From Resource Monitoring to Requirements-based Adaptation — An Integrated Approach

Holger Eichelberger
Institute of Computer Science
University of Hildesheim
Universitätsplatz 1
D-31141 Hildesheim,
Germany
eichelberger@sse.uni-hildesheim.de

Cui Qin
Institute of Computer Science
University of Hildesheim
Universitätsplatz 1
D-31141 Hildesheim,
Germany
qin@sse.uni-hildesheim.de

Klaus Schmid
Institute of Computer Science
University of Hildesheim
Universitätsplatz 1
D-31141 Hildesheim,
Germany
schmid@sse.uni-hildesheim.de

ABSTRACT

In large and complex systems there is a need to monitor resources as it is critical for system operation to ensure sufficient availability of resources and to adapt the system as needed. While there are various (resource)-monitoring solutions, these typically do not include an analysis part that takes care of analyzing violations and responding to them. In this paper we report on experiences, challenges and lessons learned in creating a solution for performing requirements-monitoring for resource constraints and using this as a basis for adaptation to optimize the resource behavior. Our approach rests on reusing two previous solutions (one for resource monitoring and one for requirements-based adaptation) that were built in our group.

Keywords

Monitoring, Resources, Quality Requirements, Adaptation, Event processing, SPASS-meter, EASy-Producer

1. INTRODUCTION

Adaptive systems require the observation of events, e.g., by monitoring the fulfillment of constraints drawn from given requirements, and the definition of adequate (re-)actions to the observations. As monitoring tools often exist as separate tools [16], an obvious question is how to integrate them with down-stream capabilities for analysis. This is particularly challenging if some tools have been previously built independently and are not special purpose tools for a specific application case. In this paper we describe our experiences in integrating existing tools in order to build a monitoring pipeline to achieve requirements-based adaptation. While we illustrate our approach for a resource-based application setting, the approach itself is generic and can can be applied to different application settings and monitoring tools.

ICPE '17 Companion, April 22-26, 2017, L'Aquila, Italy

DOI: http://dx.doi.org/10.1145/3053600.3053617

Figure 1 shows a process for monitoring-based adaptation. It covers the whole processing, starting with the definition of the data that should be monitored (source definition) through instrumentation of the code for gathering the resource information to the actual acquisition of the resulting data and its aggregation. In a distributed system, this data is acquired locally by various nodes and needs to be routed to one or more aggregating nodes, where the overall system state is determined and evaluation happens. The evaluation results can finally lead to an adaptation of the system. In this paper, we focus on integrating the pipeline stages for resource and requirements monitoring realized by different approaches, i.e., as indicated in Figure 1, subsequent adaptation is out of scope.

Resource monitoring aims at collecting information on the resource consumption of individual compute nodes. Tools like SPASS-meter [8] typically address the first steps of the monitoring pipeline as they include the instrumentation process (i.e., code modification itself). SPASS-meter was initially built for single system monitoring; hence, it does not include a routing component, which transports collected data to (distributed) aggregation and analysis. *Requirements monitoring* utilizes the monitored information to reason about constraints, e.g., expressing resource requirements given by Service Level Agreements. EASy-Producer is an environment that uses logical information to trigger artifact modifications. It was initially created to support product line implementation [6], but has since been extended to runtime adaptation [5]. If we compare this with typical requirements monitoring approaches [16], we see that those usually cover the routing, in particular if they support distributed environments. As they aim to aggregate arbitrary event sources, e.g., monitoring tools or components / services / agents reporting about changes of their runtime state, they typically do not aim at the creation of the monitoring events. Also, they do not support the actual adaptation as they are only monitoring solutions. In this paper we discuss the integration of both kinds of tools to cover the pipeline shown in Figure 1. The two tools we could reuse, namely SPASS-meter and EASy-Producer, did in theory support the whole processing pipeline (except for routing). Although the realized integration is specific for the two tools, we argue that the integration concept is generic and can be applied to other tools, e.g., an offline time series database covering the first steps of the monitoring pipeline.

