# Model-driven Instrumentation with Kieker and Palladio to forecast Dynamic Applications

Reiner Jung[1], Robert Heinrich[2], Eric Schmieders[3]

[1]Software Engineering Group, Kiel University
[2]Software Design and Quality, Karlsruhe Institut of Technology
[3]Software Systems Engineering, University of Duisburg-Essen

**Abstract:** Providing applications in stipulated qualities is a challenging task in today's cloud environments. The dynamic nature of the cloud requires special runtime models that reflect changes in the application structure and their deployment. These runtime models are used to forecast the application performance in order to carry out mitigative actions proactively. Current runtime models do not evolve with the application structure and quickly become outdated. Further, they do not support the derivation of probing information that is required to gather the data for evolving the runtime model. In this paper, we present the initial results of our research on a forecasting approach that combines Kieker and Palladio in order to forecast the application performance based on a dynamic runtime model. To be specific, we present two instrumentation languages to specify Kieker monitoring probes based on structural information of the application specified in Palladio component models. Moreover, we sketch a concept to forward the monitored data to our PCM-based runtime model. This will empower Palladio to carry out performance forecasts of applications deployed in dynamic environments, which is to be tackled in future research steps.

## 1 Introduction

Cloud-based applications evolve continuously during their runtime as integrated services are updated or their deployment context changes. For instance, cloud environments allow to migrate software components online from one infrastructure to another (cf. [WLK11]). Moreover, the functionality of used third-party services may evolve during runtime, e.g., by integrating additional services or due to version updates. As the application functionality is offered with certain qualities being influenced by these changes, the quality assurance of cloud applications within their dynamic context is challenging. The concept of runtime models has been established as an effective technique in order to reflect current applications states and to forecast their quality attributes such as performance [BBF09]. This enables the proactive identification of upcoming performance bottlenecks and allows to perform mitigative actions in time to maintain a certain application quality.

Owing to frequent changes, rigid prediction models proposed in the related work, e.g., [GT09] and [LWK+10], quickly become outdated and, hence, do not provide the appro-

priate basis for performing forecasting during runtime. The need for runtime models that represent actual characteristics of the application, such as the application structure and deployment information, seems to be indicated. Current instrumentation approaches for creating runtime models, such as Kieker [vHWH12], generate and maintain dynamic runtime models on the basis of monitored method invocation traces. The key characteristic of such runtime models in reflecting the current application status offer a promising basis for the forecast of application qualities in dynamic environments.

Kieker provides powerful tool-support for maintaining the synchronization between the runtime model and the actual application characteristics. However, Kieker's monitoring purpose excludes the forecast of application qualities, such as performance, which is required for carrying out mitigative actions proactively. In order to gain the ability of forecasting application attributes, we combine the monitoring capabilities of Kieker with the forecasting capabilities of Palladio (see [BKR09]) in one single runtime model. With this combination, we aim for covering two interrelated aspects in the quality assurance of dynamic cloud applications. First, this will enable the forecast of the application performance on basis of an up-to-date representation of cloud applications. Second, the combination will furnish the verification of structural changes, e.g., meant to mitigate forecasted performance bottlenecks, against data-geolocation policies (cf. [HHJ+13]).

However, two problems arise when combining Kieker and Palladio to achieve the aforementioned goal. The locations where to inject probes in the application for collecting method traces have to be specified, as this is required to generate the dynamic runtime model. One available source for structural information of the application is the Palladio component model (PCM) that may be exploited as reference for the probing information. Thus, the first problem is how to specify probing information based on PCMs. The second problem is that monitoring results (i.e., the runtime information) must be forwarded to the PCM. Thus, the second problem is how to enrich the PCM with the collected monitoring data.

In this paper, we present the results of an initial step towards the integration of both tools. We introduce two instrumentation languages to address the first problem allowing to generate probes in our next research steps. The instrumentation aspect language (IAL) allows to specify where monitoring probes are injected in the code based on structural application information codified in the PCM. It also specifies which application attributes have to be monitored. To this aim, the instrumentation record language (IRL) describes data structures to represent the monitored information. In order to address the second problem, we propose an approach to augment conventional PCMs with monitoring information to furnish a PCM-based runtime model. Furthermore, application usage profiles are generated based on monitoring information that are subsequently used to carry out accurate forecasts. This closes one gap in MDSE and makes Palladio models beneficial during runtime, e.g. for runtime adaptation of the observed cloud application.

