Abstract—In software engineering, performance specifications of components support the successful evolution of complex software systems. Having trustworthy specifications is important to reliably detect unwanted effects of modifications on the performance using prediction techniques before they are experienced in live systems. This is especially important if there is no test system available and a system can’t be taken down or replaced in its entirety. Existing approaches neglect stating the quality of specifications at all and hence the quality of the prediction is lowered if the assumption that all used specifications are suitable does not hold. In this paper, we propose a test-based approach to validate performance specifications against deployed component implementations. The validation is used to certify specifications which in turn allow assessing the suitability of specifications for predicting the performance of a software system. A small example shows that the certification approach is applicable and creates trustworthy performance specifications.

I. INTRODUCTION

Performance prediction plays an important role in the development and evolution of complex component-based software systems. For example, their use in the development phase enables software engineers to predict the performance of different design alternatives and hence select the best alternative. In the deployment phase, these specifications guide the selection and sizing of an appropriate execution environment and on the deployment of components within this environment. In the maintenance or evolution phase, performance predictions allow to examine the effect of modifications on the performance and reduce the probability of discovering unwanted behavior in live systems. This detection is especially important if there is no test system and test data available and a system can’t be taken down or replaced in its entirety. In general, prediction techniques for the performance of component-based system use specifications for the components’ performance-relevant behavior as well as information on their assembly for their predictions.

The reliance on such performance predictions requires a predictable assembly of components. A predictable assembly of components in turn requires trust in both, the prediction method as well as the component performance specifications. Prediction method should be based on a sound and falsifiable scientific theory. Component specifications should state objective, test- and verifiable information on the performance-relevant behavior. Testing by independent certification authorities according to a procedure checked by experts can ensure the trustworthiness of such specifications. Due to the necessary effort the testing of complex software systems needs to be limited in most cases to certain parameter ranges. For example testing a performance specification regardless of deployment environment and usage profile requires in general prohibitive effort. Explicit statements about such limits enable statements about quality of the prediction and aid in identify potential risks.

Existing performance prediction approaches rely on the capability of software engineers to select suitable performance specifications of a component. Research focused on validating prediction approaches under the assumption that the specifications were suitable for the situation at hand. Validating performance specifications is seen as manual activity during the generation of the specifications. Reasoning about the suitability or quality of an existing performance specification is however complicated if there are no explicit validity statements connected to it. Especially if specifications should be reused in different context, for example because of late composition, missing validity statements increase overall efforts as specifications have to be recreated and revalidated.

In this paper, we propose an approach to certify performance specifications and explicitly state the quality and limitation of the specifications. The approach is based on a test-based validation of the specifications against deployed component implementations. Statistical reasoning is used to assure trust in the performed validation. This enables the verifiability by third parties and eases testing the validity. Additionally, it aids in the protection of interests and trade secrets in marketplaces as it is sufficient to publish the certified specifications.

The applicability of the approach to create trustworthy specifications is demonstrated in a small example. The necessary validation effort as well as the certifiable quality are shown and discussed.

The paper is structured as follows. Section II contains a description of the information necessary to validate performance specifications. The envisioned certification process is described in Section III. Section IV contains a description of existing influencing factors on software performance and which specification is chosen in the presented approach. Section V shows a small certification example. Section VI points out and
discusses related work. Section VII concludes the paper.

II. Validating Performance Specifications

Validation of specifications is often made based on test suites. For example, the functional validation if an application server fulfills the Java platform enterprise edition requirements. Each test in such a test suite provides a simple pass or fail outcome by comparing expected and experienced results.

However, if performance is considered the comparison of expected and experienced results is much more difficult. Performance is for example influenced by input parameters, internal state, the performance of other required services, resource contention, and the deployment platform. Depending on the use measurement method and granularity, performance measurements often influence the performance itself. Additionally, experience shows that measured execution times are scattered in productive environments. Reasons are manifold, e.g. physical effects like heat or age, or caching of data, concurrent processing of background services, and the precision of measurements can cause scatter. Many performance specifications only refer to average or worst case considerations. These are for example used in Service Level Agreements (SLA) to ensure adequate performance.

