1 Abstracts of all SOSP 2014 Contributions

1.1 The Descartes Modeling Language: Status Quo
(*Samuel Kounev, Fabian Brosig, and Nikolaus Huber*)

This presentation will present a summary of the latest developments on the Descartes Modeling Language and related tools. The Descartes Modeling Language (DML: http://www.descartes.tools), also referred to as Descartes Meta-Model (DMM), is a new architecture-level language for modeling performance and resource management related aspects of modern dynamic software systems and IT infrastructures. DML provides appropriate modeling abstractions to describe the resource landscape, the application architecture, the adaptation space, and the adaptation processes of a software system and its IT infrastructure. Technically, DML is comprised of several sub-languages, each of them specified using OMGs Meta-Object Facility (MOF) and referred to as meta-model in OMGs terminology. The various sub-languages can be used both in offline and online settings for application scenarios like system sizing, capacity planning and trade-off analysis as well as for self-aware resource management during operation. The talk will first introduce the various sub-languages and then present a summary of several case studies showing how DML can be applied in real-life scenarios. Finally, an overview of related tools in the Descartes Tool Chain (http://www.descartes.tools) will be presented.

1.2 Evaluating the Prediction Accuracy of Generated Performance Models in Up- and Downscaling Scenarios
(*Andreas Brunnert, Stefan Neubig, and Helmut Krcmar*)

This paper evaluates an improved performance model generation approach for Java Enterprise Edition (EE) applications. Performance models are generated for a Java EE application deployment and are used as input for a simulation engine to predict performance (i.e. response time, throughput, resource utilization) in up- and downscaling scenarios. Performance is predicted for increased and reduced numbers of CPU cores as well as for different workload scenarios. Simulation results are compared with measurements for corresponding scenarios using average values and measures of dispersion to evaluate the prediction accuracy of the models. The results show that these models predict mean response time, CPU utilization and throughput in all scenarios with a relative error of mostly below 20

1.3 Investigating the Use of Bayesian Networks in the Hora Approach for Component-based Online Failure Prediction
(*Teerat Pitakrat and André van Hoorn*)

Online failure prediction is an approach that aims to predict potential failures that can occur in the near future. There are a number of techniques that have been used, e.g., time

series forecasting, machine learning, and anomaly detection. These techniques are applied to the data that can be collected from the system and that contain information regarding the current state of the system, such as, response time, log files, and resource utilization.

The existing works which employ these prediction techniques to predict failures can be grouped into two categories. The first category approaches the task by using the technique to predict failures at specific locations in the system. For example, time series forecasting may be used to predict the response time at the system boundary. Once it is predicted that the response time tend to go beyond a certain value in the near future, a warning is then issued. On the other hand, the second category applies the prediction technique to the whole system, i.e., using the data collected from all locations and creating a model that can analyze and conclude from these data whether a failure is expected on the system level.

In our work, we take another direction by combining techniques in the first category with the architectural model of the system to predict not only the failures but also their consequences. In other words, the existing prediction techniques provide prediction results of each component while the architectural model allows the predicted failures to be extrapolated and further predicts whether they will affect other components in the system.

We have developed the prediction framework based on Kieker's pipe-and-filter architecture and employed its monitoring capability to obtain the data at runtime. The architectural model of the system is extracted from the monitoring data and used to create a Bayesian network that can represent the failure dependency between components. The prediction results obtained from each component failure predictors are forwarded to the Bayesian network to predict the failure propagation.

1.4 Predicting Energy Consumption by Extending the Palladio Component Model (*Felix Willnecker, Andreas Brunnert, and Helmut Krcmar*)

The rising energy demand in data centers and the limited battery lifetime of mobile devices introduces new challenges for the software engineering community. Addressing these challenges requires ways to measure and predict the energy consumption of software systems. Energy consumption is influenced by the resource demands of a software system, the hardware on which it is running, and its workload. Trade-off decisions between performance and energy can occur. To support these decisions, we propose an extension of the meta-model of the Palladio Component Model (PCM) that allows for energy consumption predictions. Energy consumption is defined as power demand integrated over time. The PCM meta-model is thus extended with a power consumption model element in order to predict the power demand of a software system over time. This paper covers two evaluations for this meta-model extension: one for a Java-based enterprise application (SPECjEnterprise2010) and another one for a mobile application (Runtastic). Predictions using an extended PCM meta-model for two SPECjEnterprise2010 deployments match energy consumption measurements with an error below 13 %. Energy consumption predictions for a mobile application match corresponding measurements on the Android operating system with an error of below 17.2 %.

