KlaperSuite: an Integrated Model-Driven Environment for Non-Functional Requirements Analysis of Component-Based Systems

Andrea Ciancone[1], Antonio Filieri[1], Luigi Drago[1], Raffaela Mirandola[1], and Vincenzo Grassi[2]

[1] Politecnico di Milano, Piazza Leonardo Da Vinci 32, 20133 Milano, Italy
andrea.ciancone@mail.polimi.it, {filieri,drago,mirandola}@elet.polimi.it
[2] Università di Roma "Tor Vergata", Viale del Politecnico 1, 00133 Roma, Italy
vgrassi@info.uniroma2.it

Abstract. Automatic prediction tools play a key role in enabling the application of non-functional requirements analysis to selection and assembly of components for Component-Based Systems, reducing the need for strong mathematical skills to software designers. Exploiting the paradigm of Model Driven Engineering (MDE), it is possible to automate transformations from design models to analytical models, enabling for formal property verification. MDE is the core paradigm of KlaperSuite presented in this paper, which exploits the KLAPER pivot language to fill the gap between Design and Analysis of Component-Based Systems for reliability and performance properties. KlaperSuite is a family of tools empowering designers with the ability to capture and analyze QoS views of their systems by building a one-click bridge towards a number of established verification instruments.

1 Introduction

Discovering late during the development process that a software system does not meet certain non-functional requirements can be harmful. The impact of changes — if applied when a complete implementation of a system exists — on development costs and on failure risks may be non negligible. Indeed, it has been already pointed out that anticipating the analysis of non-functional properties — such as performance and reliability — at design time can mitigate these issues [1–3]. The work we present in this paper goes in this direction, by supporting early analysis of non-functional attributes for software systems built with a Component-Based (CB) development paradigm.

Component-based software systems are essentially assemblies of preexisting, independently developed components. The focus of the development process shifts from custom design and implementation to selection, composition and coordination [4, 5]. In a component based setting, analysis must be carried out before assembling and this can lead to early discovery potential problems related

to non-functional attributes. Which components are selected, how they are composed, and how they are coordinated should in turn depend on the results of analyses, as pointed out by Crnkovic in [4].

However, existing techniques for non-functional analysis rely on very specific performance-related formalisms — such as Queueing Networks (QNs), Petri Nets (PNs), or Markovian models — but software systems are rarely represented in these terms. Designers, who usually lack sufficient experience in performance engineering, prefer *design-oriented* models such as UML [6]. To cope with this mismatch, tools have been recently proposed in literature. The idea is leverage Model Driven Engineering (MDE) [7] techniques to automatically derive, by means of *model transformations*, performance models from design-oriented models of the system (augmented with additional information related to the non-functional attributes of interest). Existing analysis methodologies [2, 8–10] may be in turn applied as is.

However, defining this kind of transformations could be quite difficult. The large semantic gap between the source and the target meta-models of the transformation, the heterogeneous design notations that could be used by different component providers, and the different target analysis formalisms are all examples of barriers for transformations development. The usage of intermediate modeling languages, which capture relevant information for QoS analyses, has been proposed to mitigate these problems. Intermediate languages in fact bridge design-oriented and analysis-oriented notations, and help in distilling the information needed by performance analysis tools [11, 12, 9]. Instead of directly transforming design models to performance models, a two-step transformation from the source model to the intermediate model, and from the intermediate model to the target model is proposed.

In this paper we describe KlaperSuite, an integrated environment for the performance and reliability analysis leveraging KLAPER (Kernel LAnguage for PErformance and Reliability analysis) [11]. KLAPER is an intermediate language supporting the generation of stochastic models, to predict performance and reliability, from design-level models of component-based software systems. Transformations from design models to analytical models are completely automated in a one-click way. Designers are indeed empowered with the ability to analyze their systems, with established verification instruments, in a seamless and integrated environment.

The remainder of this paper is organized as follows. Section 2 outlines the KLAPER language and its main characteristics. In section 3 we present the different types of analysis included in KlaperSuite, spanning from reliability, to performance, and to generation of simulation prototypes. Sections 5 and 6 describe existing literature related to our work and future research directions, respectively.

2 KLAPER

In this section we first present the key points of our MDE-based approach to the generation of a performance/reliability model for a CB system (Section 2.1). Then we present the meta-model of the intermediate language that we use to support this approach (Section 2.2).

2.1 The basic methodology

The central element of our framework is the usage of KLAPER [11] whose goal is to split the complex task of deriving an analysis model (e.g. a queueing network) from a high level design model (expressed using UML or other component-oriented notations) into two separate and presumably simpler tasks:

- extracting from the design model only the information that is relevant for the analysis of some QoS attribute and expressing it in terms of the key concepts provided by the intermediate language;
- generating an analysis model based on the information expressed in the intermediate language.