Validating performance specifications stating the exact execution time requires determining for which environments which variations between expected and experienced results are acceptable. For example, a specification can state that for a given environment an undisturbed execution takes between 4.9 and 5.1 ms, or equally that the undisturbed execution takes 5 ms with an acceptable deviation of ±2%. The definition of acceptable variations between expected and experienced result depends on two independent factors besides the environment. One factor is the range of input parameters for which the specification is valid and the second factor is the precision of the specified resource demands. All factors are explained further in the following paragraphs.

A. Hardware Environment

The hardware environment describes the hardware and its configuration for which the specification is valid. If a specification is validated for an environment it may also be valid for other environments. This is due to the fact that prediction approaches can allow transferring specifications from one environment to another. The environments for which transfers are supported can be considered as an equivalence class and validation can be limited to one instance of this class. An example in which transfers work quite well is if only the speed of a processor changes between two environments. However, automated and trustworthy reasoning on this would require machine-readable specifications and validating the specification transfer capabilities of the prediction approach.

It is assumed that software engineers using the prediction approaches know the limits of the transferring capabilities and are aware of their implications. The validation of transfer capabilities is not part of the presented approach. An example for a hardware specification is a Pentium IV Northwood with 2.6 Ghz, Intel Chipset, SATA hard drive, and 2 GB of RAM.

B. Software Environment

The software environment describes the software and it’s configuration for which the specification is valid. The same capabilities and limitation apply as for hardware environments. An example for a specification is a Sun JRE 1.5 on Windows XP SP3 with the JRE configured in server mode.

C. Input Range

The input range specifies the parameter ranges for parameterized specifications for which the specification is valid. For example, if a specification has the parameter file size and it is validated for file sizes between 3 and 50 MB this should be stated as input range.

D. Resource Demand Precision

The resource demand precision states how exact resource demands of the specification are within the input range in relation to the implementation executed in the specified environment. The specification is considered valid as long as specified and measured resource demands are below the deviation threshold defined as resource demand precision. A certain deviation will exist in most cases. This is either due to the measurement method or due to the fact that performance specifications are abstractions of the real behavior. Depending on the kind of the specified demand (expected constant value, gaussian distribution, arbitrary distribution) different validation estimators can be used. A Validation Estimator can for example be an interval, the mean, maximal deviation, variance, or the Mann-Whitney-Wilcoxon test. As statistical testing is used for validation, the validity statement can only be made with a certain probability. For example the certainty that the distribution of traces from the specification and the implementation are from the same distribution can be 95%. An example of a resource demand precision for an expected resource demand of 100 ms and a maximal deviation of ±10% will still accept a measured value 105 ms.

III. Certification of Performance Specifications

Our certification approach implements a test-based validation of performance specifications against deployed component implementation. The certification process is sketched in figure 3 and explained in the following paragraphs.

At the beginning, the component creator issues a validation request to the certification authority. He has to provide the component specification, the respective implementation, and the validity statements which should be used for validation. Validity statements contain the information discussed in chapter II.

An evaluator within the certification authority then assesses the validity of the specification. Therefore, execution of the following steps is necessary.
A. Generate Testcases

In this step, test cases are generated automatically which are used to evaluate the specification. Automatic generation is chosen to ensure reproducibility of validation results and reduce the necessary human effort. If random numbers are used, a pseudo-random number generator has to be used and initialized. The initialization number has to be stored along with the validation information.