1.5 Towards Modeling and Analysis of Power Consumption of Self-Adaptive Software Systems in Palladio
(*Christian Stier, Henning Groenda, and Anne Koziolek*)

Architecture-level evaluations of Palladio currently lack support for the analysis of the power efficiency of software systems and the effect of power management techniques on other quality attributes. This neglects that the power consumption of software systems constitutes a substantial proportion of their total cost of ownership. Currently, reasoning on the influence of design decisions on power consumption and making trade-off decisions with other Quality of Service (QoS) attributes is deferred until a system is in operation. Reasoning approaches that evaluate a system's energy efficiency have not reached an abstraction suited for architecture-level analyses. Palladio and its extension SimuLizar for self-adaptive systems lack support for specifying and reasoning on power efficiency under changing user load. In this paper, we (i) show our ideas on how power efficiency and trade-off decisions with other QoS attributes can be evaluated for static and self-adaptive systems and (ii) propose additions to the Palladio Component Model (PCM) taking into account the power provisioning infrastructure and constraints.

1.6 Integrating Workload Specification and Extraction for Model-Based and Measurement-Based Performance Evaluation: An Approach for Session-Based Software Systems
(*André van Hoorn, Christian Vögele, Eike Schulz, Wilhelm Hasselbring, and Helmut Krcmar*)

Workload modeling and generation/simulation are essential to systematically and accurately evaluate performance properties of software systems for both measurement-based (e.g., load testing and benchmarking) and model-based performance evaluation (e.g., prediction). For measurement-based evaluation, established load generation tools exist that take workload specifications as input and generate corresponding requests to the system under test (SUT). Specifications of workloads are also tightly integrated in formalisms and approaches for model-based performance evaluation, including analytic and simulative techniques. The definition of workload specifications that represent the real workload as accurately as possible is one of the biggest challenges in both areas. Only few approaches for sharing workload specifications between the model-based and the measurement-based worlds exist.

In this talk, we will present our approach that aims to interchange and automate the extraction and transformation of workload specifications for measurement-based and model-based performance evaluation of session-based software systems. The approach comprises three main components: First, a domain specific language (DSL) allows the layered modeling of probabilistic workload specifications of session-based systems. Second, we automatically extract instances of this DSL from recorded session logs of production systems, e.g., employing Kieker. During this extraction process, different groups of customers with similar navigational patterns are identified using configurable clustering algorithms. Third,

instances of the DSL are transformed to workload specifications for load generation tools and model-based performance evaluation tools. We will present a transformation to the common load testing tool Apache JMeter, including the Markov4JMeter extension developed in our previous work. Moreover, we will present our work in progress on transforming instances of the DSL to workload models in the Palladio Component Model. We evaluate our approach by comparing workload-specific characteristics (e.g., session lengths and arrival rates) from the recorded and the extracted/generated workload of the industry benchmark SPECjEnterprise2010.

1.7 6 years of Performance Modeling at ABB Corporate Research (*Heiko Koziolek*)

Despite significant scientific research, systematic performance engineering techniques are still hardly used in industry, as many practitioners rely on ad-hoc performance firefighting. It is still not well understood where more sophisticated performance modeling approaches are appropriate and the maturity of the existing tools and processes can be improved. While there have been several industrial case studies on performance modeling in the last few years, more experience is needed to better understand the constraints in practice and to optimize existing tool-chains. This talk summarizes six years of performance modeling at ABB, a multi-national corporation operating mainly in the power and automation industries. In three projects, different approaches to performance modeling were taken, and experiences on the capabilities and limitations of existing tools were gathered. The talk reports on several lessons learned from these projects, for example the need for more efficient performance modeling and the integration of measurement and modeling tools.