The remainder of the paper is structured as follows. Section 2 describes an application scenario based on CoCoME [RRMP11]. Section 3 gives an overview of the proposed approach, which is detailed in Section 3.1 and Section 3.2. Section 4 discussed related work before the paper is concluded in Section 5.

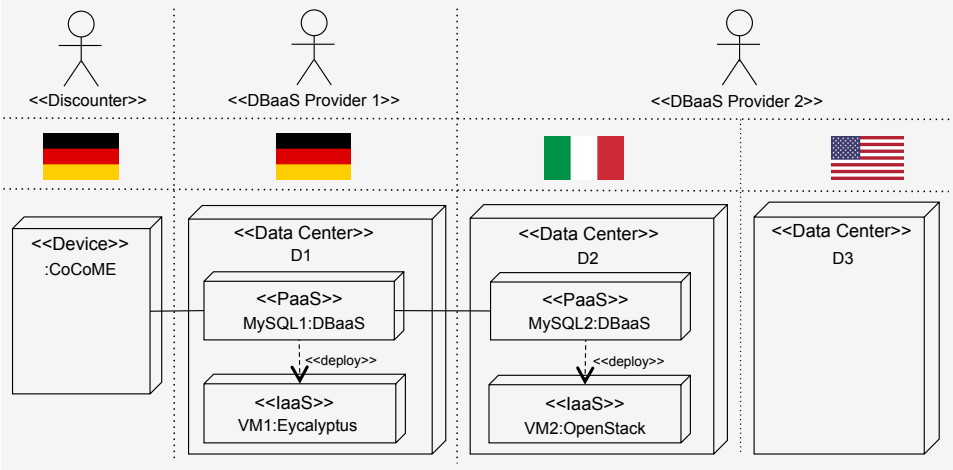


Figure 1: Example iObserve Adaptation Scenario for observation of data-migration processes in the CoCoME case study in which some German data policy is violated.

## 2 Example Adaptation Scenario: Observation of data-migration processes in the CoCoME case study

In our motivating scenario, an international supermarket chain, i.e., stakeholder 'Discounter' (see Fig. 1), runs a trading application, which is structurally based on CoCoME [RRMP11]. The discounter uses the trading application for management and controlling purposes. Among other tasks, the application gathers shopping transactions of customers to offer payback discounts. The business logic of the trading application is hosted on-premise, being located in Germany. For scalability reasons the persistency layer is outsourced to the cloud. The application uses a DBaaS solution provided by 'DBaaS Provider 1'. For the sake of simplicity, we assume in this scenario that the provider owns the underlying infrastructure. Anyway, the infrastructure could be owned by a third party as well. The used DBaaS is a MySQL instance named 'MySQL1' offered as a service that is executed on an Eycalyptus virtual machine 'VM1'. The data center hosting VM1 is located in Germany. In order to offer apparently unlimited storage capacities (cf. cloud characteristics described by the NIST standard [MG11]) the database is connected to storage resources provided by stakeholder 'DBaaS Provider 2' that operates data centers worldwide. The database containing personally identifiable information (PII), i.e., purchasing and credit card information of EU customers, is shared among both databases and is hosted at different data centers 'D1' and 'D2'. Due to the EU data protection directive (see http://eur-lex.europa.eu/LexUriServ/LexUriServ.do?uri=CELEX:31995L0046:en:HTML) the PII is not allowed to cross the EU borders, which is stipulated in data-geolocation policies.

In the following, we sketch a scenario based on this architecture that leads to a violation of the data-geolocation policies. The 'Discounter' runs a discount campaign during christmas. An unexpected large amount of users respond to this campaign, which leads to a huge amount of transactions collected by the trading application. In order to assure a high