These two tasks may be solved independently of each other. Moreover, as a positive side effect of this two-step approach, we mitigate the *"n-by-m"* problem of translating n heterogeneous design notations (that could be used by different component providers) into m analysis notations (that support different kinds of analysis), reducing it to a less complex task of defining $n + m$ transformations: n from different design notations to the intermediate language, and m from it to different analysis notations.

The KLAPER goal is to capture in a lightweight and compact model only the relevant information for the stochastic performance and reliability analysis of CB systems, while abstracting away irrelevant details.

To integrate this kernel language into an MDE framework, leveraging the current state of the art in the field of model transformation methodologies, KLAPER is defined as a Meta-Object Facility (MOF) meta-model [13]. According to MOF, a (meta)model is basically a constrained directed labeled graph, and a meta-model defines the syntactic and semantic rules to build legal models.

Hence, we can use the MOF facilities to devise transformations to/from KLAPER models, provided that a MOF meta-model exists for the corresponding source/target model. According to the MDE perspective, these transformations can be defined as a set of rules that map elements of the source meta-model onto elements of the target meta-model.

2.2 The KLAPER meta-model

Figure 1 shows the structure of the KLAPER meta-model[11]. To support the distillation from the design models of a CB system of the relevant information for stochastic performance/reliability analysis, KLAPER is built around an abstract

representation of such a system, modeled (including the underlying platform) as an assembly of interacting *Resources*. Each Resource offers (and possibly requires) one or more *Services*. A KLAPER Resource is thus an abstract modeling concept that can be used to represent both software components and physical resources like processors and communication links.

A *scheduling policy* and a *multiplicity* (number of concurrent requests that can be served in parallel) can be associated with a resource to possibly model access control policies for the services offered by that resource. Each service offered by a resource is characterized by its *formal parameters* that can be instantiated with actual values by other resources requiring that service. We point out that both the formal parameters and their corresponding actual parameters are intended to represent a suitable abstraction (for the analysis purposes) of the real service parameters. For example, a real list parameter for some list processing software component could be abstractly represented as an integer valued random variable, where the integer value represents the list size, and its probability distribution provides information about the likelihood of different sizes in a given analysis scenario. We explicitly introduce service parameters to better support compositional and parametric analysis.

To bring performance/reliability related information within such an abstract model, each activity in the system is modeled as the execution of a *Step* that may take time to be completed, and/or may fail before its completion: the *internalExecTime*, *internalFailTime* and *internalFailProb* attributes of each step may be used to give a probabilistic characterization of these aspects of a step execution.

Steps are grouped in *Behaviors* (directed graphs of nodes) that may be associated either with the Services offered by Resources (reactive behavior), or with a *Workload* modeling the demand injected into the system by external entities like the system users (proactive behavior). *Control steps* can be used to regulate the flow of control from step to step, according to a probabilistic setting.

A *ServiceCall* step is a special kind of Step that models the relationship between required and offered services. Each ServiceCall specifies the name of the requested service and the type of resource that should provide it.

The relationship between a ServiceCall and the actual recipient of the call is represented separately by means of instances of the *Binding* metaclass. This allows a clear separation between the models of the components (by means of Resources/Services) and the model of their composition. In fact a set of bindings can be regarded as a self-contained specification of an assembly. Similarly, since the service call concept is also used at the KLAPER level to model the access of software components to platform level services, a suitable set of bindings can model as well the deployment of the application components on the underlying platform.

Finally, we point out that the performance/reliability attributes associated with a behavior step concern only the *internal* characteristics of the behavior; they do not take into account possible delays or failures caused by the use of other required services, that are needed to complete that step. In this respect,

we remark that when we build a KLAPER model (first task outlined above) our goal is mainly "descriptive". The Bindings included in the model help to identify which external services may cause additional delays or failure possibilities. How to properly mix this "external" information with the internal information to get an overall picture of the service performance or reliability must be solved during the generation and solution of an analysis model derived from a KLAPER model.