1.8 Performance Challenges in a Mainframe System (*Stefan Fütterling and Michael Großmann*)

The global ordering system of a car manufacturer processes several hundred thousand car orders each year. It is used by thousands of dialog-users at car-dealerships, market systems, production plants and several other third party systems which are also supplied with all necessary data. The system is running on an IBM mainframe and uses several technology stacks such as CICS/Cobol, Websphere Application Server/Java, Messaging and DB2. The system is subject to a constant change and growth due to enhanced functionalities, the addition of new markets and market systems, as well as the rise in car sales and the increasing complexity of the cars themselves.

The general growth has lead to an increase in mainframe costs, where today no one can answer exactly the questions on how much CPU consumption a user (or user-class) is triggering or in which parts of the system the most CPU consumption is caused. Additionally the goal for 2016 is, to charge-back CPU cost on a user-class basis. Here, a detailed reporting is necessary, where CPU consumption is evaluated on a per user-class and per business

transaction level. The new reporting will help to improve the operation, the optimization and the analysis of the system.

The general questions to be answered are:

- Who is calling which business transaction and how much cpu-time is consumed?
- How expensive is a business transaction?
- How can the cpu-consumption of secondary transactions be evaluated in an event based system and how can it be attributed to user-classes and/or business transactions?
- Which technology (i.e. CICS/Cobol or WAS/Java) causes the growth? Where are the cost drivers?
- How large is the future cpu-consumption in the context of a forecast for the predicted growth in car sales and additional functionality due to new releases.

The presentation describes the gathering of CPU consumption data, its aggregation on basis of a user-class mapping and the creation of performance reports and performance predictions.

1.9 Using the Free Application Performance Diagnosis Tool "inspectIT" (*Stefan Siegl*)

inspectIT (http://www.inspectit.de) is a free Java performance diagnosis solution developed at NovaTec Consulting. In this live presentation I will guide you through the features of inspectIT. We will be analyzing a sample application (DVDStore), that NovaTec extended to suffer from certain performance problem patterns, which we often see at our customers.

- *<phone rings>*
- *Customer Support:* "We once again had a complaint about hang ups and slow response times when our customer wanted to buy a DVD on our store."
- *Development Guy:* "We set up tests for this situation already; we do not see any problems. Did the customer tell you what he did specifically? We are at a loss here"

...this is where we will come in and try to figure out what is actually going on in the DVDStore. Is there really a problem in the code? Is it the user not using the application correctly? We will find out.

1.10 LibReDE: A Library for Resource Demand Estimation (*Simon Spinner and Jürgen Walter*)

When creating a performance model, it is necessary to quantify the amount of resources consumed by an application serving individual requests. In distributed enterprise systems, these resource demands often cannot be observed directly, their estimation is a major challenge. Different statistical approaches to resource demand estimation based on monitoring data have been proposed in the literature, e.g., using linear regression, Kalman filtering, or optimization techniques. However, the lack of publicly available implementations of these estimation approaches hinders their adoption performance engineers. LibReDE provides a set of ready-to-use implementations of approaches to resource demand estimation. It is the first publicly available tool for this task and aims at supporting performance engineers during performance model construction. The library can be used for offline and online performance model parameterization. LibReDE helps performance engineers to select an estimation approach for a system under study by automatically selecting suitable estimation approaches based on their pre-conditions and by offering cross-validation techniques for the resulting resource demands. Based on the cross-validation results, the performance engineer is able to compare the accuracy of different estimation approaches for a system under study.