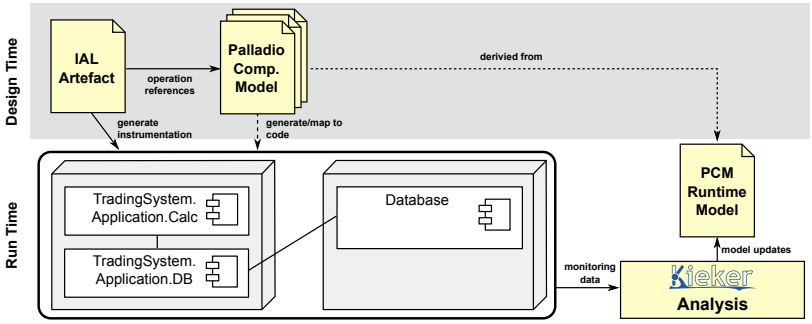


Figure 2: Kieker and Palladio Coupling

performance level the 'DBaaS Provider 2' continuously forecasts performance to identify bottlenecks that his databases may face in the near future. The forecasting algorithms indicate an upcoming violation of performance SLAs. In order to mitigate this upcoming incident the provider plans to automatically migrate the 'MySQL2' database to one of his other data centers having sufficient unused capacities during the problematic time frame to fulfill the performance SLAs. The migration of VMs during their runtime is called VM live migration and has perceived large attention recently (see, for instance, [WLK11]. Target of the live migration of VM2 (hosting MySQL2) is data center D3 being located in the USA. The 'DBaaS Provider 2' actually migrates the VM2. With this migration, PII exceeds the boundaries of the EU. While meeting the performance SLAs towards 'DBaaS Povider 1', this migration violates the data-geolocation policies attached to the PII.

## 3 Generation of the Dynamic Runtime Model

The proposed forecasting approach includes a model-driven monitoring and performance forecasting based on Kieker and Palladio. In order to carry out forecasts during runtime, several artifacts have to be created during design time (cf. Figure 2).

During design time, the PCM specifying the application has to be created. The model code relationship can either be achieved programmatically or with code generation. The PCM is annotated with information on the instrumentation probe placement specified in the IAL (see subsection 3.1). A code generator, to be developed in our future research, creates probes and weaving information out of the artifact supporting different AOP-technologies used on code level. The resulting application is then deployed and executed.

At runtime, the monitoring probes collect data and feed them into a Kieker analysis component that interprets the monitoring data and determines response times, call traces, and workload profiles. Based on those, the PCM runtime model is updated and fed to a Palladio simulator to forecast the application performance, as described in subsection 3.2.

### 3.1 Instrumentation Aspect

Instrumentation is a cross cutting concern in software systems and can therefore be expressed with aspect-oriented techniques, such as aspect-oriented modeling (AOM). Figure 3 illustrates the relationship of the instrumentation aspect language (IAL) and the instrumentation record language (IRL) to the Kieker monitoring artifacts.

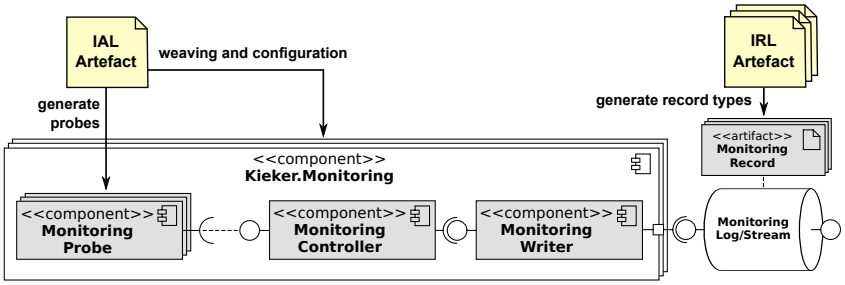


Figure 3: Overview of our model-based instrumentation approach

The IAL allows to express join points for the instrumentation aspect in a Palladio model through queries over that model. Furthermore, it allows to define the advice by specifying the record structure used for a probe and the information sources for the monitoring record. This information is then used to generate probes and additional weaving information. The record structure itself is described in artifacts written in the IRL, which provides a simple record model based on a programming language independent type system (cf. IDL of CORBA [Gro06]). The code generation process of the IAL and IRL depend on the target language, meaning the programming language used to implement the system, the different technologies to realize aspect integration, and the way Palladio model elements are represented in the target language, for example, the package structure and naming scheme of identifiers. The target language and the technologies to realize the aspect are considered together, as they both influence how code in general is generated. The naming scheme is a separate issue, as it can vary even if the same language and technology are used.

The first issue can be resolved by adding information on the used technology and thereby the programming language to the model of the software system. As Palladio is not bound to a specific technology and does not allow to specify technology choices in its model, we have two options to integrate the technology information into the model. First, we can extend Palladio to include technology information, which has the downside that modifications to Palladio effect all tools created for Palladio, making it hard to introduce this feature into the Palladio community. Furthermore, it is against the design principle of Palladio to be technology independent. Second, we can extend the aspect language to carry such information. However, the specification of the technology is not really part of the model level of the aspect. Therefore, we decided to describe the technology information through annotations based on the semantics package of the Kieler framework [vHFS11]. This allows to extend the annotations if necessary in order to cover different technologies and further properties. Moreover, it makes the distinction between aspect and technology information clearer to the user of the language.