Both directions, specification against executed implementation and vice versa, are checked by the test cases. The forward direction checks if the statements in the specification are correct, for example that a resource demand depends linearly on an input parameter. The backward direction checks that the implementation does not contain more dependencies than the ones specified, for example unspecified calls to external services, another sequence of calls, or unspecified dependencies to return values of calls. White-box code-analysis techniques are used to discover these dependencies. Additionally, input parameters which should not influence performance according to the specification should be varied randomly for the test cases to raise the chance of detecting otherwise undiscovered effects. However, without exhaustive checking of all possibilities the chance remains that there are some values which would lead to an invalid specification. This is mitigated by publishing the sample size for the measured demands so Software Engineers can judge for themselves if the risk is acceptable.

If there are required services, mock-ups for the required services are created to allow checking return-value dependencies. As with input parameters, ranges for the return values should be specified for which the validation is executed.

B. Instrument Implementation

The implementation itself must be instrumented in order to log the resource demands within the component and correlate them to the resource demand of the specification. Depending on the measurement method either measurement facilities are directly inserted into the code, platform functions of an adapted application or virtual machine container are applied, or operating system functions accessing the scheduled demands are used.

C. Deploy Implementation

The instrumented implementation and mock-ups must then be deployed in the target hardware and software environment. The environment must match the environment specified in the validity statements to allow a meaningful result of the certification. If some properties of an environment are not important for a specification and are hence not stated in the hardware environment, the evaluator can choose an appropriate environment. An example is if a component does not issue requests for a hard disk it is not relevant which hard disk exists in the execution environment.

D. Run Testcases

The testcases derived in the first step are run on the deployed implementation and measurements of the necessary resource demands are gathered. A test case is run until the requested sample size is reached, which can be either specified directly or being calculated by a confidence level. The overhead for storing the measurements themselves should be as small as possible to prevent unwanted side effects on the performance of the implementation. The measurements from all runs of the test cases are denoted as performance results.
The evaluator uses the performance results to reason about the validity of the supplied specification. He uses statistical analysis to check if the measured values are considered valid with respect to the validity statements. If this is the case then a certificate is issued referencing the tested implementation, validity statements and component specification. If the specification is not valid then the testcase(s) leading to the rejection of the validity are given to the component creator.

The presented approach currently has the following limitations and assumptions:

- The approach currently considers only the validation of resource demands for processors. Hard disk or network requests are not validated.
- It focuses on the performance needs of business information systems and does not consider guarantees, for example by a schedulability analysis, which are important in many embedded systems. The methodology is intended to compare experienced execution times and not to reason about best or worst case execution times.
- Validation is currently limited to basic components. The specifics of composite components specifications, e.g. how wiring the components is implemented, are not considered at the moment. However, if composite components are specified as basic components the approach is applicable.
- Resource demand specifications are currently only validated for fixed resource demands in the specification and a maximal deviance in percent.
- Checks that the implementation does not contain more dependencies than specified are currently not implemented.
- Data dependencies of components are currently not validated yet. However, the support for parameters of the used performance specifications allows taking these into account later on.
- Plain java objects are considered. Code weaving which is for example used in Java Enterprise Edition application servers is not considered yet.

IV. COMPONENT PERFORMANCE SPECIFICATIONS

In this chapter, links to existing prediction approaches and their component performance specifications are given. We show the selection criteria for the specification which promises the highest pay off for certification and introduce the selected specification.

Most of the existing performance prediction and assessment approaches provide own performance specifications for components. These specifications describe the performance-relevant behavior of individual components and are used by the prediction approach to reason about the behavior of an assembled system. A general survey on model-based performance prediction approaches was created by Balsamo et al. [2]. They point out and compare which information is required by each of the approaches as well as on which software life cycle phase they focus, the degree of automation, and tool support. Another view is provided by Becker et al. in [3]. They focus their survey on performance prediction of component-based systems from an engineering perspective. They review existing approaches and tools and indentify their respective strengths and weaknesses in supporting different aspects in the design and development of component-based systems.