1.11 Approaching the Cloud: Using Palladio for Scalability, Elasticity, and Efficiency Analyses (*Sebastian Lehrig and Matthias Becker*)

In cloud computing, software architects develop systems for virtually unlimited resources that cloud providers account on a pay-per-use basis. Elasticity management systems provision these resource autonomously to deal with changing workload. Such changing workloads call for new objective metrics allowing architects to quantify quality properties like scalability, elasticity, and efficiency, e.g., for requirements/SLO engineering and software design analysis. However, Palladio so far did not support such novel metrics. To cope with this problem, we extended Palladios simulation approach SimuLizar by additional metrics for scalability, elasticity, and efficiency. In this tool paper, we focus on illustrating these new capabilities. For this illustration, we analyze a simple, self-adaptive system.

1.12 Static Spotter for Scalability Anti-Patterns Detection (*Jinying Yu and Goran Piskachev*)

Understanding large and old legacy systems is tedious and error-prone task. Design patterns (and anti-patterns) detection is a reverse engineering technique which allows analysis of such systems. Software architects use this technique to recover bad design decisions. Particularly, web-based applications face scalability issues. Scalability is especially impor-

tant for cloud-based systems which need to handle different environments (e.g. change of workload over time). We want to automatically detect scalability anti-patterns of existing component-based applications on the source code level. In literature, there are different ways to define the scalability anti-patterns. Moreover, there exist several general pattern detection engines, but none of them, examines the detection of scalability anti-patterns. In our work, first we define scalability anti-patterns and specify them using our domain specific language, and second we use our model-driven static spotter to detect the specified anti-patterns. There are two inputs required for our Eclipse based implementation of the static spotter: (1) the catalog of specified patterns (in our case the scalability anti-patterns) and (2) tree model representation of the source code of our application. To demonstrate our work, we use an example web-based application where we detect the specified scalability anti-patterns.

1.13 CactoSim — Optimisation-Aware Data Centre Prediction Toolkit (*Sergej Svorobej, Henning Groenda, Christian Stier, James Byrne, and Pj Byrne*)

Virtualisation is a common technique in todays data centres for running Cloud-scale applications, providing isolation between diverse systems and customers as well as comparably lightweight relocation and consolidation of virtual on physical machines. Currently topology optimisation algorithms are in use in data centres, their configuration typically being trial-and-error based instead of using sound reasoning. Simulation can be used to enable the evaluation of such optimisation algorithms towards better reasoning at decision time. This paper describes work towards the development of a simulation-based prediction prototype (CactoSim) and its integration with the live data centre topology optimisation prototype (CactoOpt). This coupling enables the evaluation of runtime optimisations at design time. This paper describes CactoSim and how it provides extended architecture models at its interfaces but internally employs Palladio. It showcases a method for using extended architecture models with the base Palladio Component Model (PCM), including the required model transformations. In this specific case, the supported infrastructure model for data centres is described, which goes beyond PCM and takes into account the complexity of the real-world data centres virtualisation infrastructure. Finally, an integration method for simulation and live data centres is described, which allows CactoSim to pull live models and utilise them directly for simulation experiments.

1.14 Benchmarking Workflow Management Systems (*Marigianna Skouradaki, Vincenzo Ferme, Cesare Pautasso, Dieter Roller, and Frank Leymann*)

The goal of the BenchFlow project is to design the first benchmark for assessing and comparing the performance of BPMN 2.0 Workflow Management Systems (WfMSs). WfMSs have become the platform to build composite service-oriented applications, whose performance depends on two factors: the performance of the workflow system itself and the

performance of the composed services (which could lie outside of the control of the workflow).

Despite the rapid evolution of benchmarks, there is only a recent appearance of them targeting to Service Oriented Architecture (SOA) middleware. In particular for WfMS there is not yet a currently accepted benchmark, despite the recent appearance of standard workflow modeling languages such as BPMN 2.0. Our main goal is to present to the community the open challenges of a complex industry-relevant benchmark.

1.15 Application Performance Monitoring: Trade-Off between Overhead Reduction and Maintainability
(*Jan Waller, Florian Fittkau, and Wilhelm Hasselbring*)

Monitoring of a software system provides insights into its runtime behavior, improving system analysis and comprehension. System-level monitoring approaches focus, e.g., on network monitoring, providing information on externally visible system behavior. Application-level performance monitoring frameworks, such as Kieker or Dapper, allow to observe the internal application behavior, but introduce runtime overhead depending on the number of instrumentation probes.