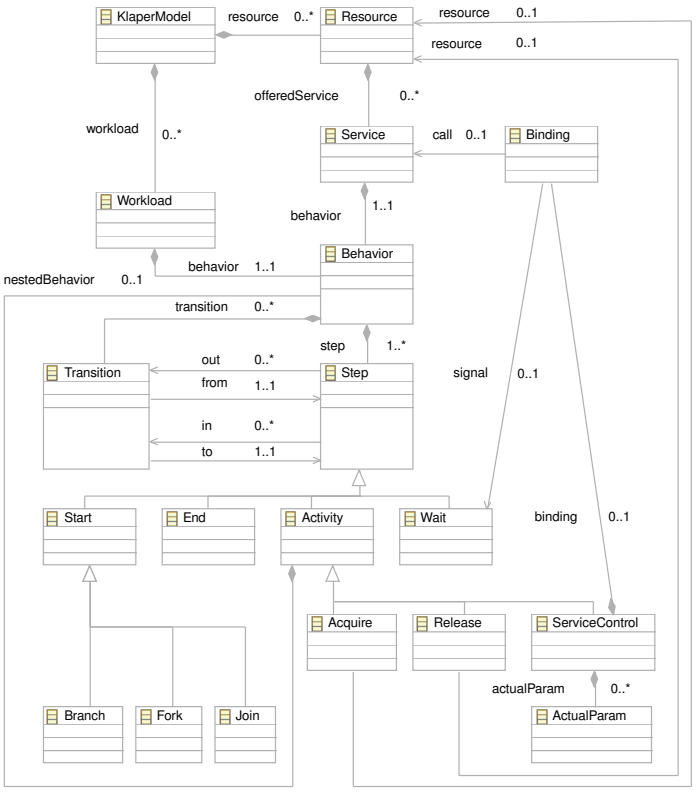

Fig. 1. The KLAPER MOF meta-model

3 The KlaperSuite Analysis Tools

The main purpose of the KLAPER-based analysis is to provide a set of tools that support early verification of non-functional requirements. Such an analysis, applied at early stages of design, allows identifying possible issues while the development team has the largest decision scope.

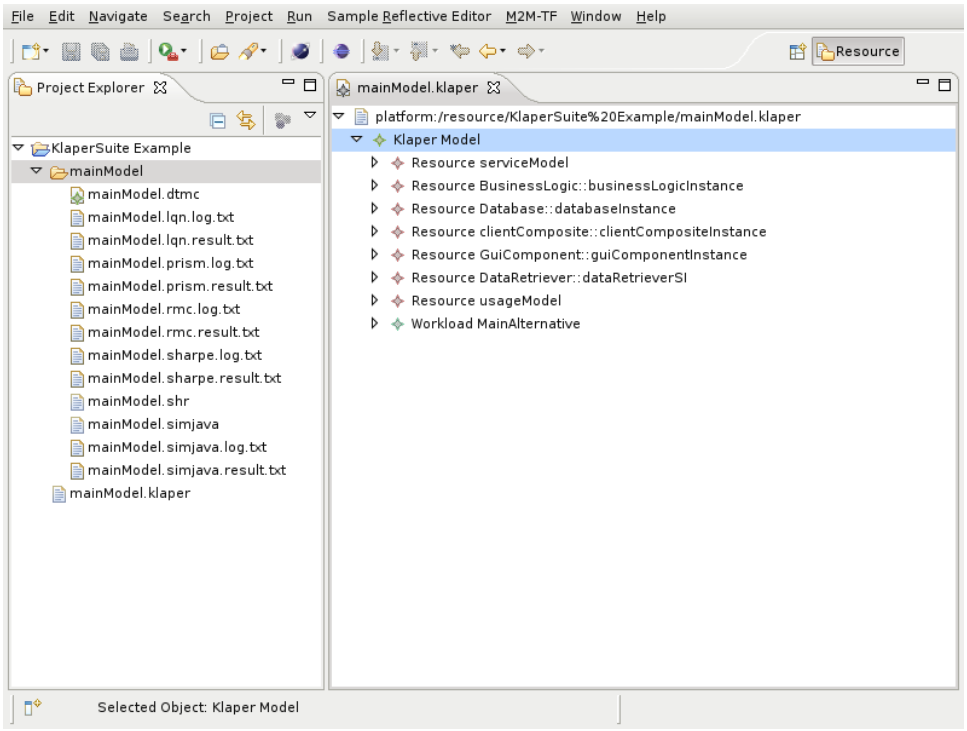

Fig. 2. An example of a KlaperSuite workspace

KlaperSuite aims at providing a family of KLAPER-based tools which can execute a number of analysis tasks on KLAPER models. All the tools in the KlaperSuite are fully automated and require at most a few configuration parameters to be set. The entire environment is integrated in the Eclipse IDE [14], in order to provide a unified interface and a familiar environment for both academic and industrial developers.

Most of the tools are able to automatically transform KLAPER models into appropriate inputs for each of the external tools involved in the analysis process, and then capture analysis results and store them in text files, which are human readable. It is then easy to extend the suite by adding specific parsers in order to put back results into any computer readable form.

The KlaperSuite's purpose is to fill the gap from KLAPER to non-functional verification tools. A consolidated development process may possibly benefit from the implementation of automatic model transformations from already established design metamodels to KLAPER. This single time investment can enable designers to take advantage of the entire family of analysis tools.

Analysis plugins can also store intermediate files (i.e. third parties tools input files) that can be further analyzed for different purposes or by external experts.