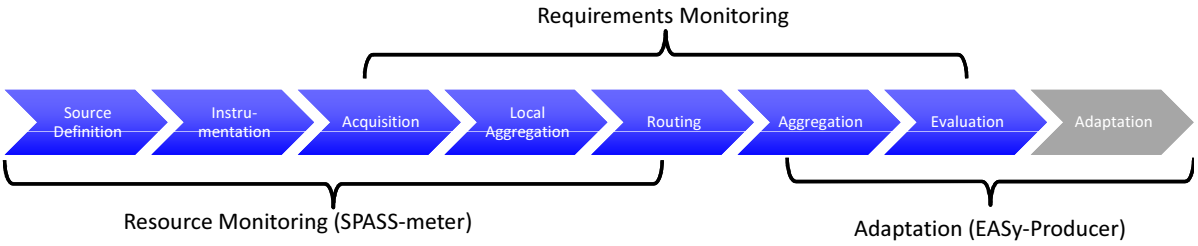

Figure 1: Monitoring process.

The contribution of this paper is two-fold: (1) We show how we integrated a resource-monitoring framework with an existing solution for constraint analysis. Here, we discuss the match among the concepts and how we bridged the mismatch. (2) We discuss the problems encountered in and lessons learned from creating this particular integration, both on conceptual and implementation level. The proposed loose integration enables reusing powerful components with little additional code and allows using the original approaches in further settings. We assume that our contributions will be helpful to others aiming at creating monitoring solutions for large systems using preexisting components.

The paper is structured as follows: The next section discusses related work, while Section 3 introduces the illustrating application scenario. Sections 4 and 5 discuss SPASS-meter and EASy-Producer, respectively. The core contribution is given in Section 6, in particular the lessons learned in Section 6.3. Finally, Section 7 provides the conclusions.

2. RELATED WORK

An important distinction for an integration approach like ours are the conceptual differences between resource and requirements monitoring tools as they provide a very different basis for integration. In this section we compare both.

Resource monitoring approaches aim at obtaining resource consumption information from the respective System Under Monitoring (SUM). While there are several approaches discussed in literature, we focus here on work which includes some support for analysis, e.g., specifying constraints and reasoning over them. Some examples are J-SEAL2 [1], SxC [11] or Kieker [4, 19]. These approaches differ in terms of their monitoring capabilities, such as which kinds of resources are considered [8], and, more related to this paper, their analysis capabilities. For example, J-SEAL2 focuses on threshold checking, SxC supports Boolean constraints and Kieker reasoning over expressions in Object Constraint Language (OCL). However, while approaches like Kieker [19] are extensible, they focus only on constraint specification and reasoning on information they can observe by themselves.

Requirements monitoring approaches typically allow gathering information from multiple (external) sources, but often do not aim themselves at obtaining raw monitoring information. In contrast to resource monitoring approaches, requirement monitoring approaches often provide more sophisticated aggregation techniques and more expressive constraint languages, typically involving some form of temporal logic. Examples are Reminds [20] and RMTL [9], which even combine first order logic and temporal constraints. A more detailed comparison of these approaches is given in [16].

In this paper, we aim at combining an existing resource monitoring approach with reasoning capabilities (in our case provided by EASy-Producer). This effectively produces a requirements monitoring approach. To our knowledge, there are currently only few approaches aiming at such an integration, e.g., the performance query language DQL as frontend for Kieker [2], but none which pursues the specific combination that we present in this paper.

3. THE QUALIMASTER CONTEXT

The case study motivating our approach is provided by the EU-Project QualiMaster[1], which aims at real-time risk analysis in financial markets. In QualiMaster, we develop a self-adaptive Big Data infrastructure (illustrated in Figure 2) analyzing the risk of market failures based on a combination of market data with data from social networks like Twitter. We study correlations among different market instruments and process social network data using sentiment analysis and related techniques. Part of the problem is to identify risks and predictive indicators in real-time data streams. The data volume can easily rise to billions of messages a day. Also, to identify financial risks, several thousands of financial products and market players must be observed simultaneously, e.g., in QualiMaster through real-time correlation computation of 5000 market players.

Big Data infrastructures handling intensive data processing rely on distributed processing environments, e.g., Apache Storm[2] can provision a certain amount of resources as a processing cluster. However, during processing, the actual resource requirements may vary as the stream characteristics of the data processing such as volume or volatility can change over time. For example, hectic financial markets can cause bursty streams leading to changes of the stream characteristics by several orders of magnitude. The typical way to handle bursty streams is to allocate additional computing resources for the current data processing. However, private processing clusters used for analyzing licensed data may have significant but limited resources. Although the elasticity of resource provisioning [17] in a cloud environment allows extending the processing cluster, minimizing processing costs is still the main goal for data streaming systems [10] creating another incentive to limit resource usage.