We report on how we were able to significantly reduce the runtime overhead of the Kieker monitoring framework. For achieving this optimization, we employed micro-benchmarks with a structured performance engineering approach. During optimization, we kept track of the impact on maintainability of the framework. In this paper, we discuss the emerged trade-off between performance and maintainability in this context.

To the best of our knowledge, publications on monitoring frameworks provide none or only weak performance evaluations, making comparisons cumbersome. However, our micro-benchmark, presented in this paper, provides a basis for such comparisons. Our experiment code and data are available as open source software such that interested researchers may repeat or extend our experiments for comparison on other hardware platforms or with other monitoring frameworks.

1.16 The DIN/ISO Definition and a Measurement Procedure of Software Efficiency
(*Werner Dirlewanger*)

Colloquially people used to assign the attribute speed (i. e. performance) not only to hardware but also to software. But software does not have an attribute speed. It defines—for each user initiated task—a sequence of information processing steps (machine instructions) to be performed. To find a suitable term for the aspect under construction we use two implementations (Imp1 and Imp2) of the application on the same hardware and perform two performance measurements yielding the performance values P1 and P2. To find a term describing the the behaviour of the application software we compare P1 to P2. This yields a measure of how good Imp2 transforms the task submitted into a sequence of

machine instructions compared to Imp1: Imp2 is less or more efficient.

This procedure requires a fitting definition of P and a measurement method which simulates in detail the properties of the user entity and delivers detailed performance values. This is done by the ISO 14656 method. It measures the timely throughput, the mean execution times and the total throughput, each as a Vektor of m values, where m ist the total number of task types. The software efficiency values are: The timely throughput efficiency I_{TI}, the mean execution time efficiency I_{ME} and the total throughput efficiency I_{TH}, each being a vector of m values. Additionally there is the quotient of maximal timely served users I_{MAXUSR} which is a scalar.

In the talk the ISO method will be explained. All is supported by examples of real measurement projects. History: DIN 66273 defines a very detailed performance measurement method, but the software efficiency aspect was not yet part of this Standard. The use of the DIN performance measurement method for getting software efficiency values was firstly published in the author's DIN-textbook. ISO took over the idea in ISO 14756. Comprehensive description: The author's textbook on ISO 14756.

1.17 Enabling Assembly of Systems and its Implications within the Palladio Component Model
(*Misha Strittmatter*)

The scope of Palladio models has always been software systems and their inner structure and behavior. That is why the ability to assemble structures into structure of higher order stops at the system level. This makes it impossible to simulate a system in interplay with other systems without using workarounds. This contribution proposes meta-model changes, which enable the composition of systems. The implications of these changes onto the submodels of Palladio are discussed. Systems and their interfaces will then be defined within repositories. Only one composed structure diagram will be needed. Usage Models will be able to call the interfaces of one outer structure, which may be a component or system. If a model should be simulated, an allocation model for this outer structure has to be provided. As a side effect, it will then be possible to apply load directly onto a single component, which may be beneficial in some cases. Further implications onto Palladios tooling and developer role concept are also discussed. The modification will be performed in the scope of the PCM refactoring process, so it will not cause much additional development effort nor will it break functionality, as this is first mitigated by a backwards transformation.

1.18 Parallel Simulation of Queueing Petri Nets
(*Jürgen Walter, Simon Spinner, and Samuel Kounev*)

Queueing Petri Nets (QPNs) show high accuracy to model and analyze various computer systems. To apply these models at runtime scenarios a speedup of simulation is desirable.

Prediction speed at design time is of importance as well, as we can see at the transformation from Palladio Component Model (PCM) to QPNs. SimQPN is a simulation engine to analyze QPNs. At a time where multi-core processors are standard, parallelization is one way to speedup simulation. Decades of intensive research showed no general solution for parallel discrete event simulation, which provides reasonable speedups. In this context, Queueing Networks (QNs) and Petri Nets (PNs) have been extensively studied. We research the application of parallel simulation for QPNs. We developed a parallel simulation engine that employs application level and event level parallelism. The simulation engine parallelizes by the use of a conservative barrier-based synchronization algorithm. Here we optimized for common model structures and applied active waiting. The speedup for a single run depends on the capability of the model. Hence, we performed three case studies which all show speedups throughout parallelization.