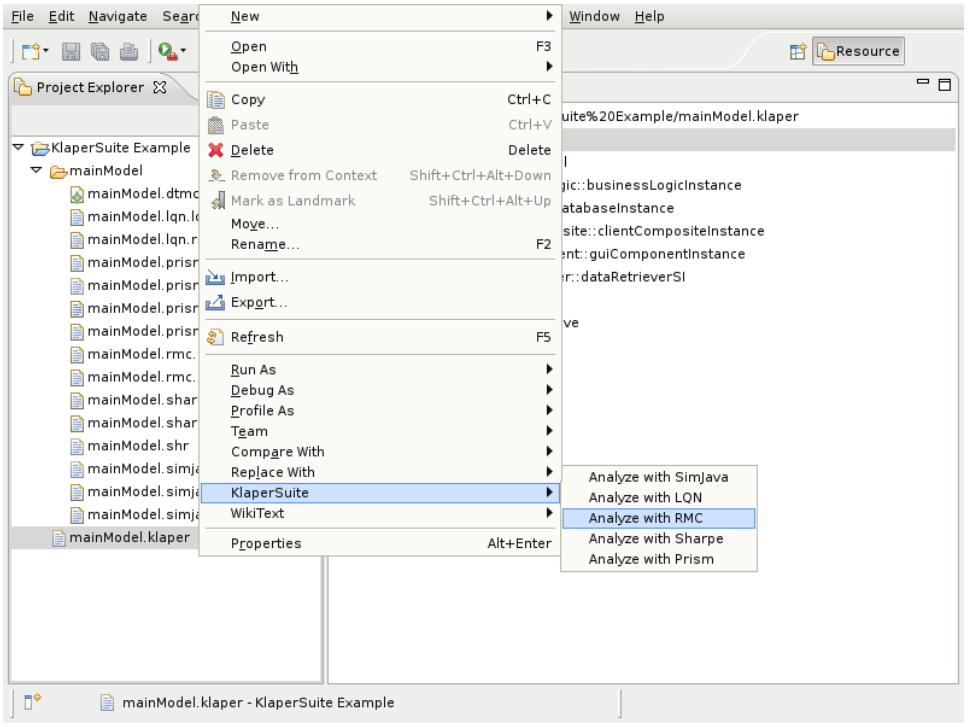

Fig. 3. Menu for the launches of the KlaperSuite tools

Download instructions for the KlaperSuite can be found at `http://home.dei.polimi.it/filieri/klapersuite`. In the same location is also available an example workspace, which has not been described in this paper because of the lack of space.

In the following of this section we present the set of verification features currently supported by KlaperSuite and illustrated in Fig. 4. They will be grouped in three subsets depending on the purpose of their inclusion. More specifically, Section 3.1 will present analysis features concerning reliability estimation, Section 3.2 concerns performance prediction, while Section 3.3 will present a simulation-based analysis tool which provides verification of both reliability and performance properties, as well as a lightweight prototype of the system to be.

3.1 Reliability

Reliability is the first non functional aspect we focus on. There are a number of tools that allow the evaluation of various facets of reliability [15]. A KLAPER model can be automatically mapped in a Markov Chain, that is a stochastic characterization of the system under design able to capture various information affecting software reliability.

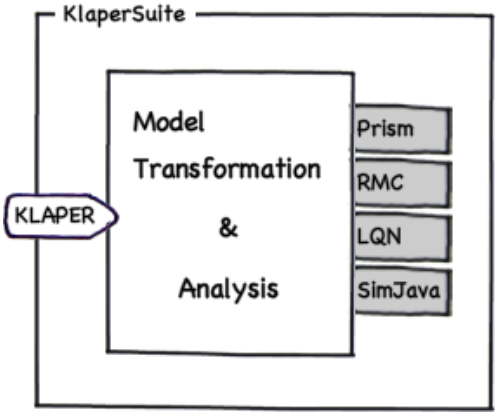

Fig. 4. High level view of KlaperSuite

Reliability is one of the so-called *user-centered* property [16], in the sense that the reliability of a system strictly depends on its usage. While a failure probability is associated with each system component, the actual usage determines which parts of the systems are more stressed by clients and thus can have an higher perceived impact. The usage profile of a system is embedded in two parts of a KLAPER model, namely *workload* and *branch annotations*. Workload is directly related to the intended usage of the system by its clients, that is, which functionalities they invoke. Branch probabilities are instead more related to the distribution of inputs inserted by the clients.

PRISM Model-Checking. Mapping a KLAPER model into a corresponding *Discrete Time Markov Chain* (DTMC) is straightforward. A DTMC can be roughly seen as finite state-transition automata where each state s_i has a certain probability p_{ij} to reach state s_j. As for Probability theory, for each state s_i it holds that $\sum_j p_{ij} = 1$. States of a DTMC are used to represent relevant states of the execution of a software system. For example a state may represent an internal action or the invocation of a service. In DTMC-based reliability analysis, it is common to enhance the model of the system with a set of states that represents meta-condition of the execution, that is, they do not correspond to neither internal actions nor external invocations, but rather to failures or success. These *meta-states* are typically related to permanent conditions of the system, and thus their counterpart in the domain of DTMCs are *absorbing states*. Any state s_i such that $p_{ii} = 1$ is said to be absorbing, with the immediate meaning that state s_i, once reached, cannot be left.