[1]http://qualimaster.eu
[2]http://storm.apache.org

In QualiMaster, we address this by adapting the data processing to fulfill the actual resource requirements. This involves switching among alternative algorithms [13], but also opportunistic utilization of high-performance co-processors like FPGAs (Field Programmable Gate Arrays). For this context, we created a specific integration of performance and requirements monitoring. It uses SPASS-meter, the FPGAs and Storm as distributed event sources for determining response time/processing latency, throughput, load and memory usage. As part of the requirements monitoring, we aggregate over data processing pipelines, consisting of data sources, data processing algorithms and data sinks and evaluate resource constraints over these concepts.

4. SPASS-METER

SPASS-meter has been created as a resource monitoring framework for Java programs and Android Apps [8]. Main design goals of the tool where flexibility, integrated online data aggregation, and low monitoring overhead. It supports the steps source definition, instrumentation, data acquisition and aggregation. During source definition a performance engineer defines the parts of the SUM and the forms of resource consumption that shall be monitored. This definition can either be given as code annotations or in an external XML-file, the latter allowing monitoring of legacy applications.

In terms of what to monitor, SPASS-meter allows the scope definition on various levels of granularity. The most important level is the monitoring group level. A *monitoring group* is defined by the user as of a combination of interfaces or methods. SPASS-meter determines the resources defined for a monitoring group that are consumed when the members of a group are called at runtime. In the QualiMaster case, a monitoring group represents, e.g., a correlation computation algorithm and its utilized resources. Some data can also be determined based on the whole application, the Java Virtual Machine (JVM), or the operating system.

SPASS-meter can monitor a wide range of resources. For example, CPU / execution time, memory allocation, network-I/O, file-I/O, and battery consumption. The latter can only be measured on some platforms. When defining what and where to monitor, an arbitrary combination can be specified, i.e., it is possible that one monitoring group encompasses some interfaces and observes the utilized CPU-time, while for another group of individual methods all file-I/O is recorded. All data is auto-aggregated on the level given by the source definition, i.e., if we want to monitor the file-I/O for a monitoring group than we get the data on this level of granularity, although this group may correspond to hundreds of individual measurement points.

5. EASY-PRODUCER

EASy-Producer was initially created as a tool for supporting product line engineering [6]. In product line engineering a *variability model* describes the set of possible variants (products). This information is then used in a second step (instantiation) for deriving specific products. While this instantiation typically happens during development time, it can also happen at later points in time, e.g., during installation time or runtime. The point in time when a specific variant decision from the variability model is mapped to a resolution is called the *binding time*. For the same variant, different decisions can be instantiated at different binding times, even the same decision can have different binding times for different products (meta-variability) [18].

The EASy-Producer environment, while initially focused on traditional product line engineering, was created right from the start with binding time flexibility in mind. It consists mainly of two parts: there exists a variability description language, called IVML, and an instantiation language, called VIL. IVML is used to describe the different possible variability decisions. These decisions are defined as variables of traditional data types (e.g., Boolean, Integer, enumerations, Strings, etc.), but the language also supports more complex types like lists, sets, records and even cross-references. The language also provides capabilities to define constraints among individual decisions using an OCL-inspired language. IVML is highly expressive, hence, it can even be used to model complete data-processing pipelines as shown in [7].

The underlying interpretation of the language is a non-monotonic default logic, meaning values can be given as defaults, reasoning is supported by an integrated reasoner on this, including detection of conflicts and value derivation. Values can be changed at later points in time (e.g., later binding times) until they are defined at some point as final.

The VIL language has then the purpose of mapping a set of artifacts according to a variability description to a specific instance. At development time this can be done using preprocessing or other techniques, while at runtime this is typically done by reconfiguring components. In an adaptation scenario (like the one presented here), a specific configuration is defined using IVML and this is then instantiated using VIL. As the situation changes, a new configuration is defined and then instantiated again using VIL, leading to a reconfiguration of the system.

