Affidavit: Automated Reliability Prediction and Analysis of Software Architectures

João M. Franco, Francisco Correia, Raul Barbosa, and Mário Zenha-Rela

Faculty of Science and Technology
University of Coimbra, Portugal
{jmfranco,fcorreia,rbarbosa,mzrela}@dei.uc.pt
http://www.uc.pt/fctuc/dei/

Abstract. Quality attributes (*e.g.*, performance, reliability and security) are detailed in a system's architecture and determine the fitness of purpose and satisfaction of stakeholders regarding the final product. Although research has provided methods to assess different quality attributes, few checks are automatically performed. Manually checking a quality attribute from a large and complex architecture is a time-consuming and error-prone task. This paper addresses this issue by generating stochastic models that predict and analyze reliability from a software architecture description, in an automated way. In addition, our approach has been compiled into a tool targeting system architects, the Affidavit tool. This tool is accessible from an architecture development framework and provides information about structural issues and reliability bottlenecks of systems. As a result, Affidavit allows architects to reason about the designed architecture, helping to avoid architectural arrangements that might have a negative impact on the overall system reliability, at the same time that it indicates the most suitable arrangement for specific contexts. This paper describes the tool and its implementation details, demonstrating its capabilities on practical systems.

1 Introduction

Software architecture is a fundamental activity of software development, in which the documentation about the system's structure and properties at a high-level abstraction of the whole system is produced. As such, software architecture supports designer's decisions on the quality attributes expressed in the final product such as performance, maintainability, security and reliability. Their assessment is of utmost importance to serve as a guidance on the decisions to be taken, as well as, to identify issues and prevent additional costs on fixing late-detected problems. However, in current practice very few of these quality attributes are automatically checked, inducing time-consuming and error-prone tasks.

This paper presents a tool [1] that automatically assesses, predicts and performs a thorough analysis on the reliability of a software architecture described in the Acme Architectural Description Language (ADL) [2]. The Affidavit tool was developed as a plugin for AcmeStudio [3], aiming to provide automated reliability analysis [4] with an integrated development environment. AcmeStudio is an architecture development environment written as a plugin of the Eclipse RCP Framework [5]. It is a graphical architecture design tool that creates, modifies or iterates over architectural descriptions in the Acme language.

Affidavit has been integrated into AcmeStudio, which makes it close to the architecture iteration development cycle, being accessible and easy to use for architects. Thus, Affidavit provides valuable feedback on issues and changes of reliability that an architectural iteration has over another. Also, it guides users until the architecture is compliant with the client requirements and needs, regarding system reliability.

This paper is organized as follows. Section 2 discusses the adopted method and the complete set of features of the tool. Section 3 demonstrates the applicability of our approach through modeling, prediction and analysis of a software architecture. Section 4 presents the related work before Section 5 concludes the paper.

2 Affidavit Overview

The context of Affidavit is shown through the diagram presented in Figure 1. This diagram shows how AcmeStudio and Affidavit, both the front-end and the analysis plugin, interact with each other. Our tool acts as a plugin for the AcmeStudio using the modeled ADL to produce a stochastic model which exhibits the behavior of the system regarding its reliability. Thus, we provide means for architects to predict and analyze system reliability through two different methods: reliability prediction and sensitivity analysis (thoroughly described in Section 2.2). In each one of these methods, our tool will add the proper information about the simulation to the history view (*i.e.*, result, simulation description and

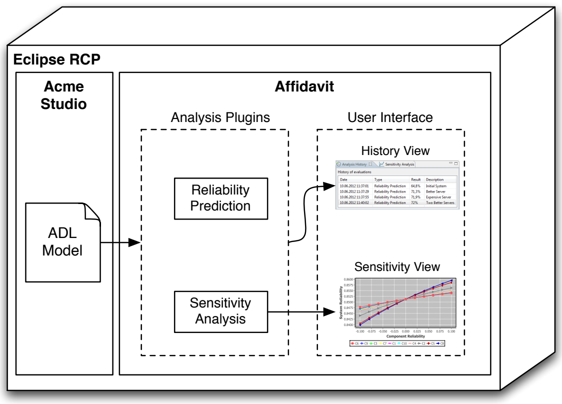


Fig. 1. Affidavit context diagram

type). From the history view, the architect has at his disposal the possibility to visualize the architecture used as input in each simulation. When a sensitivity analysis is performed, Affidavit will also present a graphical representation of the obtained results, as it is illustrated by the sensitivity view in Figure 1.

