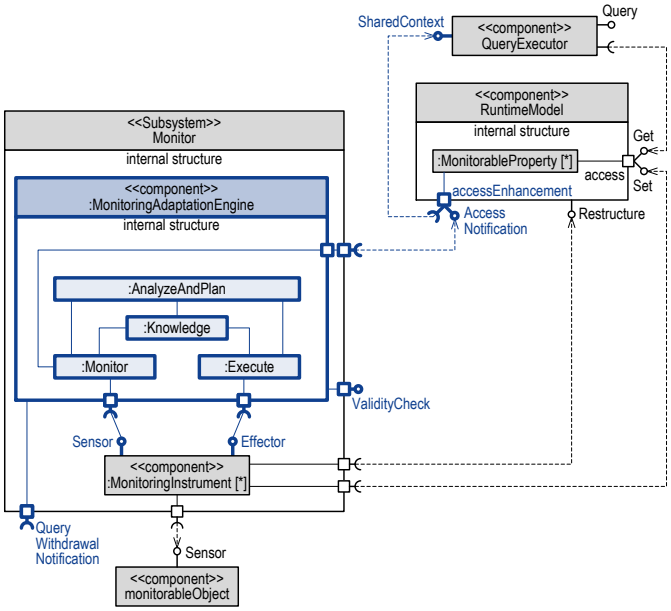

Fig. 3: Monitoring adaptation engine

III. Approach

The goal of the proposed adaptive monitoring approach is to provide those monitoring results that are currently of interest in order to avoid unnecessary monitoring effort. The monitoring results are accessible through a data structure such as a runtime model from where they can be retrieved by querying.

The required adaptation of the monitoring subsystem is driven by the *monitoring adaptation engine* (MAE), which is depicted in Fig. 3. The engine is based on a MAPE-K control loop [15]. The *Monitor* function of the loop in the MAE observes the state of the monitoring instruments as well as which monitoring results are currently of interest. The *Analyze* and *Plan* functions determine whether and when the monitoring configuration needs to be adapted. The changes are then applied by the *Execute* function. All functions of the loop act upon shared *Knowledge*.

The MAE determines for which operational MPs the monitoring needs to be active by observing if they are accessed while the runtime model is queried. As prerequisite the runtime model needs to contain representations of the operational MPs. Monitoring instruments for configurational MPs add those representations as structural elements to the runtime model using the *Restructure* interface shown in Fig. 3.

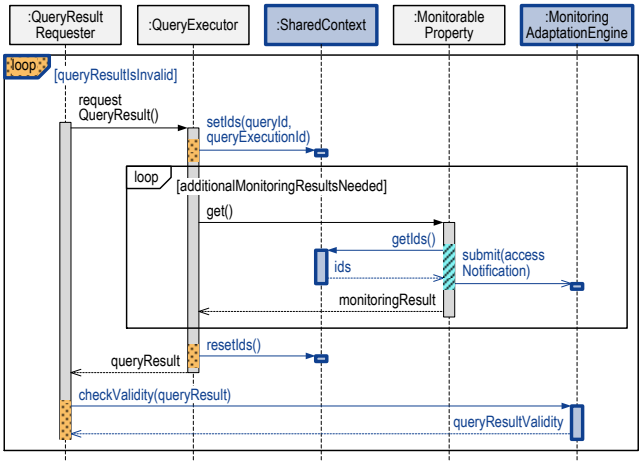

Fig. 4: Overview monitorable property get access

Fig. 4 provides more details on observing MP accesses in the runtime model. First the *QueryResultRequester* requests a result for a query like the one discussed in our example above. For this purpose the requester uses the *Query* interface of the *QueryExecutor*. The QueryExecutor stores the identifier of the query in the *SharedContext* and then starts the query execution. During the execution every MP representation in the runtime model that is accessed through the *Get* interface submits an access notification. The notification contains the MP identifier and the identifier of the executed query, which is available through the SharedContext. The notification is submitted asynchronously to a destination from where the MAE can get it.

Afterwards, the MP returns a monitoring result or a default value to the query executor. The default value is supposed to avoid errors and the abortion of the query execution due to not yet available monitoring results. Returning a default value makes the query result invalid but allows activating further required MPs also for other match candidates because the QueryExecutor continues working.