The second problem is the mapping of references from model to code. As we cannot control the realization of modeled components in code, we cannot write a simple resolving algorithm covering this area. Instead, we provide an interface for a reference resolution method that can be introduced on project basis.

### 3.1.1 Instrumentation Aspect Language (IAL)

The instrumentation aspect language is based on the Xtext (see http://www.xtext.org) language framework. The IAL itself comprises a path like expression language to refer to Palladio model elements, and expressions to define the advice.

```
package kieker.monitoring.probe

@communication SOA-J2EE
aspect TradingSystem.**.Application[#name='Store1'].Calc.getCart(in Cart) {
    before BeforeOperationEvent(time,id,$name)
    after AfterOperationEvent(time,id,$name)
}
```

Listing 1: Model of a monitoring aspect formulated in the insturmentation aspect language

Listing 1 shows a small IAL example realizing the instrumentation of the software system described in our scenario. The example instruments components which belong to Store1 or the discounter. The first line in the listing specifies the package structure the aspect belongs to. The annotation in line three expresses the technology used to realize the communication for the specified method. The join point expression is a path query, navigating the Palladio model. In this case it refers to all methods in our scenario TradingSystem of a Calc component inside an Application component that is named Store1.

The path syntax is also used for the advice to define the parameters of the monitoring records. time or id are framework functions and are expressed by their name. The third parameter in the example is $name. While design time and run time properties of components are prefixed with a #, properties provided by the reflection API are prefixed with $. Therefore, the $name property refers to the name of the context element, which is the method getCard in the example. The language supports other reflection API calls and built in functions that will be published on the Kieker development website.

To complement the language and its auto-generated editor, the tooling includes code generation. Our generator consists of one main generator component, providing aspect reference resolution by querying the Palladio model and collecting the resulting model nodes. The single model nodes are expanded to full qualified paths and then passed to a technology specific code generator, which is selected based on the annotation information.

The technology specific generator covers code generation for the probes and its configuration files. To be able to create correct references in these configuration files, it requires a project specific mapping function, which is introduced on load time into the generator component using the inject feature of Xtend (see http://www.xtend.org) that we use to implement the generators. Based on this information, the technology specific generator produces the probe and configuration informations for the aspect integration.

### 3.1.2 Instrumentation Record Language (IRL)

The instrumentation record language (IRL) allows to declare data structures in a programming language independent way. The present incarnation of the IRL supports properties and constants of primitive data types. The IRL is equipped with a flat type system excluding recursive types. This keeps the complexity of code generators and type mappings at a minimum. Furthermore, it hinders the creation of complex data structures that would cause an increase in monitoring costs.

The type system implemented in the IRL provides a small set of primitive types (such as int, string, boolean, and arrays or vectors) that can be represented in any programming languages. Furthermore, the IRL has sub-typing capabilities, which allows to specialize record types and minimizes the specification effort. For example, the IRL code snippet in Listing 3.1.2 realizes three record types used in the Kieker event-based trace monitoring concept and a geolocation record.

```
package kieker.common.record.flow.trace.operation

struct AbstractOperationEvent {
    const string NO_SIGNATURE = "<no signature>"
    long timestamp = -1
    long traceId = -1
    int orderIndex = -1
    string operationSignature = NO_SIGNATURE
    string classSignature = NO_SIGNATURE
}
```

```
struct AfterOperationFailedEvent :
        AbstractOperationEvent {
    string cause = "<no cause>"
}

package kieker.common.record.geolocation

struct GeoLocation {
    int countryCode // ISO 3166-1 numeric
}
```

Listing 2: Record types for monitoring for traces and geo location

The IRL code generator is designed to be extended by any number of record generator supporting new languages or APIs. At present, we support Java, C, and Perl. These language specific generators must implement the generator API implemented through the abstract class RecordLangGenericGenerator. The class defines a set of abstract methods and provides a model query function to collect all property declarations of all parent types of a record type, a common functionality for all generators.

## 3.2 Palladio Usage Model Adaptation

The PCM includes a usage model that represents the behavior of certain classes of application users. Thus, it represents the interaction between users and the application in the form of sequences of system entry calls annotated with the workload the system has to handle. In our approach, we create a usage model based on runtime information or adapt an existing usage model accordingly. Therefore, we must collect monitoring data, interpret it, and use it to modify the usage model. The resulting model is used together with other partial PCM models for forecasting.