2.1 Required Annotations

Our approach requires that specific annotations are embedded in the architecture as properties for a faithful translation and a more accurate prediction. The specification of each required annotation is eased by using a tool feature that automatically adds all the needed properties to the elements in any architecture that will be subject of analysis. In the following paragraphs we detail which annotations are required and what is their purpose.

- *Reliability specification.* In order to generate a proper stochastic model that is able to accurately predict its reliability, it is required to add the probability of failure on demand (between 0 and 1) for each architectural element. This failure behavior is specified in terms of a percentage, denoting the number of successful requests over the total requests performed to that specific module. For instance, if a module has 80% of reliability, it means that 8 out of 10 requests are well performed and the other 2 fail on some cause, such as malformed input or other source of failure.
- *Assignment of usage profile (or operational profile).* We require the specification of transition probabilities between different components in order to model the average usage of each path in the system. Specifically, each user performs different actions, leading to the invocation of different components. Since, components may have different reliabilities, each action required by the user will output a specific reliability value.
- *Architectural Styles.* Architects may use different styles with well-known solutions to commonly occurring problems. Each style behaves differently and it provides distinct reliability values from each other. Hence, if a specific style is applied to the architecture, we require its specification in order to generate a stochastic model that exhibits its behavior. Our tool supports the following style patterns: fault-tolerance, parallelism, call-and-return and batch-sequential.
- *System control flow.* The stochastic model requires the specification of the transition flow that is being held from a component to another. Hence, we use annotated ports to distinguish between output and input transitions.

2.2 Stochastic Model Generation

Our approach contemplates the automated generation of an architecture-based stochastic model that expresses the reliability behavior of the system. Reliability is a probabilistic property that depends on the usage of the system, its architecture, used architectural styles and the reliability of each component. To express the behavior of the system a widely accepted method is the use of a Discrete

Time Markov Chain (DTMC) [6–9].
A discrete-time Markov Chain is a tuple $M = (S, \overline{s}, \mathbf{P}, L)$, where:

- S is a finite, non-empty set of states;
- $\overline{s} \in S$ is the initial state;
- $\mathbf{P} : S \times S \rightarrow [0, 1]$ is the transition probability matrix where $\Sigma_{s' \in S} \mathbf{P}(s, s') = 1$ for all $s \in S$;
- $L : S \times S \rightarrow 2^{AP}$ is a labelling function which assigns to each state $s \in S$ the set $L(s)$ of atomic propositions that are valid in the state. If $L(s_1) = L(s_2)$, then $s_1 = s_2$.

Our generation procedure uses the software architecture of the scenario to be analyzed, where each software component in the architecture is mapped into a state in the DTMC and each state has a probability of successfully respond to client requests. In addition, the transitions between states are defined as the usage of the system (also known as usage profile or operational profile).

The DTMC also includes absorbing states, which are self-loop transitions and model the successful and failure states of the system. The rationale behind this approach is that each request is processed through the various software components in the system and reaches one of two conditions: a successful (s_C) or a failure state (s_F).

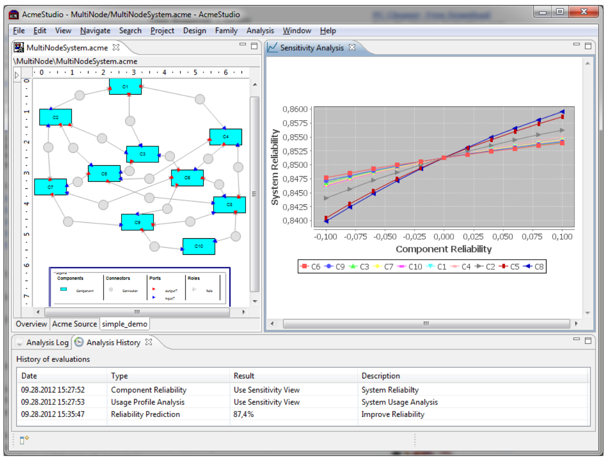
Regarding the assumptions that we rely on, we assume that every component has a probability of failing, so every state in the DTMC has an edge to the absorbing failure state. In addition, the system reliability is expressed through a reachability property ($true\ U\ state = s_C$) using Probabilistic Computation Tree Logic (PCTL) [10].