Reliability can be defined as the probability of reaching any absorbing state corresponding to a success condition from the state corresponding to the execution's start. But a designer may be interested also in more complex properties related to reliability, such as the probability that the system fails given that it reached a certain execution state, or that a certain kind of failure arises. In order

to specify those properties, given a DTMC model of the system under design, it is possible to use special purpose logic languages, such as PCTL [17] and its extension, PCTL* [18]. Such logics allow the formal description of a set of paths through a DTMC. Then a Probabilistic Model-Checker is able to compute the probability for the execution to follow exactly those paths.

For example, assuming that there is a single absorbing success state s_s, we are interested in considering all the possible paths which will eventually reach s_s. Such a path property can be easily formalized in PCTL(*) as $\diamond s = s_s$, which literally means that eventually ($\diamond$) the current state of execution (s) will be equal to the success state (s_s). The eventually operator assumes that the execution of the system always begin from its defined initial state (which corresponds to a *start* step of a KLAPER workflow) and that can reach s_s in any finite number of state transitions.

This preamble is to justify the idea to include in our suite a transformation toward a DTMC+PCTL* model. In order to be able to exploit available model checkers, transformation must finally provide input files for one of them. The two mostly established are PRISM [19, 20] and MRMC [21]. The former exploits symbolic manipulation of PCTL properties in order to verify them on a compact representation of the state space; so it might be beneficial in case of complex formulae. The latter uses an explicit state-space representation that makes it possibly require more memory, but makes the verification quite fast, at least for simple formulae such as reachability. The reader interested in more details about Probabilistic Model Checking could refer to [22].

KlaperSuite is able to automatically transform a KLAPER model into a PRISM input, that is, a DTMC and a PCTL represented in PRISM's textual syntax. Our tool is able to extract the global reliability and to put it in a text file. But the produced PRISM models are completely consistent with all the information in the KLAPER source, and can thus be further analyzed by means of the other PRISM's advanced features [20], viable also through its graphical interfaces. Also PRISM can itself convert models and properties in MRMC's syntax, thus enabling a second way of analysis.

The transformation from KLAPER to PRISM is realized in two steps. The first is a model-to-model transformation from KLAPER to an intermediate meta-model which reproduce the structure of a PRISM model. This transformation is implemented in QVT-Operational, the imperative model-to-model transformation language standardized by the OMG [23]. The second step is a model-to-text transformation implemented in Xpand2 [24], that generates the textual representation of the PRISM model to be analyzed. Both QVTO and Xpand2 are natively supported by Eclipse.

The most critical issue in analyzing KLAPER models for reliability through PRISM is that KLAPER model supports the specification of (possibly recursive) function calls. Such a feature is not naturally captured by Discrete Time Markov Chains, which are instead successfully adopted in many research works [25]. The reason is that software's control flow is hard to be flattened in a finite sequence of function calls without loosing precision (remember that highly reliable software

may require estimation's precision up to 10^{-7}). In order to properly analyze our models through PRISM, we need to enhance DTMC models with some Process Algebra constructs in order to stochastically simulate function calls. This formalization allows PRISM to obtain results with arbitrary accuracy. By default KlaperSuite requires results with maximum error magnitude of 10^{-12}. This value can be increased or decreased at will.

The problem with the combination of DTMCs and Process Algebra lies in the exponential state-space explosion. Hence even small KLAPER models can lead to untreatable PRISM analyses, in presence of recursive invocations. This issue introduce the need for a more efficient way to deal with recursiveness, namely Recursive Markov Chains, that will be presented in the next section.

Recursive Markov Chains. A Recursive Markov Chain can be seen as a collection of finite-state Markov Chains with the ability to invoke each other, possibly in a recursive way. They have been introduced in [26] and come with a strong mathematical support. RMCs can be analyzed (by means of equation systems) in a very efficient way in order to evaluate reachability properties. Reliability, intended as the probability of successfully accomplish the assigned task, as well as the probability of failure given that the execution has reached a certain execution state, can be formalized as reachability properties, as well as a number of other interesting requirements.

Also, by construction KLAPER behaviors are *1-exit* control flows, that is they only have a single *end* step. This allow us to verify any reachability property in P-time. In practice RMC analysis of KLAPER models has been successfully applied in the european project Q-Impress [27] and proved to be really efficient on real-world industrial models.

The first step of the transformation from KLAPER to RMC is the same model-to-model transformation used for the PRISM based analysis. From the intermediate PRISM-tailored model, KlaperSuite extrapolates a system of equations that is directly solved by our Java implementation, without any need for external tools.

Reliability estimation is then reported in the result file, while an extensive log file contains a textual representation of the equations system and the complete solution, that is, the probability, from each modeled execution state to reach the successful completion of the execution.

With respect to PRISM, RMC-based analysis can handle very large models with recursive invocations. On the other hand it does only support verification of reachability properties over Markov chains. The accuracy of results is arbitrary also for RMC analysis and set by default to 10^{-12}.