1.19 Adaptive Instrumentation of Java Applications for Experiment-Based Performance Analysis

(***Henning Schulz, Albert Flaig, Alexander Wert, and André van Hoorn***)

Running load tests, instrumentation of selected application parts is a common practice in order to measure performance metrics. Instrumentation means to extend the target application by measurement probes while executing measurements. For instance, in Java bytecode instrumentation, additional commands are inserted into the bytecode of the target application. Since data generation is time-consuming, it may affect the target application and thus the measurement. Countering this problem, stepwise approaches execute several measurements while using only few measurement probes per measurement. Utilizing existing approaches, the target application has to be restarted in order to change the instrumentation. The resulting measurement overhead can cause the execution of stepwise measurements to be impracticable or bound to high manual effort. In this presentation, we introduce the Adaptable Instrumentation and Monitoring (AIM) framework enabling stepwise measurements without system restarts. Furthermore, we show the advantages of selective instrumentation with AIM over excessive instrumentations. For instance, we introduce an approach to highly precise and fully automated performance-model calibration. Thereby, the relative error is smaller than 4%, whereas excessive instrumentations introduce an error of up to 50%. Last not least, we present the embedding of AIM in the Kieker framework, merging the adaptability of AIM with the comprehensive monitoring and analysis capabilities of Kieker.

1.20 Using and Extending LIMBO for the Descriptive Modeling of Arrival Behaviors

(***Jóakim v. Kistowski, Nikolas Roman Herbst, and Samuel Kounev***)

LIMBO is a tool for the creation of load profiles with variable intensity over time both from scratch and from existing data. Primarily, LIMBO's intended use is the description

of load intensity variations in open workloads. Specifically, LIMBO can be used for the creation of custom request or user arrival time-stamps or for the re-parameterization of existing traces.

LIMBO uses the Descartes Load Intensity Model (DLIM) internally for the formalized description of its load intensity profiles. The DLIM formalism can be understood as a structure for piece-wise defined and combined mathematical functions. We will outline DLIM and its elements and demonstrate its integration within LIMBO.

LIMBO is capable of generating request or user arrival time stamps from DLIM instances. In a next step, these generated time-stamps can be used for both open workload based benchmarking and simulations. The TimestampTimer plug-in for JMeter already allows the former. LIMBO also offers a range of tools for easy load intensity modeling and analysis, including, but not limited to, a visual decomposition of load intensity time-series into seasonal and trend parts, a simplified load intensity model as part of a model creation wizard, and an automated model instance extractor.

As part of our LIMBO tutorial, we explain these features in detail. We demonstrate common use cases for LIMBO and show how they are supported. We also focus on LIMBOs extensible architecture and discuss how to use LIMBO as part of another tool-chain.

1.21 Using Java EE ProtoCom for SAP HANA Cloud (*Christian Klaussner and Sebastian Lehrig*)

Performance engineers analyze the performance of software architectures before their actual implementation to resolve performance bottlenecks in early development phases. Performance prototyping is such an approach where software architecture models are transformed to runnable performance prototypes that can provide analysis data for a specific target operation platform. This coupling to the operation platform comprises new challenges for performance prototyping in the context of cloud computing because a variety of different cloud platforms exists. Because the choice of platform impacts performance, this variety of platforms induces the need for prototype transformations that either support several platforms directly or that are easily extensible. However, current performance prototyping approaches are tied to only a small set of concrete platforms and lack an investigation of their extensibility for new platforms, thus rendering performance protoyping ineffective for cloud computing. To cope with this problem, we extended Palladios performance prototyping approach ProtoCom by an additional target platform, namely the SAP HANA Cloud, and analyzed its extensibility during the extension process. In this tool paper, we focus on illustrating the capabilities of our extension of ProtoCom. For this illustration, we use a simple example system for which we create a ProtoCom performance prototype. We particularly run this prototype in the SAP HANA Cloud.