In general, performance specifications depend on the methodology used in the approach to predict the performance as well as which influencing factors on software components are taken into account. These influencing factors are stated for example by Koziolek in [4, p.42]:

- Implemented Algorithms
- Service Parameters and Internal State
- Performance of Required Services
- Resource Contention
- Deployment Platform

We reviewed the different existing prediction approaches and specifications in order to identify the one with the best support of accounting for the influencing factors and which is suitable for complex component-based systems. These requirements were selected because a better consideration of these factors fosters reusing a specification in different context. A higher degree of reuse promises higher payoffs for certification effort. Finally, the performance specifications of the Palladio Component Model (PCM) [1] were selected. PCM specifications allow taking implemented algorithms, service parameters, dependencies to required services, resource contention and dependencies to the deployment platform into account. However, the prediction of resource contention effects is focused on the resource processor. Other resources, e.g. memory or hard disk access, are only considered on a basic level.

In PCM, the behavior of services provided by a component is described by so-called Resource Demanding Service EFFect (RDSEFF) specifications. They describe the control and data flow of a component. The description is as abstract as possible while still allowing accounting for the influencing factors listed in the last paragraph. The elements of a RDSEFF and their relation to the influencing factors are explained in the following paragraph. More detailed descriptions of RDSEFFs are available in [5], [6], and [1].

The behavior modeled in a RDSEFF consists of required service calls, branches, loops, resource demands, resource acquisions, resource releases, and forked behaviors. Of course, all of these reflect the implemented algorithm or, if still in the design stage, the estimation of the performance of an implemented algorithm. All parameters of the elements, e.g. number of loops or resource demand, can be specified as arbitrary distribution functions and/or contain service parameters which influence the demand.

a) Required service calls: allow accounting for the performance of required service by making the point of a service call explicit and allow weaving in specifications for the behavior of the required service. These calls can be parameterized to take service parameters into account.

b) Branches and loops: allow accounting for service parameters influencing the performance of the component
void execute(int number, List array) {
    requiredService1();
    // internal computation
    innerMethod();
    if (number >= 0)
        for (item in array)
            requiredService2();
    else
        requiredService3();
}

Figure 2. Example: Source Code and corresponding RDSEFF [1]

c) Resource demands: allow to account for the effect of resource contention due to the demands issued by components. The demand in abstract units of a component on resources is specified. If the deployment of the component is known these units can be converted and the time required completing the demand computed.

d) Resource acquisition and release: allows accounting for the internal state of components and enables synchronization.

e) Forked behavior: allows accounting for resource contention effects even within a component by enabling to express concurrent execution of behavior within a component.

Figure 2 provides a simple example of the structure and information contained in a RDSEFF. In the example, the abstract performance-relevant behavior of the service execute of a component is described. Algorithmic complexity and substantial source code is hidden in the method calls to allow easier presentation. First, a call to a required service is issued. After its completion a component internal calculation is made, encapsulated in the innerMethod. The execution of this calculation requires 1000 units on the processor. Afterwards, the control flow splits depending on the number of elements in the service parameter array. Either the required service requiredService2 is executed as many times as array has elements, or the required service requiredService3 is executed.

V. Example

In this chapter, a simple example demonstrates the validation of a performance specification.

The validated component is labeled ComponentUnderTest. It requires the interface ProcessingRequest which provides the parameterless service process. It requires a component implementing the interface HelperService which provides the parameterless service calculate. The RDSEFF of ProcessingRequest for process first request 500 units of CPU demand for an internal calculation (innerMethod). Afterwards the required service is executed and another internal calculation requests 250 units of CPU demand (dataProcessing). This is depicted in figure 3.

Two implementations are generated using the performance prototype approach of Becker et al. [7]. Both implementations are based on the specification, but the second one should request 270 units of CPU demand at dataProcessing.