6. INTEGRATION

The integration of general-purpose resource monitoring with a generic reconfiguration approach supporting requirements monitoring and constraints analysis is more beneficial than inventing a new combined solution from scratch. An integration allows reusing stable components with high flexibility and high expressiveness and reduces the overall realization effort. In this section, we discuss our integration concept (Section 6.1), the technical aspects of the integration (Section 6.2) and the lessons that we learned (Section 6.3).

6.1 Approach

We discuss now our approach to integrate resource monitoring and constraint analysis / reconfiguration. First, we introduce the overall approach and refine it then in the context of our scenario. Next, we discuss how monitored information can be taken over into the reconfiguration as well as an example from the application domain. Finally, we elaborate on potential conceptual problems.

In our approach, SPASS-meter provides a quantified representation of the system state, which shall be used by EASy-Producer to reason about resource constraints and to detect violations of resource constraints. Essentially, the two frameworks are generic while the reconfiguration model (IVML + VIL) is domain-/application-specific. As both frameworks are independent components, the integration should be as loose as possible to enable further reuse. The core part of the integration is to map the system state into the runtime variables (decision variables marked for runtime binding) of

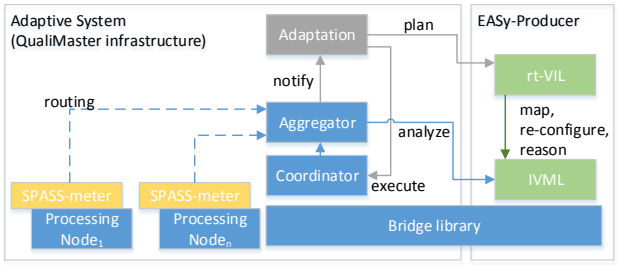

Figure 2: Integration overview.

the reconfiguration model and to perform reasoning through EASy-Producer. In the most simple case, the monitoring groups of SPASS-meter correspond directly to IVML variables. We will discuss now a more complex mapping.

In QualiMaster, we aim at monitoring a distributed system, i.e., multiple instances of SPASS-meter are running in parallel on different nodes. SPASS-meter already performs a local aggregation, e.g., in terms of the memory consumption of a monitoring group. This monitoring information must be collected, aggregated and consolidated into a common representation of the (cluster) system state, which is maintained in our case by a central node. For this purpose, the nodes perform simple routing, i.e., they pass their information in a regular fashion via network to the central aggregation component (Aggregator in Figure 2). This aggregation is domain-/application-specific, as it involves aggregation to concepts such as data processing pipelines represented in the reconfiguration model, i.e., a specific mapping is required. The Aggregator receives also information provided by other sources, e.g., the stream processing framework or the FPGAs. Further, the Aggregator calculates derived measures, e.g., throughput or latency for pipelines based on their configured structure. For constraint analysis, the Aggregator passes the consolidated information to EASy-Producer, which maps the received information onto the reconfiguration model and performs the reasoning.

In the reconfiguration model, we represent domain concepts such as pipelines as records with respective slots for the individual resource consumptions, typically as real values. Examples are memory consumption, execution time per data item, throughput per second, etc. The mapping relates the system state to the respective IVML variables and changes the values accordingly. This can be achieved 1) one-by-one, i.e., for a system state observation determine the corresponding IVML variable and change the value, or 2) on-demand, i.e., transparently return the actual monitoring values upon request, e.g., by the reasoner. We follow the first alternative as we discuss in more detail in Section 6.2.

To illustrate the approach, we describe now the flow of monitored data until constraint analysis. Imagine a data processing algorithm is allocating some memory. SPASS-meter records the consumption in the monitoring group defined in the source definition. Periodically, SPASS-meter routs the (aggregated) monitoring values as an event to the Aggregator, which performs the consolidation to domain concepts. First, the Aggregator assigns the received memory consumption to the respective algorithm, which implements the data processing. As the algorithm is executed by a processing node, the Aggregator changes also the memory consumption of the processing node and, in turn, of the containing pipeline. This can lead to updates of derived values, e.g., minimum, maximum or average memory consumption in the involved system state concepts. Periodically, the Aggregator freezes the system state, maps the system state into the reconfiguration model and executes the IVML reasoner, which identifies the failing constraints.