3.2 Performance

Early evaluation of performance can be obtained by either analytical modeling or simulation. In this Section we focus on modeling, while in Section 3.3 we will briefly discuss simulation facilities of the KlaperSuite.

The two most basic, though general-purpose, measurable properties for performance are response-time and throughput. One of the most widely accepted mathematical models to estimate those properties are Layered Queuing Networks (LQNs) [9, 28]. LQNs introduce new modeling concepts to represent software systems. Systems are represented as a layered hierarchy of LQN tasks (each one corresponding to a KLAPER *Resource*) which interact, and generate demands for underlying physical resources. Each task may offer different kinds of services, called *entries*. An entry corresponds to a KLAPER service and can be described either as a *phase* or as an *activity*. Phases allow for description of simple sequential behaviors; activities allow for description of more complex behaviors, e.g., with control structures such as forks, joins, and loops.

An LQN model can be analyzed by means of special purpose mathematical softwares. In the KlaperSuite we make use of the LQN Solver from Carleton University[3]. In order to produce input files for that solver we designed a two step model transformation. The first step is a QVTO model-to-model transformation from KLAPER to an intermediate meta-model which is an abstract representation of the analytical model. Then, the abstract representation is transformed into an input file for the LQN solver by means of an Xpand model-to-text transformation.

The obtained LQN models can then be solved. Examples of the kind of analysis results that can be derived applying the LQN solver to the obtained LQN model are task utilization, throughput and response time. Different configurations can be easily analyzed by a simple change in the LQN parameters. The analysis of the obtained performance results can help in the identification of critical components, such as bottlenecks, which can prevent the fulfillment of performance requirements.

3.3 Simulation

The simulation engine of the KlaperSuite was initially designed with the only purpose to validate the previous analysis tools, but can be used to simulate any KLAPER model, tough it was not designed to deal with scalability issues.

The simulator is based on the SimJava library for the simulation of discrete event systems[4]. Upon SimJava, KlaperSuite builds a lightweight prototype of the system in which each service is simulated through a SimJava Entity. Each entity runs in its own thread and is connected to the others by *ports* which allow communications consisting of sending and receiving events. Communications among entities are defined consistently with the corresponding KLAPER behavior to be simulated. A central control thread monitors the execution of the prototype and records execution times and failure occurrences of each Entity, as they are inferred from the trace of events. The control thread's log is then analyzed in order to derive statistical estimation for performance and reliability properties.

[3] `http://www.layeredqueues.org`
[4] `http://www.dcs.ed.ac.uk/home/hase/simjava/`

In order to produce the Java code of the prototype another two steps model transformation is in place. The first step transforms the KLAPER model into an intermediate meta-model corresponding to the structure of the prototype[5] and is implemented in QVTO. The second step is a model-to-text transformation implemented in Xpand which generates the Java code.

The previous tools have been validated through simulation [29]. Notice that for intrinsic reasons simulation is computationally expensive with respect to mathematical analysis to verify the set of reliability and performance properties discussed in this paper. Nevertheless, the use of an established tool such as SimJava allows for further enhancement of the Java prototypes, that can, for example, be instrumented with a larger set of monitors or with special purpose features.

4 Tools Integration Status

Table 1 shows the current development status of the KlaperSuite. Some of the tools have been developed in the past as standalone and their integration is still ongoing. All the single tools are hosted on Sourceforge[6].

5 Related work

In the last years, it has been widely recognized the need of including early quality prediction in the software development process. In particular, there has been an increasing interest in model transformation methodologies for the generation of analysis-oriented target models (including performance and reliability models) starting from design-oriented source models, possibly augmented with suitable annotations. Several proposals have been presented concerning the direct generation of performance analysis models. Each of these proposals focuses on a particular type of source design-oriented model and a particular type of target analysis-oriented model, with the former spanning UML, Message Sequence Chart, Use Case Maps, formal or ADL languages and the latter spanning Petri nets, queueing networks, layered queueing network, stochastic process algebras, Markov processes (see [2, 3] for overviews of these proposals and the WOSP conference series [10] for recent proposals on this topic).

Some proposals have also been presented for the generation of reliability models. All the proposals we are aware of start from UML models with proper annotations, and generate reliability models such as fault trees, state diagrams, Markov processes, hazard analysis techniques and Bayesian models (see [30, 31] for a recent update on this topic).

The use of bridge models expressed in some suitable intermediate language has also been proposed in the literature to support the generation of analysis

[5] The meta-model can be found in the Sourceforge repository.