At the end of the query execution the QueryExecutor removes the query identifier from the SharedContext and returns the query result. As the query result might be falsified by default values the QuerResultRequestor needs to reconfirm the validity of the result with the MAE using the *ValidityCheck* interface. In case of an invalid result the QueryResultRequestor may trigger the query execution again.

Based on the access notifications the MAE determines for which MPs monitoring needs to be preformed. After activating the necessary monitoring instruments using their *Effector* interface depicted in Fig. 3 they produce monitoring results about the monitored objects. The instruments store those results with the corresponding MPs in the runtime model using the *Set* interface.

Through the query identifier in each access notification the MAE can keep track of which MPs are required by which query. This allows the MAE to deactivate those MPs that are no longer associated with a query in order to save monitoring effort. The *QueryWithdrawalNotification* interface can be use to inform the MAE that a particular query will not be used anymore. In our example this could happen after the experiment with the new software component ended.

Employing model-driven engineering technologies especially the Eclipse Modeling Framework for implementing the proposed approach with a runtime model is advantageous because required features are already provided or easily added.

IV. Evaluation

The motivating example introduced in Sec. II-A is now used to illustrate potential of the proposed approach to reduce monitoring effort. However first we discuss the risk of latency caused by the approach as well as why the approach is generic.

A. Genericness of the approach

The proposed approach determines which monitoring results to produce by observing which are requested while the data structure for accessing them is queried. Solely based on those observations the monitoring configuration is adapted. Therefore the executed queries influence the monitoring configuration. However it is completely transparent to the approach why which monitoring result is needed. This makes it reusable for multiple different purposes. The approach is called generic because it does not include a domain specific or single purpose adaptation algorithm.

Special requirements for adopting this generic approach are emphasized in the Figs. 3 and 4. The relevant parts are depicted with a bold blue line and have already been described in Sec. III. The dotted orange parts in Fig. 4 are related to the query execution and can be implemented through a lightweight wrapper around the original QueryExecutor, which could for example, be a plain Java algorithm or be based on languages like the VIATRA Query Language [16] or story patterns as applied in [17]. Code generation or aspect weaving could be employed to minimize the implementation effort for the hatched turquoise part related to the access notification submission. Overall, due to the reusable MAE the effort for adopting the proposed approach is comparatively low.

B. Risk of latency

The proposed approach allows saving the most monitoring effort by activating only the actually required MPs as late as possible. Which MPs to activate can depend on the monitoring results of other MPs. This causes uncertainty due to the fact that monitoring results of MPs can change over time. One way to handle this uncertainty is to activate all MPs at once that could potentially be required. This can be achieved by using the query that we defined at the end of Sec. II-D as query with standard evaluation. This query probably causes the activation of more than actually required MPs but avoids the latency caused by MP monitoring activation for subsequent query executions at least as long as no configurational MP changes. Using a query with short-circuit evaluation as defined in Sec. II-D, enables detecting phenomena with even less active MPs. The drawback is the higher risk of latency due

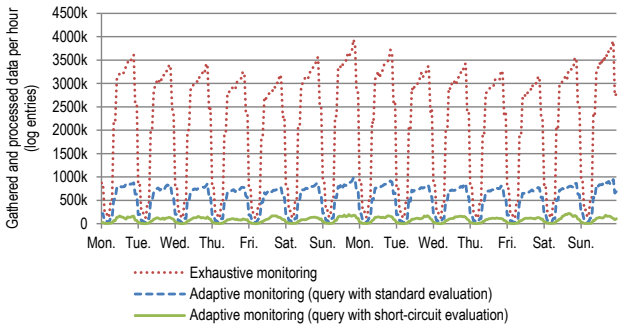

Fig. 5: Gathered and processed runtime data per hour

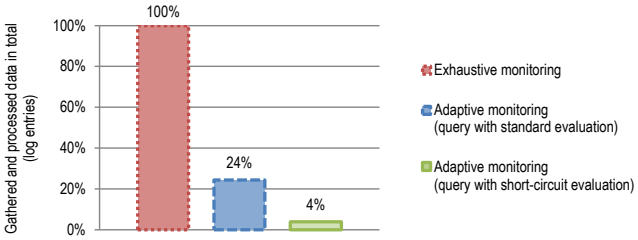

Fig. 6: Gathered and processed runtime data in total

to monitoring activation, as in this case latency can be caused by changing monitoring results of both, configurational and operational MPs as well as the stepwise MP activation through multiple query executions. In summary, the proposed approach implicates accepting the risk of latency due to monitoring activation and thus longer query response times for the opportunity to reduce monitoring effort.