To predict the system reliability, Affidavit solves the DTMC by using Prism [11], a probabilistic model checking tool, which allows to compute the probabilities from the initial state s_i to the absorbing successful state s_C. Each architectural style has its behavior embedded in the DTMC by using code templates that express the relationships and actions of that particular style. Sensitivity analysis uses the generated DTMC to perform small variations in every component reliability and usage profile. The aim of this method is to identify critical points and diminishing returns in the architecture.

2.3 Implementation

Figure 2 illustrates the Affidavit tool simulating an architecture specified in AcmeStudio. As it can be seen from the *History View*, the depicted simulation has performed both the reliability prediction and sensitivity analysis on the specified architecture.
Reliability prediction is presented as percentage value and it is useful for comparing different architectures, guiding the user on what are the differences from an architecture to the other. As for the sensitivity analysis, it is presented through a graphic that relates the variations performed (*i.e.*, component reliabilities or system usage profile) with the overall system reliability.

The line chart exhibited in Figure 2 illustrates the variations performed in the reliability for each component in the architecture, represented as a line in the graph. This type of analysis provides information about which components have a greater impact in the overall system reliability and those which display diminishing returns.


AcmeStudio - MultiNode/MultiNodeSystem.acme - AcmeStudio
File Edit View Navigate Search Project Design Family Analysis Window Help
MultiNodeSystem.acme
Sensitivity Analysis
System Reliability
Component Reliability
C6 C9 C3 C7 C10 C1 C4 C2 C5 C8
Overview
Acme Source
simple_demo
Analysis Log
Analysis History

History of evaluations

Date	Type	Result	Description
09.28.2012 15:27:52	Component Reliability	Use Sensitivity View	System Reliabilty
09.28.2012 15:27:53	Usage Profile Analysis	Use Sensitivity View	System Usage Analysis
09.28.2012 15:35:47	Reliability Prediction	87,4%	Improve Reliability

Fig. 2. Architecture analysis using Affidavit

3 Demonstration

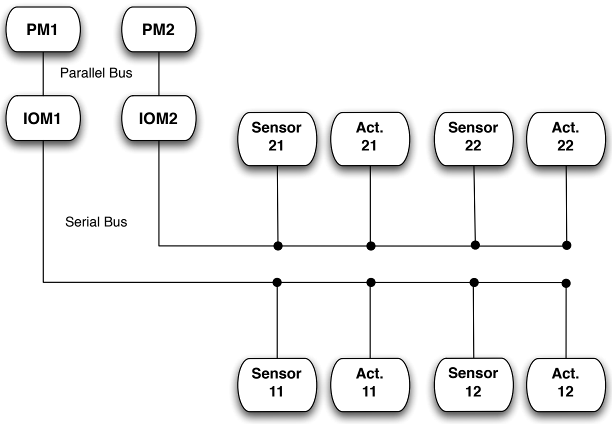
Affidavit main goal is to assess and analyze system reliability from an architectural specification. With this in mind, we modeled two different scenarios to demonstrate the usefulness and validity of our tool. The used architectures and reliability values specified in this demonstration were not extracted from a real system and they do not represent a real case-study. Their purpose is only to support our method and show the applicability of our tool. The two used scenarios have the same architectural elements and reliability values, but they differ in components' disposal and in the applied architectural styles. For each scenario, we present a reliability value output by that specific arrangement and we perform a thorough analysis to identify which components or connections are affecting system reliability the most.

3.1 Scenario #1

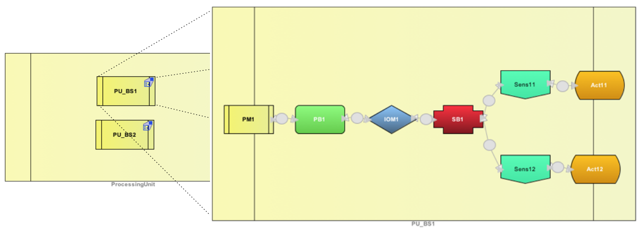
The system used in this scenario is illustrated in Figure 3. Its diagram is depicted in 3(a) and it is composed by a fault-tolerant architecture with two equal systems. Each system has:

– A Processing Module (PM);
– A Parallel Bus (PB);
– An Input/Output Module (IOM);
– Two Sensors (Sensor 1 and 2);
– Two Actuators (Act. 1 and 2).

Each sensor performs a different function in the system and it will invoke its own actuator (*i.e.*, Sensor1 invokes Actuator1, but not Actuator2). Hence, from the total number of requests performed by the system, we specify the usage profile as 40% of the requests are resolved by Sensor1 and the other 60% are resolved by Sensor2.