The validity statements are as follows. The hardware environment consists of an Intel Core 2 processor T5600, Intel chipset, and 2 GB RAM. One CPU unit in the specification equals 0.0001 ms on that machine. The software environment consists of Windows XP Professional SP3, a SUN Java VM 1.6.0_13 with HotSpot Client VM (build 11.3-b02, mixed mode, sharing). The example does not use input parameters, so none are listed. The resource demand precision for all resource demands uses intervals as validation estimator. 95% of the experienced resource demands of the internal action innerMethod should lie in the interval 50ms ± 6ms, and within 25ms ± 3ms for the internal action dataProcessing.

In the generate testcases phase, a sample size of 10000 is selected to be sure that a validation returns a trustworthy result. As there are no parameters in the specification there is just one test requesting the service process. In the example, the validation is limited to the forward direction. A mock-up for
the required service is created manually. To demonstrate that any internal processing, for example to calculate any returned values, does not negatively influence validation an internal demand of 250 CPU units is issued (see also figure 3). In the instrument implementation phase, the code is instrumented manually. Passed time is measured using the wall-clock time returned by the Java operation System.nanoTime(). The implementation is then deployed manually in the validation environment. In the run testcases phase the test are run automatically until the sample size is reached.

The results for *dataProcessing* are depicted in figure 4. For implementation 1 and *dataProcessing*, 96.26% lie within the interval. For implementation 2 and *dataProcessing*, 37.63% which would lead to the decision that the specification is invalid for this implementation. For *innerMethod*, the values are 97.47% and 97.36% respectively. Overall, implementation 1 is regarded as valid.

The example is limited to the call of required services and processing of internal actions. The other constructs are not supported by this early lab prototype because they depend on validating parameterized specifications and considered as future work. The value of 95% of all measurements that should lie within an interval was chosen as the measurement method used wall-clock time and the used validation system had many low-load background jobs running which influenced the measurements.

**VI. RELATED WORK**

Related work can be split into three different areas: Certifications, Performance Testing, and Performance Specification Validation. Each area is presented in a separate subchapter.

**A. Certification**

The research on component certification started in the early 90s and is still ongoing as shown in Alvaro et al.’s survey [8]. The survey shows the history of component certification and that certification approaches developed in the 90s focus on statements about the reliability of software components using test cases or mathematically analyzable models. Starting around 2000, the focus of the approaches shifted towards the certification of extra-functional aspects in general and the prediction of systems built out of certified components, e.g. [9], [10].

Wallnau also did some basic work on classifying certification approaches, for example the 10 useful distinctions for certification approaches in [11]. According to this classification the approach presented in this paper aims at reducing the gap between the knowledge about what a component does to what it actually does. It supports a *descriptive* certification of *objective* measures. The software products will be examined *empirically* with a given *context* in a *procedural* manner.

Meyer introduced in [12] a component quality model which distinguishes certification approaches between a “low road”
and a “high road”. The low road summarizes the validation of existing component behavior and the high road means verification of component behavior with fully proven correctness properties. The approach described in this paper is based on statistical and empirical testing and hence belongs to the former category.

Hissam, Moreno, Wallnau et al. introduced the concept of predictable assemblies in [10] and later extended it to the Predictable Assembly from Certifiable Components (PACC) approach, described in [13]. PACC allows the prediction of the runtime behavior of a software system from the properties of its components and their patterns of interactions. PACC’s performance reasoning framework currently focuses on fixed-priority preemptive scheduling, making it suitable to analyze hard real-time systems [14]. In contrast to the approach proposed in this paper, the authors concentrate on the support for hard real-time systems instead of business information systems. Additionally, they focus on the small areas and conditions in which some correctness properties can be proven.

The Cleanroom Software Engineering approach of Mills et al. [15] and the corresponding process of [16] target reliability of software systems. The approach and process aim to make development more manageable and predictable by using statistical quality control. The philosophy behind cleanroom software engineering is to avoid dependences on costly defect-removal processes by writing code increments right the first time and verifying their correctness before testing. Its process model incorporates the statistical quality certification of code increments as they accumulate into a system. Cleanroom software engineering yields software that is correct by mathematically sound design and software that is certified by statistically-valid testing. In contrast to the approach presented in this paper, the Cleanroom Software Engineering approach belongs to the high road approaches. However, statistical quality control is also used in the approach presented in this paper.