C. Monitoring effort reduction

The motivating example, which we introduced in Sec. II-A, also allows comparing scenarios with adaptive and exhaustive monitoring. If monitoring exhaustively then all MPs are always active and no query execution is required to activate them.

For our example we chose the number of gathered and processed log entries as an easily and reliably determinable monitoring effort indicator. The results depicted in the Figs. 5 and 6 indicate the monitoring effort per hour over time and in total, respectively. The effort depends on the user traffic on the mRUBiS system. The simulated traffic is based on data about actual online shopping behavior [18, 19].

For the comparison of monitoring efforts only those operational MPs of the mRUBiS system are considered that are referenced in Fig. 2, namely the MPs called erroneous and considerSafeMode. For the MP named erroneous two log entries are created per user request. Additional 18 entries are created if the considerSafeMode MP is activated, too.

The results for the comparison have been generated for a mRUBiS instance with 400 tenants and for an experiment duration of two weeks. 100 tenants were selected by the software manufacturer to take part in the experiment. 90 of them had an importance value greater one. For ten out of those 90 tenants the experiment had to be aborted and the safe mode be activated due to issues with the new component version. About 36 million requests were handled in total. The average error rate was 40 per one million requests for normal and 1000 per one million for problematic deployments.

In the exhaustive monitoring scenario the above mentioned operational MPs were continuously active for all 400 tenants.

For the first adaptive monitoring scenario based on the query with standard evaluation defined in Sec. II-D the Expr. 2 was utilized. In this case both operational MPs were immediately activated for 90 tenants after the experiment started and the configurational MP constrains Q, R and S shown in Fig. 2 have been evaluated.

In the second adaptive monitoring scenario based on the query with short-circuit evaluation only the erroneous MP got activated at the beginning. Due to short-circuit evaluation the expensive considerSafeMode MP only got activated temporarily, whenever an error occurred.

As the Figs. 5 and 6 illustrate, the proposed adaptive monitoring approach allows reducing the monitoring effort significantly. In comparison the scenario based on the query without short-circuit evaluation produces and processes only 24% of the log entries that would be caused by exhaustive monitoring. The scenario based on the query with short-circuit evaluation only requires 4%.

V. Related Work

A common approach is to adapt the monitoring simultaneously with other changes to the monitored system instead of only when the data is actually requested. For example, [5] and [6] describe how a new monitoring configuration is derived from changed service level agreement specifications and gets activated with the adaptation of the monitored system.

Other approaches do not separate deciding which data is required from obtaining it by reconfiguring the monitoring, what makes those approaches specific to a certain monitoring goal such as anomaly detection. To identify the required subset of the metrics for adaptive monitoring, [7] suggests to use the metric correlation information. As due to the dependencies in a system often failures affect many metrics, the approach argues that to detect anomalous behavior a small subset of the metrics is often sufficient to detect anomalies. A monitoring rule set is employed in [8] to define, which probes are active when and where. The monitoring rules can be updated at runtime and are periodically executed to enable and disable probes as required. In [9] adaptive monitoring is used to generate a representation of a system. While the system behaves normally this approach activates only minimal monitoring to check if the system remains in the normal state. When selecting the active measurement points the algorithm also considers their cost and entropy. To reduce the overhead for monitoring, [10] suggests to use lightweight global monitoring for normal conditions and precise and localized monitoring in case problems have been detected. The heuristic employed by the approach exploits the assumption that the cause of newly detected problems is likely to come from the most recent changes in the application.

In contrast to our proposed direction the approaches discussed do not provide a generic solution to adaptive monitoring. In [5, 6] the adaptation of the MAPE-K activities including monitoring is guided by goals. However, the additional higher-level adaptation engine that adapts the MAPE-K activities based on the given goals is not generic and monitoring purpose independent but requires a substantial development effort each time the scheme is employed. Also when the monitoring is adjusted to the needs of the other MAPE-K activities, the existing proposals [7–10] require either that a specific kind of solution is employed for the other MAPE-K activity [7, 9, 10] or that the knowledge about the right adaptation steps for the monitoring steps must be explicitly specified [8]. Consequently, the existing approaches are either limited to specific solutions employed for specific monitoring purposes respectively other MAPE-K activities or require a substantial development effort. In contrast, the solution proposed in this paper is generic and independent of the purposes for which the monitoring is performed and requires only little additional development effort.