(a) System's Diagram



(b) Representation of the system's architecture in AcmeStudio

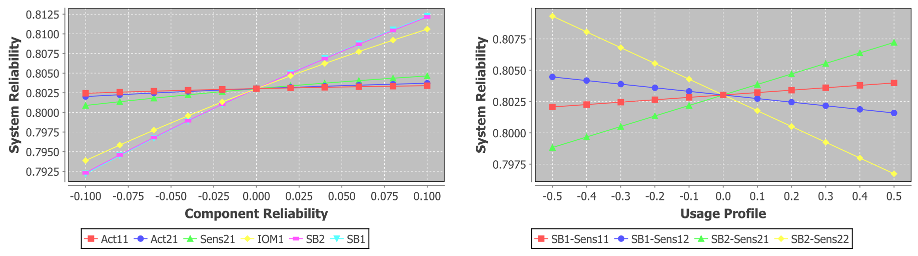
Fig. 3. Scenario #1

Then, we modeled this scenario in AcmeStudio, illustrated by Figure 3(b), and we specified the reliability values according to Table 1.

The performed analysis predicts that the modeled system has 80.3% of reliability. Meaning that from the total number of requests performed in the system,

Table 1. Component reliabilities

Component	PM1	PB1	IOM1	SB1	Sens11	Sens12	Act11	Act12
Reliability	0.95	0.98	0.85	0.80	0.99	0.92	0.99	0.95
Component	PM2	PB2	IOM2	SB2	Sens21	Sens22	Act21	Act22
Reliability	0.95	0.98	0.85	0.80	0.94	0.91	0.98	0.93



(a) Variation on components' reliabilities (b) Variation in system usage profile

Fig. 4. Sensitivity Analysis on Scenario#1

19.7% fail on some cause and they cannot be successfully resolved. Figure 4 shows the variations performed in the sensitivity analysis process according to the method presented by Franco *et al.* [12]. Graphic 4(a) depicts the variation of 10% on the reliabilities of the different components in the system. In this graphic only the three best and worst reliability variations are illustrated from a total of sixteen components. We rank the variations by calculating the derivative of the reliability around the point where the variation is null (*i.e.*, variation of 0%). This ranking is shown through the graphic's caption, where, from the left to the right, components are ordered from the lower to the higher increase of the impact on the overall system reliability.
In this scenario, components Act11, Act21 and Sens21 are the ones in which their variation has less impact on the overall reliability, and can be considered as the diminishing returns in the system. On the other hand, components SB1, SB2 and IOM1 are the ones where their variation has the highest impact. As a side note, IOM2 presents the same variation value than IOM1, but it has been omitted since our approach only presents three variations and IOM1 is listed first than the IOM2.

Regarding usage profile, Graphic 4(b) shows the variation of 50% on the load of requests that are performed from the SB component to the Sensors. The connection from SB1 to Sens11 shows that it is an already diminishing return, but the increase on usage profile from SB2 to Sens21 leads to an improvement on the overall system reliability. In addition, Graphic 4(b) informs architects that the load in the connection from SB2 to Sens22 and Sens21 is not fair, and it should be subject to change to get the maximum benefit on the system reliability.

3.2 Scenario #2

In this scenario we applied the same architectural elements, reliability values and usage profile from Scenario #1. However, as can be seen from Diagram 5(a), it differs in the applied styles and in the architectural structure. Specifically, we put together the two Serial Buses as one, which acts as fault-tolerant, connecting every other architectural element to it. In addition, we joined the Sensors and Actuators from the previous scenario and we setup them as fault-tolerant.
Figure 5(b) illustrates the system representation in AcmeStudio, which was used for predicting and analyzing system reliability. Thus, in this scenario the predicted reliability was of 91.1%, which is an increase of 10.8% when comparing with the previous scenario.

Figure 6 depicts the sensitivity analysis performed on the system. The variation on the reliability of system components is illustrated in Graphic 6(a) and since we used the same reliability values as in Scenario#2, the results are very similar. The only difference is that the component Sens21 in the previous scenario is replaced by Act22 as one of the diminishing returns. Regarding usage profile, Graphic 6(b) shows that increasing the load of requests on the connection SB-Sens2 would increase the system reliability.