As we rely on a loose integration of two independent frameworks, the mapping between system state and reconfiguration framework is central to our approach. Depending on the monitored system, this mapping can be rather simple, e.g., an identifier-value mapping if just a few top-level resource observations are involved. In the QualiMaster case, the mapping involves complex domain-specific structures, which may require a certain effort for realizing the mapping and affecting the efficiency of the approach at runtime.

6.2 Technical Aspects

We aim at a loose integration of monitoring with the reconfiguration framework. This also applies for the integration with the overall system, i.e., the integration shall call/extend both frameworks only at a few points. In this section, we discuss the technical aspects of the integration, the issues that occurred with SPASS-meter and EASy-Producer and the decisions that we made to resolve the issues as well as remaining technical problems.

For the integration, SPASS-meter provides an observer API to notify interested parties upon changes in the monitoring groups. We realized a specific observer, which turns monitored values into events and routs them to the Aggregator component. EASy-Producer must be called to load the reconfiguration model, to perform reasoning/constraint analysis and to determine the adaptive decisions. We discuss now two major technical issues that we faced during the realization of a prototype for our case study, in particular the application to distributed monitoring and the realization of the specific mapping used in our case study.

One technical problem arises from monitoring a distributed stream processing system using a generic resource monitoring approach, which already performs data aggregation. As introduced above, the lowest aggregation level of SPASS-meter is the monitoring group, which corresponds in object-oriented systems to the class level, i.e., it aggregates over all instances of a class. However, in our case, the most fine grained domain concept is a task representing a potentially parallelized instance of a data analysis algorithm. As Storm realizes tasks as instances of classes, aggregating on class level by SPASS-meter hides important information and prevents reasoning over resource constraints for tasks. In turn, this prevents related adaptations, e.g., as supported by our prototype, changing the parallelization degree at runtime, ranging from sequential execution to execution on a certain node. Moreover, we must relate the local instances running on a node to application-specific cluster-unique task identifiers required by the Aggregator. As resolution, we extended SPASS-meter with a further, optional aggregation layer on instance-level. These instance-level monitoring groups are identified by a locally unique identifier. The integration of the QualiMaster system with SPASS-meter, a SPASS-meter observer plugin, defines the relation among local identifiers and task identifiers so that monitoring events based on task identifiers can be routed and aggregated.

Another issue is the realization of the mapping between

SPASS-meter and EASY-Producer. Basically, we opted for a one-by-one mapping as the second approach discussed above would require significant changes in EASy-Producer. A simple application-specific mapping can be defined in VIL without additional code as long as data types provided by EASy-Producer are used. However, performance experiments with an initial prototype indicated that for complex systems such as QualiMaster a mapping realized in source code could be more efficient. Due to this insight, we realized a bridge library between the QualiMaster system and EASy-Producer as shown in Figure 2. The library implements the mapping between monitoring and reconfiguration and refines the constraint analysis by efficient domain-specific functionality. The domain-specific functionality also allows simplifying IVML resource constraints, e.g., to omit relevancy conditions such as a pipeline must be running to cause resource violations. Using the bridge library we improved the execution time of the mapping by 87% and of the constraint analysis by 89% [15]. However, the library also requires dependencies to all involved components leading to an increase of build and testing complexity.

So far, we rely on a central Aggregator, which is connected by an event bus to the processing nodes and other QualiMaster components. By default, the processing nodes send four times per second monitoring events, which contain multiple observed values as a batch. Experiments show, that this is sufficient for operating multiple pipelines with around 50 nodes in parallel. Moreover, initial scalability experiments indicate that we can reason on constraint violations each second for up to 500 nodes. We believe that higher scalability can be achieved by a hierarchy of Aggregators and an appropriate routing component.

6.3 Lessons Learned

We integrated two independent frameworks to leverage resource monitoring to constraint-based requirements monitoring. In this section, we discuss the lessons that we learned by applying our approach in practice.

Was it a good idea to pursue such an integration? Both frameworks provide rich functionality, which can be reused out-of-the-box: SPASS-meter is a flexible low-overhead resource monitoring framework. EASy-Producer is a powerful adaptation tool offering rich configuration and instantiation concepts as well as a highly expressive constraint language. An integration allows reusing the respective functionality without facing the effort of re-implementing it in a specific system. Moreover, both frameworks can still be utilized in a standalone manner, e.g., EASy-Producer for configuring and instantiating software product lines. As discussed above, an integration can be achieved through calling/extending existing functionality and by defining a mapping between resource monitoring, aggregation, and reconfiguration. On the one hand, reusing generic components also comes at a certain price, e.g., a complex or domain-specific system state representation. On the other hand, application-specific functionality can help simplifying the reconfiguration model and optimizing the runtime performance. Although the performance of our approach fulfills our needs, further optimizations may be needed to support situations with scalability demands, e.g., an even faster approach for passing data or a tighter integration between the components.