Analysis techniques which are part of the Kieker framework are used to analyze the monitoring records and gather call traces and workloads (i.e., frequency of calls to the system). Based on the trace information, the sequence of system entry calls is constructed. The

frequency of calls to the system is used to initialize the workload specification. For this, a statistical analysis is conducted on the monitoring data, which results in probability distributions suitable for workload specification. In addition, if the monitoring identified movements of software components from one provider to another, for example, other partial PCM models are adapted in a similar way.

Triggered by the events captured by Kieker, the PCM-based runtime model is continuously modified and thus reflects current application characteristics. In order to identify possible upcoming bottlenecks, trends have to be derived from the monitoring history. Trends in call traces and workloads are used to iteratively adapt the PCM usage model to reflect how the system usage profile may evolve when time advances. Similarly, other partial PCM models are adapted. Thus, the trend observed in history via monitoring is continued at modeling level. As soon as the monitoring identifies another trend, the PCM is adapted accordingly. For each iteration, a forecasting is conducted based on the current characteristics of the system to identify possible bottlenecks.

## 4 Related Work

The utilization of runtime models to carry out forecasts of different application qualities has received considerable attention in the past. Runtime model based performance prediction has been addressed by, e.g., Ivanovic et al. [ICH11], who propose static analysis techniques to create runtime models able to predict performance bottlenecks to appear in the near future. In order to carry out performance predictions, Leitner et al. [LWK+10] combine prediction models, created manually during design time, with monitoring data, and machine learning techniques. Schmieders et al. [SM11] propose the use of model checkers, monitoring data and context assumptions in order to proactively identify performance bottlenecks. Availability has been addressed by, e.g., Bucchiarone et al. [BMPR12], who combine multilayer monitoring data and decision trees to trigger adaptations proactively in order to maintain desired application availability thresholds. Ghezzi et al. [GT09] use Markov chains and monitoring data as input for a model checker that checks the model against availability requirements derived from SLAs.

While some solutions above are able to cope with resource re-assignments, they all fall short in coping with newly introduced services at runtime. In order to tackle this issue, the work proposed by van der Aalst et al. in [vdASS11] builds a runtime model automatically, i.e., a transition system, by analyzing and aggregating monitored method invocation traces. As being annotated with response time information, the transition system furnishes the performance forecasts of the application. Hoorn proposes the SLAstic approach [vHRGH09] that uses architecture description models (ADM) developed at design time and adapts that model at runtime based on monitoring data. However, SLAstic is not able to cope with the integration of new probe and record types. Futhermore, the ADM is a specialized model that must be manually created to meet the structure of the implementation. The aforementioned approaches do not apply usage models that aim to reflecting the use of the application by its users. As usage models have been successfully applied to predict application qualities during design time (cf. [BKR07]), we assume that the utilization of

this technique will increase the effectiveness of forecasting during runtime.

## 5 Conclusion and Outlook

The forecasting of application qualities is a complex challenge in dynamic environments such as the cloud. To address this challenge, we propose a dynamic runtime model that interrelates runtime monitoring data, collected by Kieker, and performance forecasting, carried out by Palladio, in order to avoid performance break-ins proactively, i.e., before they occur.

In this paper, we tackled two problems arising from the conceptual integration of Kieker and Palladio. Firstly, we address how to derive probes from a Palladio component model (PCM) that we need to instrument the application with probes. We propose two languages (IRL and IAL) that enable Kieker to gather performance characteristics of the monitored application, and to create a dynamic runtime model. This model will be used by Palladio to carry out performance predictions. Secondly, once the monitoring data has been gathered, it has to be forwarded to the related representations in the runtime model, such that adaptation or maintenance decisions can be taken based on the monitored behavior. To handle this problem, we developed data filters which propagate the monitoring data to the usage and allocation runtime models.

We motivated our work with a realistic scenario in which cloud components migrate over the Internet to remote data centers, which aims at avoiding performance break-ins. The problem of enforcing data-geolocations is not addressed yet and will be part of our future work. Furthermore, we plan to support the collection of arbitrary state information of the monitored application. This requires the integration of the corresponding type system on a dynamic basis, which is a non trivial task.