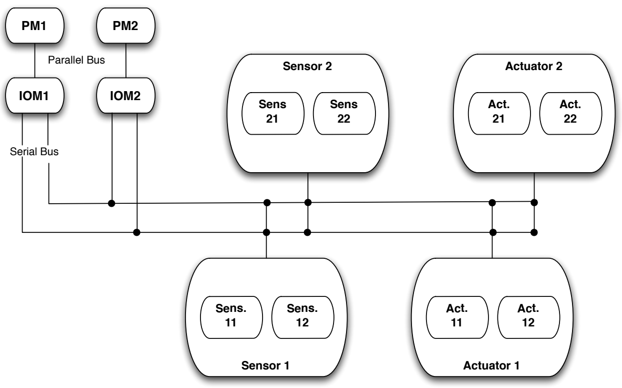
3.3 Demonstration Conclusion

The provided examples show that our tool is able to assess different architectures taking into consideration the distinct styles applied. Therefore, we provide means for architects compare, test and validate different architectures and possible architectural solutions that would fulfill the quality requirements established by the stakeholder. In addition, the sensitivity analysis informs architects on what should be the future direction to evolve the system, regarding reliability as a quality attribute.

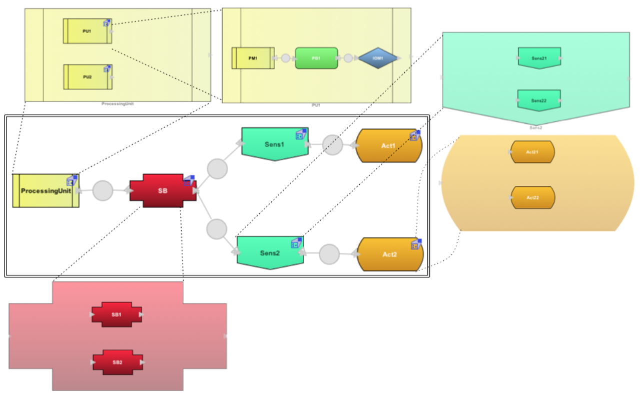
As an example, from the information provided by the sensitivity analysis in Scenario#2 (*i.e.*, increasing by 10% the reliability of components SB1, SB2 and IOM1, as well as, increasing the load of requests sent to the component Sens2 by 20%) Affidavit predicted the system reliability as 92.7%. Thus, an improvement of 12.4% and 1.6%, when comparing with Scenario #1 and #2, respectively.

4 Related Work

In this section we present set of tools or methods that allow to perform quantitative prediction of properties, such as reliability or performance, from an architectural description. Palladio Component Model [13, 14] is a tool that supports the prediction of both performance and reliability, as non-functional properties,

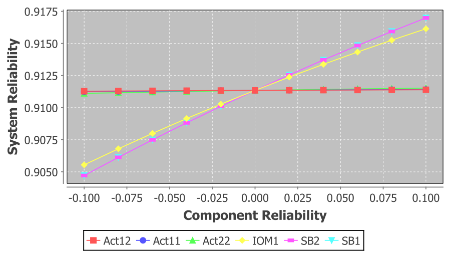


(a) System's Diagram

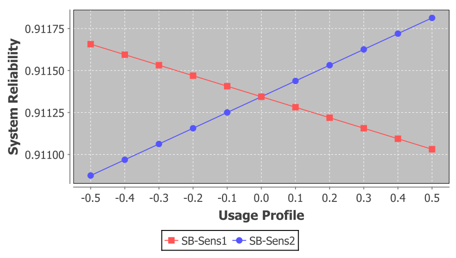


(b) Representation of the system's architecture in AcmeStudio

Fig. 5. Scenario #2



(a) Variation on components' reliabilities



(b) Variation in system usage profile

Fig. 6. Sensitivity Analysis of Scenario#2

from software architectures during design. It is targeted at model-driven quantitative predictions, deriving stochastic regular expressions and queuing network models. In addition, it allows to identify bottlenecks and support architectural design decisions. However, Palladio does not account with different architectural styles and nothing is stated on which ADL is used.

LIGHT [15] is a platform that allows to generate and customize a Domain-Specific Language (DSL) for a particular project. In addition, this platform performs a set of simulations to analyze certain system features, such as latency, memory usage, energy consumption and reliability. Regarding reliability, LIGHT platform implements the method developed by Roshandel *et al.* [16]. Hecht et. al [17] demonstrate the use of Mobius (a tool for developing dependability models of stochastic, discrete-event systems) on AADL to automatically generate a reliability/dependability model. In particular, they use a graphical representation of the system along with error behavior specification producing Stochastic Petri Nets (SPN) and Stochastic Activity Network (SAN) representations. Mobius allows to specify the failure detection and recovery behavior of the system, although this specification relies on manual effort to be produced.