[6] `http://sourceforge.net/projects/klaper/`

Tool	Purpose	Features	Integration Status
Klaper2Prism	Reliability	– System mapped to DTMC model. – Reliability properties expressed in PCTL*. – Efficient on complex formulae. – Does not scale on recursive service invocations.	Fully integrated
Klaper2RMC	Reliability	– System mapped to RMC model. – State reachability properties only. – Efficient for recursive service invocations. – Highly scalable on large systems.	Fully integrated
Klaper2LQN	Performance	– System mapped to LQN model. – Response time, throughput, state residence time. – Does not scale on large systems	Still standalone
Klaper2 SimJava	Simulation	– System mapped to SimJava application. – Reliability and performance estimation. – Extensible via SimJava features.	Partially integrated

Table 1. Tools integration status.

oriented models from design oriented models [11, 32] [7]. A MOF compliant kernel language specifically related to performance analysis has been proposed in [32] with the goal of specifying intermediate models between design-oriented UML models with performance annotations and performance models. The transformations from UML to an intermediate model and from it to different performance models (spanning layered queueing networks, extended queueing networks and stochastic Petri nets) are also briefly outlined in [12, 9, 32].

With respect to the kernel language of [12, 9, 32] KLAPER is intended to serve as intermediate language also for reliability and, possibly joint performance-reliability (performability) analysis, starting from heterogeneous design notations, and is specifically targeted to component-based systems. It has been ap-

[7] In a different context from non-functional requirement analysis of component-based systems one of the first proposal can be found in [33], where the kernel language is called a "pivot metamodel".

plied for the analysis of performance and reliability using queuing networks and Markov models [11] and for performance analysis of the CoCoME case study [34, 35] through layered queueing networks. Extensions of KLAPER has also been proposed for the analysis of self-adaptive [36] and reactive [37] systems. In these works the application of KLAPER has been mainly manual and only some steps were automated.

Recently, KLAPER has been used within the European project Q-ImPrESS [27]. Q-ImPrESS aims at building a framework for service orientation of critical systems. Such a framework is deeply founded on model transformations, which allow to automatically fill the gap between design and analysis models. Hence KLAPER facilities have been exploited in the definition of the reliability tool in the Q-ImPrESS tool chain.

With respect to previous works, we presented in this paper KlaperSuite a fully automated and integrated environment including a family of tools empowering designers with the ability to capture and analyze the performance and reliability figures of their systems. The possibility of using different verification tools together with a simulation-based analysis tool makes KlaperSuite a unique instrument for predicting the software qualities during the development process.

6 Conclusions

In this paper we presented KlaperSuite, an integrated environment for performance and reliability analysis leveraging KLAPER intermediate language. KlaperSuite allows the automatic generation of stochastic models to verify and predict performance and reliability properties of component-based software systems. Analyses can be applied on high level design models, in order to provide support for early properties evaluation. By using this tool, designers are empowered with the ability to analyze their systems, with established verification instruments, in a seamless and integrated environment.

As future extension of the KlaperSuite, we are planning to implement model transformations from higher level design models (first of all UML) to KLAPER. In this way KlaperSuite will be easier to integrate in established development cycles. On a longer perspective, we also plan to explore the possibility of extracting KLAPER model directly from annotated code, which will encourage the use of analytical models by programmers.

Finally, only part of the tools have been evaluate on real industrial models inside the Q-Impress project. We are currently working on the experimentation of this environment on different testbeds, to assess its effectiveness through a more comprehensive set of real experiments.

Acknowledgments

The authors would like to thank all the persons who have worked on the tools belonging to KlaperSuite, in particular Federico Carbonetti and Enrico Ran-

dazzo who implemented LQN and SimJava transformations. The work has been partially supported by the EU project Q-ImPrESS (FP7 215013).

References

1. Smith, C.U., Williams, L.G.: Performance and Scalability of Distributed Software Architectures: an SPE Approach. Addison Wesley (2002)
2. Balsamo, S., DiMarco, A., Inverardi, P., Simeoni, M.: Model-based performance prediction in software development: A survey. IEEE Transactions on Software Engineering **30**(5) (2004) 295–310
3. Koziolek, H.: Performance evaluation of component-based software systems: A survey. Perform. Eval. **67**(8) (2010) 634–658
4. Crnkovic, I.: Building Reliable Component-Based Software Systems. Artech House, Inc., Norwood, MA, USA (2002)
5. Szyperski, C.: Component Software: Beyond Object-Oriented Programming. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA (2002)
6. Object Management Group: UML 2.0 superstructure specification (2002)
7. Atkinson, C., Kühne, T.: Model-driven development: A metamodeling foundation. IEEE Softw. **20**(5) (2003) 36–41
8. Becker, S., Koziolek, H., Reussner, R.: The palladio component model for model-driven performance prediction. Journal of Systems and Software **82**(1) (2009) 3–22
9. Woodside, M., Petriu, D.C., Petriu, D.B., Shen, H., Israr, T., Merseguer, J.: Performance by unified model analysis (puma). In: WOSP '05: Proceedings of the 5th international workshop on Software and performance, New York, NY, USA, ACM Press (2005) 1–12
10. : Wosp : Proceedings of the international workshop on software and performance (1998-2010)
11. Grassi, V., Mirandola, R., Sabetta, A.: Filling the gap between design and performance/reliability models of component-based systems: A model-driven approach. J. Syst. Softw. **80**(4) (2007) 528–558
12. Gu, G.P., Petriu, D.C.: From uml to lqn by xml algebra-based model transformations. In: WOSP '05: Proceedings of the 5th international workshop on Software and performance, New York, NY, USA, ACM Press (2005) 99–110
13. Object Management Group: MOF version 2.0, ptc/04-10-15 (2004)
14. Foundation, T.E.: Project website. `http://www.eclipse.org` (2010)
15. Horgan, J., Mathur, A.: Software testing and reliability. The Handbook of Software Reliability Engineering (1996) 531–565
16. Cheung, R.C.: A user-oriented software reliability model. IEEE Trans. Softw. Eng. **6**(2) (1980) 118–125
17. Hansson, H., Jonsson, B.: A logic for reasoning about time and reliability. Formal Aspects of Computing **6** (1994) 512–535 10.1007/BF01211866.
18. Reynolds, M.: An axiomatization of pctl*. Information and Computation **201**(1) (2005) 72 – 119
19. Hinton, A., Kwiatkowska, M., Norman, G., Parker, D.: Prism: A tool for automatic verification of probabilistic systems. Proc. 12th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS) **3920** (2006) 441–444