1.22 Experience with Continuous Integration for Kieker (*Nils Christian Ehmke, Christian Wulf, and Wilhelm Hasselbring*)

Developing new features and extensions for the Kieker framework is very often an agile process. We are successfully using CI (Continuous Integration) as an assistance during the development for some years now, extending it more and more. Unit tests and performance benchmarks are executed regularly, static analysis tools ensure a constant code quality, and nightly builds allow an easy access to snapshot versions of Kieker. All of this is not only a support for more experienced Kieker developers, but also for new members joining the project.

Our presentation is an experience report about the success of CI in Kieker. We detail the structure, the used tools and the history of our CI infrastructure. We report on advantages and disadvantages recognized during the usage. Furthermore, the presentation includes an outlook for further activities regarding our CI infrastructure.

1.23 Towards Performance Awareness in Java EE Development Environments (*Alexandru Danciu, Andreas Brunnert, and Helmut Krcmar*)

This paper presents an approach to introduce performance awareness in integrated development environments (IDE) for Java Enterprise Edition (EE) applications. The approach automatically predicts the response time of EE component operations during implementation time and presents these predictions within an IDE. Source code is parsed and represented as an abstract syntax tree (AST). This structure is converted into a Palladio Component Model (PCM). Calls to other Java EE component operations are represented as external service calls in the model. These calls are parameterized with monitoring data acquired by Kieker from Java EE servers. Immediate predictions derived using analytical techniques are provided each time changes to the code are saved. The prediction results are always visible within the source code editor, to guide developers during the component development process. In addition to this immediate feedback mechanism, developers can explicitly trigger a more extensive response time prediction for the whole component using simulation. The talk will cover the conceptional approach, the current state of the implementation and sketches for the user interface.

1.24 Identifying Semantically Cohesive Modules within the Palladio Meta-Model (*Misha Strittmatter and Michael Langhammer*)

The Palladio meta-model is currently contained within one file and is subdivided into packages. This structure formed through the years, while the PCM was developed and extended. The top-level packages are partly aligned with the containment structure of the PCM submodels (e.g. repository, system). There are other top-level packages, which contain crosscutting or general concepts (e.g. parameter, composition, entity). Further

ones are concerned with quality dimensions (e.g. reliability, QoS Annotations). Some extensions distributed their new classes across packages, which were semantically fitting. Within the scope of the Palladio refactoring, this structure is being transformed into a modular structure. The goal is to divide the Palladio meta-model into smaller, semantically cohesive meta-models (meta-model modules). This has several advantages. The meta-model becomes better maintainable and easier to extend and understand. When a Palladio model is created, the model developer can choose the meta-model modules which are relevant to him and is not confused by the full amount of content from all extensions. Within this contribution, first, the current structure of the Palladio meta-model is briefly explained. Next, the semantically cohesive modules, which are currently contained or even scattered across the current structure, are presented and their interdependencies explained.

1.25 Evolution of the Palladio Component Model: Process and Modeling Methods (*Reiner Jung, Misha Strittmatter, Philipp Merkle, and Robert Heinrich*)

Palladio and its component meta-model have grown over time to support a wide variety of models for application performance prediction. The present meta-model is a large interwoven structure. Specific additions, e.g., for reliability and other quality properties, have been developed separately and resulted often in derived meta-models which cannot be used together. Furthermore, as discussed on KPDAYS 2013, certain views, typing and instance definitions lack a precise definition. Therefore, the Palladio Component Model (PCM) must evolve into a modular and easy to extend meta-model, and the Palladio tools must support this modularity to allow users to combine those Palladio functionality they need for their purposes. To achieve this modularity, we must solve two challenges. First, the modularization well be a long running task. Therefore, we need a process and road map to guide us from the present Palladio to the modular future Palladio. And second, the complexity of the existing metamodel and the need to keep the resulting meta-models maintainable, we need guidance for the modularization of the PCM and its tooling. Furthermore, the use methods must support future evolution steps. In this presentation, we first provide a brief summary on our method to construct modular meta-models based on aspect and view based modeling. This general distinction is supported by collected construction advice and contextual meta-model patterns. Furthermore, we illustrate how tools should be constructed around such metamodels. And second, we lay out a plan, how this evolution of the PCM can be realized in a stepwise approach which could also be distributed among different volunteers. As the tooling is an integral part of Palladio, the evolution of the meta-model will cause alterations to the tooling. Therefore, the plan encompasses both tool and meta-model evolution.