While realizing the integration, we learned about the following benefits and drawbacks of the proposed approach. An integration of existing powerful approaches allows reusing their capabilities with little additional code instead of facing the effort of building a complete custom solution. Moreover, a loose integration as proposed in this paper enables us to utilize the approaches further on as individual tools. However, one drawback is the design of the bridge library, which currently implies significantly higher testing and building efforts. Another issue is the need for dealing with generic functionality, which can be migrated to SPASS-meter or EASy-Producer in order to simplify the creation of the bridge library, the mapping or the reconfiguration model. We believe that the gained knowledge will help us in realizing further monitoring systems through extensive reuse.

Can we generalize the loose integration approach to other frameworks, e.g., monitoring frameworks such as Kieker [4, 19] or other constraint analysis components? In principle, we believe that this is possible if the components provide access to the collected data. SPASS-meter allows defining observers on the collected data that we use to perform the routing of collected data to the Aggregator. A similar approach is likely possible for Kieker through plug-in extensions [19]. An alternative could be using a common exchange format such as Open.Xtrace [12] enabling a more generic integration. Also other kinds of data sources such as offline monitoring databases or DQL could be used instead. Similarly, another constraint analysis tool could be used, e.g., Drools[3] or Dresden OCL [3] if an API is provided to load the respective constraint model, to map the state representation onto the model, to perform reasoning and to obtain detailed results such as failing constraint clauses. However, the actual integration effort and the runtime performance heavily depends on the used components. For example, experiments with Drools showed that a significant development and mapping effort is needed. Moreover, a model-specific reasoner such as the one provided by EASy-Producer can be significantly faster. Experiments show that we improved the reasoning time for a IVML model with 1000 variables from 2s for a Drools-based implementation to around 330ms using the EASy-Producer reasoner [14]. In the mean time, the EASy-Producer reasoner is able to process the QualiMaster reconfiguration model with more than 4000 variables in less than 250ms [7, 15].

7. CONCLUSIONS

Operation and self-adaptation of large-scale systems require a combination of low-level monitoring approaches and higher-level analysis approaches. While low-level monitoring provides basic runtime information, higher-level analysis such as requirements monitoring enable aggregation, reasoning, and decision-making. In this paper, we discussed the integration of a flexible resource monitoring approach (SPASS-meter) with a reconfiguration and adaptation framework (EASy-Producer) in the context of adaptive Big Data stream processing.

Our approach relies on the loose integration of existing frameworks through an application-specific mapping of monitored information onto a reconfiguration model to enable requirements-based reasoning and analysis over constraints. This approach allows reusing the functionality provided by two frameworks, which still can be applied independently. While such a mapping can be developed by the generic func-

[3]https://www.drools.org/

tionality provided by the integrated frameworks, it can also be realized in terms of source code. Such a programmed mapping based on existing generic components can lead to significant performance improvements as well as a simplification of the model constraints. The mapping forms the heart of a bridge library, which links the involved systems, but, in our case, increases the testing and building effort. We believe that the approach can be transferred to realize similar loose integrations for other monitoring, reasoning or analysis enabling reuse of these frameworks and easing the development of monitoring solutions for large-scale and complex software systems.

So far, we focused on internal events, in particular resource consumption. In the future, we aim at developing a more general requirements monitoring tool based on the integration and reuse approach discussed in this paper. As part of this work, we aim at identifying/migrating more generic and reusable concepts for the involved frameworks in order to ease their integration.

8. ACKNOWLEDGMENTS

We thank Sascha El-Sharkawy, Roman Sizonenko, and Aike Sass for their work on EASy-Producer and SPASSmeter. This work was partially supported by the QualiMaster (grant 619525) funded by the European Commission in the 7th framework programme. Any opinions expressed herein are solely by the authors and not of the EU.