VI. Conclusion and Future Work

After introducing the required terminology in Sec. II we described in Sec. III our adaptive monitoring approach. We explained how it supports changing foci of attention for different monitoring purposes without specific development efforts for each purpose by better separating the monitoring adaptation mechanism from the rest of the system.

Finally, in Sec. IV we showed that substantial savings concerning the runtime overhead for the monitoring can be achieved in a more or less aggressive manner for the two classes of queries.

As future work we plan to also consider deactivating MPs more aggressively, to consider change events to trigger the reevaluation of queries to minimize the need to wait for activation of MPs required by the queries and to address also a combination of the approach with incremental queries as consider in [17].

References

[1] T. Vogel, S. Neumann, *et al.*, "Model-Driven Architectural Monitoring and Adaptation for Autonomic Systems," in *Proceedings of the 6th International Conference on Autonomic Computing (ICAC)*, 2009.

[2] B. H. C. Cheng, R. de Lemos, *et al.*, "Software engineering for self-adaptive systems: A research roadmap," in *Software Engineering for Self-Adaptive Systems*, B. H. C. Cheng, R. de Lemos, *et al.*, Eds., 2009.

[3] Y. Brun, R. Desmarais, *et al.*, "A design space for self-adaptive systems," in *Software Engineering for Self-Adaptive Systems II: International Seminar, Dagstuhl Castle, Germany, October 24-29, 2010 Revised Selected and Invited Papers*, R. de Lemos *et al.*, Eds., 2013.

[4] N. Bencomo, G. Blair, *et al.*, "Report on the 7th international workshop on models@run.time," *SIGSOFT Software Engineering Notes*, 2013.

[5] N. Villegas, "Context management and self-adaptivity for situation-aware smart software systems," PhD thesis, University of Victoria, 2013.

[6] G. Tamura, N. M. Villegas, *et al.*, "Improving context-awareness in self-adaptation using the dynamico reference model," in *Proceedings of the 8th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, 2013.

[7] M. A. Munawar, "Adaptive Monitoring of Complex Software Systems using Management Metrics," PhD thesis, University of Waterloo, 2009.

[8] J. Ehlers, "Self-adaptive performance monitoring for component-based software systems," PhD thesis, University of Kiel, 2012.

[9] M. Psiuk, "Goal-driven Adaptive Monitoring of Dynamic Service Oriented Systems," PhD thesis, AGH University of Science and Technology, 2013.

[10] I. Gonzalez-Herrera, J. Bourcier, *et al.*, "Scapegoat: An adaptive monitoring framework for component-based systems," in *2014 IEEE/IFIP Conference on Software Architecture (WICSA)*, 2014.

[11] D. Jeswani, M. Natu, *et al.*, "Adaptive monitoring: Application of probing to adapt passive monitoring," *Journal of Network and Systems Management*, 2015.

[12] R. Ding, H. Zhou, *et al.*, "Log2: A cost-aware logging mechanism for performance diagnosis," in *Proceedings of the 2015 USENIX Annual Technical Conference (USENIX ATC)*, 2015.

[13] A. Fabijan, P. Dmitriev, *et al.*, "The evolution of continuous experimentation in software product development: From data to a data-driven organization at scale," in *2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE)*, 2017.

[14] T. Vogel, "mRUBiS: An exemplar for model-based architectural self-healing and self-optimization," in *2018 ACM/IEEE 13th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, 2018.

[15] *An architectural blueprint for autonomic computing.* IBM Corporation, 2006.

[16] *The VIATRA Query Language*. [Online]. Available: https://www.eclipse.org/viatra/documentation/query-language.html.

[17] S. Ghahremani, H. Giese, *et al.*, "Efficient utility-driven self-healing employing adaptation rules for large dynamic architectures," in *2017 IEEE International Conference on Autonomic Computing (ICAC)*, 2017.

[18] P. Cohen, *When do us and uk shoppers shop online?* 2016. [Online]. Available: https://www.similarweb.com/blog/shopping-days.

[19] *Verteilung der Kaufvorgänge europäischer Konsumenten auf Online-Shops nach Kaufuhrzeit im Jahr 2009*, Deutsche Card Services, 2010. [Online]. Available: https://de.statista.com/statistik/daten/studie/77930/umfrage/kaufvorgaenge-auf-online-shops-nach-uhrzeit-in-europa.