20. Kwiatkowska, M., Norman, G., Parker, D.: Prism 2.0: a tool for probabilistic model checking. Quantitative Evaluation of Systems, 2004. QEST 2004. Proceedings. First International Conference on the (2004) 322–323
21. Katoen, J.P., Khattri, M., Zapreevt, I.: A markov reward model checker. In: Quantitative Evaluation of Systems, 2005. Second International Conference on the. (2005) 243 – 244
22. Baier, C., Katoen, J., et al.: Principles of model checking. (2008)
23. Group, O.M.: Qvt 1.0 specification. (`http://www.omg.org/spec/QVT/1.0/`)
24. Efftinge, S., Kadura, C.: Xpand language reference (2006)
25. Goseva-Popstojanova, K., Trivedi, K.S.: Architecture-based approach to reliability assessment of software systems. Performance Evaluation **45**(2-3) (2001) 179 – 204
26. Etessami, K., Yannakakis, M.: Recursive markov chains, stochastic grammars, and monotone systems of nonlinear equations. In: STACS. (2005) 340–352
27. Consortium, Q.I.: Project website. (`http://www.q-impress.eu`)
28. Rolia, J.A., Sevcik, K.C.: The method of layers. IEEE Transactions on Software Engineering **21**(8) (1995) 689–700
29. Randazzo, E.: A Model-Based Approach to Performance and Reliability Prediction. PhD thesis, Universitá degli Studi di Roma - Tor Vergata (2010)
30. Immonen, A., Niemelä, E.: Survey of reliability and availability prediction methods from the viewpoint of software architecture. Software and System Modeling **7**(1) (2008) 49–65
31. Bernardi, S., Merseguer, J., Petriu, D.C.: Adding dependability analysis capabilities to the MARTE profile. In: Model Driven Engineering Languages and Systems, 11th International Conference, MoDELS 2008, Toulouse, France, September 28 - October 3, 2008. Proceedings, MoDELS. Volume 5301 of Lecture Notes in Computer Science., Springer (2008) 736–750
32. Petriu, D.B., Woodside, C.M.: An intermediate metamodel with scenarios and resources for generating performance models from uml designs. Software and System Modeling **6**(2) (2007) 163–184
33. Jouault, F., Allilaire, F., Bézivin, J., Kurtev, I.: Atl: A model transformation tool. Sci. Comput. Program. **72**(1-2) (2008) 31–39
34. Rausch, A., Reussner, R., Mirandola, R., Plasil, F., eds.: The Common Component Modeling Example: Comparing Software Component Models [result from the Dagstuhl research seminar for CoCoME, August 1-3, 2007]. In Rausch, A., Reussner, R., Mirandola, R., Plasil, F., eds.: CoCoME. Volume 5153 of Lecture Notes in Computer Science., Springer (2008)
35. Grassi, V., Mirandola, R., Randazzo, E., Sabetta, A.: Klaper: An intermediate language for model-driven predictive analysis of performance and reliability. In: CoCoME. Volume 5153 of Lecture Notes in Computer Science., Springer (2007) 327–356
36. Grassi, V., Mirandola, R., Randazzo, E.: Model-driven assessment of qos-aware self-adaptation. In: Software Engineering for Self-Adaptive Systems. Volume 5525 of Lecture Notes in Computer Science. (2009) 201–222
37. Perez-Palacin, D., Mirandola, R., Merseguer, J., Grassi, V.: Qos-based model driven assessment of adaptive reactive systems. In: Proceedings of the 2010 Third International Conference on Software Testing, Verification, and Validation Workshops. ICSTW '10, Washington, DC, USA, IEEE Computer Society (2010) 299–308