**Acknowledgement**
This work was partially supported by the DFG (German Research Foundation) under the Priority Programme SPP1593: Design For Future – Managed Software Evolution. (grant HA 2038/4-1, RE 1674/6-1, PO 607/3-1)

## References

[BBF09] G. Blair, N. Bencomo, and R. B. France. Models@ run.time. Computer, 42:22–27, October 2009. 0018.

[BKR07] Steffen Becker, Heiko Koziolek, and Ralf Reussner. Model-Based performance prediction with the palladio component model. In WOSP '07: Proceedings of the 6th international workshop on Software and performance, pages 54–65, New York, NY, USA, 2007. ACM.

[BKR09] Steffen Becker, Heiko Koziolek, and Ralf Reussner. The Palladio component model for model-driven performance prediction. Journal of Systems and Software, 82:3–22, 2009.

[BMPR12] A. Bucchiarone, A. Marconi, M. Pistore, and H. Raik. Dynamic Adaptation of Fragment-based and Context-aware Business Processes. 2012.

[Gro06] Object Management Group. CORBA Component Model 4.0 Specification. Specification Version 4.0, Object Management Group, April 2006.

[GT09] Carlo Ghezzi and Giordano Tamburrelli. Reasoning on Non-Functional Requirements for Integrated Services, volume 0, pages 69–78. Ieee, 2009.

[HHJ+13] Wilhelm Hasselbring, Robert Heinrich, Reiner Jung, Andreas Metzger, Klaus Pohl, Ralf Reussner, and Eric Schmieders. iObserve: Integrated Observation and Modeling Techniques to Support Adaptation and Evolution of Software Systems. Research report, Kiel University, Kiel, Germany, October 2013.

[ICH11] D. Ivanovic, M. Carro, and M. Hermenegildo. Constraint-Based Runtime Prediction of SLA Violations in Service Orchestrations, QoS-Aggregation. In G. Kappel, H. Motahari, and Z. Maamar, editors, Service-Oriented Computing – ICSOC 2011, number Tbd in Lncs. Springer Verlag, 2011.

[LWK+10] Philipp Leitner, Branimir Wetzstein, Dimka Karastoyanova, Waldemar Hummer, Schahram Dustdar, and Frank Leymann. Preventing SLA Violations in Service Compositions Using Aspect-Based Fragment Substitution. In Proceedings of the 8th International Conference on Service Oriented Computing (ICSOC 2010). Springer Berlin Heidelberg, December 2010.

[MG11] Peter Mell and Timothy Grance. The NIST Definition of Cloud Computing. Technical Report 800-145, National Institute of Standards and Technology (NIST), Gaithersburg, MD, September 2011.

[RRMP11] Andreas Rausch, Ralf Reussner, Raffaela Mirandola, and Frantisek Plasil, editors. The Common Component Modelling Example (CoCoME), volume 5153 of Lecture Notes in Computer Science. Springer Verlag Berlin Heidelberg, 2011.

[SM11] Eric Schmieders and Andreas Metzger. Preventing Performance Violations of Service Compositions using Assumption-based Run-time Verification. In ServiceWave, 2011.

[vdASS11] W.M.P. van der Aalst, M.H. Schonenberg, and M. Song. Time prediction based on process mining. Information Systems, 36(2):450–475, April 2011.

[vHFS11] Reinhard von Hanxleden, Hauke Fuhrmann, and Miro Spönemann. KIELER—The KIEL Integrated Environment for Layout Eclipse Rich Client. In Proceedings of the Design, Automation and Test in Europe University Booth (DATE'11), Grenoble, France, March 2011.

[vHRGH09] André van Hoorn, Matthias Rohr, Asad Gul, and Wilhelm Hasselbring. An Adaptation Framework Enabling Resource-Efficient Operation of Software Systems. In Nenad Medvidovic and Tetsuo Tamai, editors, Proceedings of the 2nd Warm-Up Workshop for ACM/IEEE ICSE 2010 (WUP '09), pages 41–44. ACM, April 2009.

[vHWH12] André van Hoorn, Jan Waller, and Wilhelm Hasselbring. Kieker: A Framework for Application Performance Monitoring and Dynamic Software Analysis. In Proceedings of the 3rd ACM/SPEC International Conference on Performance Engineering (ICPE 2012), pages 247–248. ACM, April 2012.

[WLK11] Bing Wei, Chuang Lin, and Xiangzhen Kong. Energy optimized modeling for live migration in virtual data center. In 2011 International Conference on Computer Science and Network Technology (ICCSNT), volume 4, pages 2311–2315, 2011.