References

[1] W. Binder, J. G. Hulaas, and A. Villazón. Portable resource control in Java. In *Conference on Object-oriented Programming, Systems, Languages, and Applications (OOPSLA '01)*, pages 139–155. ACM, 2001.

[2] M. Blohm, M. Pahlberg, S. Vogel, J. Walter, and D. Okanovic. Kieker4DQL: Declarative performance measurement. In *Symposium on Software Performance (SSP '16)*, 2016.

[3] B. Demuth and C. Wilke. Model and object verification by using Dresden OCL. In *Russian-German Workshop Innovation Information Technologies: Theory and Practice*, page 81, 2009.

[4] J. Ehlers and W. Hasselbring. A self-adaptive monitoring framework for component-based software systems. In *European Conference on Software Architecture (ECSA'11)*, pages 278–286, 2011.

[5] H. Eichelberger. It's about the mix: Integration of compile and runtime variability. In *FAS*W Workshop on Dynamic Software Product Lines (DSPL '16)*, 2016.

[6] H. Eichelberger, S. El-Sharkawy, C. Kröher, and K. Schmid. EASy-Producer: Product Line Development for Variant-rich Ecosystems. In *Software Product Line Conference (SPLC '14), Vol 2*, pages 133–137, 2014.

[7] H. Eichelberger, C. Qin, R. Sizonenko, and K. Schmid. Using IVML to Model the Topology of Big Data Processing Pipelines. In *Software Product Line Conference (SPLC '16)*, pages 204–208, 2016.

[8] H. Eichelberger and K. Schmid. Flexible resource monitoring of Java programs. *Journal of Systems and Software*, 93:163 – 186, 2014.

[9] H. Gunadi and A. Tiu. Efficient runtime monitoring with metric temporal logic: A case study in the android operating system. In *Symposium on Formal Methods (FM '14)*, pages 296–311, 2014.

[10] T. Heinze, P. Meyer, Z. Jerzak, and C. Fetzer. Measuring and estimating monetary cost for cloud-based data stream processing. In *Conference on Distributed Event-based Systems (DEBS '13)*, pages 333–334, 2013.

[11] F. Martinelli, P. Mori, T. Quillinan, and C. Schaefer. A runtime monitoring environment for mobile Java. In *Conference on Software Testing Verification and Validation Workshop*, pages 270–278, April 2008.

[12] D. Okanović, A. van Hoorn, C. Heger, A. Wert, and S. Siegl. Towards performance tooling interoperability: An open format for representing execution traces. In *European Performance Engineering Workshop (EPEW '16)*, pages 94–108, 2016.

[13] C. Qin and H. Eichelberger. Impact-minimizing runtime switching of distributed stream processing algorithms. In *Big Data Processing - Reloaded Workshop of the EDBT/ICDT Joint Conference*, 2016.

[14] QualiMaster consortium. Quality-aware processing pipeline modeling, 2014. Deliverable D4.1, http://qualimaster.eu.

[15] QualiMaster consortium. Quality-aware processing pipeline modelling and adaptation, 2016. Deliverable D4.4, http://qualimaster.eu (to appear).

[16] R. Rabiser, S. Guinea, M. Vierhauser, L. Baresi, and P. Grünbacher. A comparison framework for runtime monitoring approaches. *Journal of Systems and Software*, 125:309 – 321, 2017.

[17] B. Satzger, W. Hummer, P. Leitner, and S. Dustdar. Esc: Towards an Elastic Stream Computing Platform for the Cloud. In *Conference on Cloud Computing (CLOUD '11)*, pages 348–355, 2011.

[18] K. Schmid and H. Eichelberger. Model-based implementation of meta-variability constructs: A case study using aspects. In *Variability Modeling of Software-intensive Systems (VAMOS'08)*, pages 63–71, 2008.

[19] A. van Hoorn, J. Waller, and W. Hasselbring. Kieker: A framework for application performance monitoring and dynamic software analysis. In *Conference on Performance Engineering (ICPE '12)*, pages 247–248, 2012.

[20] M. Vierhauser, R. Rabiser, P. Grunbacher, K. Seyerlehner, S. Wallner, and H. Zeisel. Reminds : A flexible runtime monitoring framework for systems of systems. *Journal of Systems and Software*, 112:123 – 136, 2016.