Alvaro et al. show in [17] the need of component certification within component-based software development for business information systems and discuss similarities and interdependencies between component selection and certification. The authors also provide a framework for component selection [18] as well as an selection process [19]. The approach proposed in this paper focuses solely on the extra-functional aspect performance and could later on be integrated in the more general framework of Alvaro et al.

Bøegh describes in [20] a formalized approach to state component properties and ensure trust in them by third-party certification whilst considering a multi-certification-standards scenario. The approach presented in this paper can be used as measure to ensure trust in quality claims for the performance of components of business information software.

In contrast to Bøegh’s third-party approach, Morris proposes in [21] an approach for self-certification. It is designed to ease certification of functional aspects for open-source or free software. He developed a generic model to express test data. Published instances of these model and the software itself allow to verify quality claims by interested parties. The approach presented in this paper differs as it focuses on the extra-functional property performance instead of functionality and there is no need to publicize the software itself, nor does it depend on a preselected set of test cases.

B. Performance Testing

The performance testing market has been growing steadily [22] and hence there is a number of commercial and non-commercial performance testing tools available. The tools are presented in the following paragraphs. In contrast to the approach presented in this paper, the test cases run by the shown tools must be specified manually. In the approach of this paper, the information in the specification and implementation is used to deduce test cases for validation.

1) HP LoadRunner software: This commercial software is part of the Performance Center from HP and generates load, measures the performance, and helps to identify problems within a system. A more detailed overview is provided at [23]. It is designed to stress test an application from end-to-end and point out scalability issues. It provides support for diagnostic probes at code-level, non-intrusive real-time monitoring at system-level, and the inspection of SQL statements.

The approach presented in this paper uses probes on the code-level to measure the runtime of component internal processing sections as exact as possible. Hence, specialized probes have to be used for measurements.

2) LISA: The commercial LISA suite from iTKO is available at [24]. It consists of three tools: LISA Test [25] for designing and executing tests at UI-level and below, LISA Validate [26] for functional and performance monitoring, and LISA Virtualize [27] for behavior simulation of dependent services. The suite can run stand-alone as well as integrated with JUnit. The advantages of such a combined approach to end-to-end functional, load, and production testing are pointed out in [28]. LISA Virtualize provides the concept of a virtualized service as described in [29]:

Service virtualization involves the imaging of software service behavior and the modeling of a virtual service to stand in for the actual service during development and testing. With a virtual service, you image the behavior of a particular service, you construct the virtual service from that behavior, and then you deploy it to a virtual service environment.

The approach presented in this paper is related to LISA Test and LISA Validate. It is related to the former as it needs to automatically generate and execute tests on implementations. The later one is interesting as the behavior of required components or services has to be simulated in order to check the specifications.

3) JMeter: Apache JMeter is part of the Apache Jakarta Project and available at [30]. It is a java-based tool designed to load test functional behavior and measure performance for various server types. Supported server types are Web, SOAP,
JDBC, LDAP, JMS, and Email. It supports caching and offline analysis/replaying of test results.

Its testing strengths lie in heavy concurrent load conditions. The approach presented in this paper focuses on specification validation in single-user cases as it can rely on PCM’s validated prediction approach to scale correctly in high concurrency situations.

4) OpenSTA: The Open System Testing Architecture (OpenSTA) is available at [31]. The current toolset has the capability of performing scripted HTTP and HTTPS load tests with performance measurements from Win32 platforms. Testing is performed using the record and replay metaphor common in most other similar and commercially available toolsets. Data collections include scripted timers, SNMP data, Windows Performance Monitor stats, and HTTP results & timings.