1.26 Toward a Generic and Concurrency-Aware Pipes & Filters Framework (*Christian Wulf, Nils Christian Ehmke, and Wilhelm Hasselbring*)

The Pipes-and-Filters design pattern is a well-known pattern to organize and execute components with sequential dependencies. The pattern is therefore often used to perform several tasks consecutively on large data streams, e.g., during image processing or dynamic analyses. In contrast to the pattern's familiarity and application, almost each common programming language lacks of flexible, feature-rich, fast, and concurrency-aware Pipes-and-Filters frameworks. So far, it is common practice that most developers write their own implementation tailored to their specific use cases and demands hampering any effective re-use. In this paper, we discuss Pipes-and-Filters architectures of several Java-based applications and point out their drawbacks concerning their applicability and efficiency. Moreover, we propose a generic and concurrency-aware Pipes-and-Filters framework and provide a reference implementation for Java called TeeTime.

1.27 Evaluation of Alternative Instrumentation Frameworks (*Dušan Okanović and Milan Vidaković*)

When monitoring application performance parameters under production workload, during continuous monitoring process, everything must be done to reduce the overhead generated by monitoring system. This overhead is highly unwanted because it can have negative effect on the end user's experience. While this overhead is inevitable, certain steps can be taken to minimize it.

Our system for continuous monitoring - DProf - uses instrumentation, and sends gathered data to the remote server for further analysis. After this data is analyzed, new monitoring configuration can be created in order to find the root cause of the performance problem. New parameters turn monitoring off, where performance data is within expected, and on, where data shows problems. This way, the overhead is reduced, because only problematic parts of software are monitored. The first version of our tool used AspectJ for instrumentation. The downside of this approach was the fact that the resulting bytecode is not fully optimized, generating even higher overhead than expected. Also, the monitored system had to be restarted each time monitoring configuration changed in order to apply new monitoring parameters.

We have explored the use of other AOP or AOP-like tools, mainly DiSL framework, with our system. The goal is to find a tool that has lower overhead than AspectJ. Support for runtime changes of monitoring configuration is also considered.

1.28 Cloud Application Design Support for Performance Optimization and Cloud Service Selection
(*Santiago Gómez Sáez, Vasilios Andrikopoulos, and Frank Leymann*)

The introduction of the Cloud computing paradigm and the significant increase of available Cloud providers and services have contributed in the last years to increase the number of application developers strongly supporting to partially or completely run their applications in the Cloud. However, application developers nowadays face application design challenges related to the efficient selection among a wide variety of Cloud services towards efficiently distributing their applications in one or multiple Cloud infrastructures. Standards like TOSCA allow for the modelling and management of application topology models, further supporting the application distribution capabilities, potentially in a multi-Cloud environment. However, such approaches focus on enabling portability among Cloud providers, rather than assisting the application developers in the Cloud services selection decision tasks. In this work we focus on the challenges associated with the application performance requirements and evolution, and therefore aim to define the means to support an efficient application (re-)distribution in order to efficiently handle fluctuating over time workloads. There are two fundamental aspects which must be taken into consideration when partially or completely running the application in the Cloud. Firstly, the distribution of the application in the Cloud has a severe effect on the application performance however it is not always evident whether it is beneficial or detrimental. Secondly, an application workload, and therefore its resources demands, fluctuates over time, and its topology may have to be adapted to address the workload evolution.