The data collection and capturing possibilities are of interest for the approach presented in this paper. However, capture and replay methods are of minor interest, as the specification validation should not depend on this kind of functional test data.

5) PushToTest: PushToTest is a commercial open source alternative for testing and monitoring and available at [32]. It provides capabilities for functional testing, load testing, and monitoring.

The approach presented in this paper could use integrated open-source tools, like Glassbox for monitoring implementations.

C. Performance Specification Validation

Pavlopoulou and Young examined residual test coverage in [33]. The program statements not covered by previous testing approaches are instrumented to see if they are actually used or if the assumption that they are seldom used in practice holds.

The approach presented in this paper can use statement coverage to determine areas which might influence performance but have not been measured before.

The effort of testing and reusing components was addressed by Weyuker in [34]. She states that high reliability and availability requirements lead to enormous costs. Additionally, components have to be tested in isolation and after integration so savings of components-of-the-shelf are not sure. Reusing components requires retesting for stability, reliability, stress, and performance testing.

The approach presented in this paper will allow testing components for specified ranges of parameters and environments. As long as the components are used within these boundaries predicting the expected behavior wrt performance should require only very low effort.

Extra-functional behavior and component testing is also considered in [35] by Hamlet. The article is about a compositional testing theory based on subdomain or partition testing. Component test points and their (input and output) propagation are considered to identify the best test criteria or cases. Focus is put on functional behavior but extra-functional behavior is covered as well. Models are used to abstract the data flow and allow deriving test cases. Resulting from the type of modeling data flow, the approach has difficulties for example in finding fix points for loops / iterations. Additionally, the article points out that theoretical comparisons between random and subdomain testing have not shown a conclusive advantage either way in detecting failures.

In contrast to the approach presented in this paper, Hamlet focuses on functional testing which is also required for the kind of extra-functional testing described in his article.

VII. Conclusion

In this paper, we showed links to existing performance prediction approaches and their performance specifications. We gave the reason why the RDSEFF specification was chosen for this approach and introduced the specification itself. Additionally, we identified and explained which information is necessary to validate performance specifications. We presented our approach to certify specifications against implementations and explained the process of assessment and certification as well as listed current limitations and assumptions. A small lab example demonstrated the applicability of the process.

The presented approach aids companies in offering components in marketplaces. The publication of certified performance specifications in a marketplace is sufficient for potential customers to reliably evaluate and select components. However, the interests and intellectual properties of the offering companies are still protected as the specifications only contain a highly abstract view on the component’s behavior and keep the disclosure of details on the used algorithms and techniques to a minimum. Having certified performance specifications additionally supports software engineers in late composition of components. The software engineers gain the knowledge if the performance specifications fulfill their requirements for the intended composition which can in turn ease the evaluation of components and reduce the necessary effort.

The certification of specifications also supports a predictable assembly of components. The information contained in these specifications allows increasing confidence in the results produced by validated performance prediction approaches or identifying potential risks. Last but not least, certification by independent authorities provides a mean for quality assurance. If the development of components is given to a contractor the stipulation of certification enables the contracting body to trust the performance of a developed component beyond a few test cases while keeping its own quality assurance effort low. The contracting body can focus on its own expertise and does not need to employ performance engineers just for quality assurance.

As a next step the design of a model describing the validity of performance specifications is planned. It is also planned to extend the approach to allow the comparison and validation of parameterized distribution functions including predictions on the necessary effort in terms of test cases and runs for the requested validation. Furthermore, support for validating dependencies to input parameters of a component’s service is planned as well. In the medium term validating the specified
against the implemented required interface and the specified and implemented external call sequence is planned. In the long term the validation should also include return value dependencies from external calls.

ACKNOWLEDGEMENTS

Support for this work has been provided by the German Federal Ministry of Education and Research (BMBF), grant No. 01BS0822. The author is thankful for this support.

REFERENCES