Altarica [18] is a workbench dedicated to reliability and dependability analyses of critical systems by using the Altarica language. This language is specified as a constraint automata and is compiled into lower level formalisms such as Boolean formulae, Petri nets and finite state automata. However, the constraint automata are not generated automatically from an architectural description, which is the main focus of our work. Although it would be conceivable for the Affidavit tool to translate ADL specifications into Altarica models, we chose to translate into the Prism language to model the system behavior. The rationale behind this choice is that Prism performs probabilistic model checking, which allows not only to formally verify system properties (e.g., safety, liveness), but also to obtain a probability of reaching a particular state.

Affidavit differs from previous works by assessing, predicting and analyzing software architectures without any manual effort. Our approach generates a Deterministic-Time Markov Chain (DTMC) expressing the reliability behavior. This behavior specification accounts with predefined architectural styles that are precompiled and ready to be used, however, it can be extended to support different ones by enriching their proper behavior to the Prism generated file. We applied our approach to Acme, a known architecture description language (ADL) which complies with the conventions of ISO/IEC/IEEE 42010 [19], an international standard for systems and software architecture descriptions. As a result, our approach offers abilities to architects reason about system reliability and assure that a particular architectural setup meets the desired quality goals.

5 Conclusion and Related Work

Today, most of the assessments and analyses of quality attributes are performed through the generation of stochastic models that exhibit the proper behavior of the system. One of the problems with this approach is that these models are most often constructed by hand and, as in any other manual activity, it is an error-prone and time-consuming task. Thus, we developed a tool that automatically generates stochastic models to predict and analyze system reliability, allowing to reduce the manual effort by decreasing the time spent on developing stochastic models and the propensity for error occurrence. In addition, the generated models can be modified by the user to enrich them through adding different architectural styles or particular features in order to make them closer to the reality as possible.

In this paper we demonstrate the applicability and usefulness of the Affidavit tool which is available for architects to test, experiment and analyze reliability properties in their modeled architectures. Affidavit can be applied in the design phase to detect and fix early reliability issues and ensure that a particular architecture meets the stakeholders reliability requirements, even before a system prototype has been built. Moreover, our approach supports architectural evolution by providing information on which architectural elements require reliability improvement and by identifying reliability trade-offs from the comparison of the original architecture with the evolved one. As a result, our approach gives support for practitioners and researchers to avoid, prevent and detect undesired or infeasible architectural redesigns which could result in degradation of the overall system reliability.

Regarding current work, we are applying our automated generation of stochastic models for self-adaptive systems. The rationale is to integrate reliability prediction methods to reason about possible adaptation strategies and decide whether a specific strategy will meet the self-adaptive goals or not. As future work, we plan to show the applicability of our approach into real world scenarios by assessing and analyzing the reliability of an open source software project. The goal is to promote our tool as well as to contribute to the open source software community.

Acknowledgment

This research was supported by a grant from the Carnegie Mellon|Portugal project AFFIDAVIT (PT/ELE/0035/2009), Fundação para a Ciência e Tecnologia (FCT), Portugal [SFRH/BD/89702/2012] and iCIS Project (CENTRO-07-ST24-FEDER-002003)..

References

1. Affidavit: Webpage. `http://affidavit.dei.uc.pt/`
2. Garlan, D., Monroe, R.T., Wile, D.: Acme: Architectural description of component-based systems. In Leavens, G.T., Sitaraman, M., eds.: Foundations of Component-Based Systems. Cambridge University Press (2000) 47–68
3. Schmerl, B., Garlan, D.: Acmestudio: Supporting style-centered architecture development. In: Proceedings of the 26th International Conference on Software Engineering. ICSE '04, Washington, DC, USA, IEEE Computer Society (2004) 704–705

4. Franco, J.M., Barbosa, R., Zenha-Rela, M.: Automated reliability prediction from formal architectural descriptions. In: Software Architecture (WICSA) and European Conference on Software Architecture (ECSA), 2012 Joint Working IEEE/IFIP Conference on. (aug. 2012) 302 –309
5. Beaton, W., d. Rivieres, J.: Eclipse platform technical overview. Technical report, The Eclipse Foundation (2006)
6. Goševa-Popstojanova, K., Trivedi, K.: Architecture-based approach to reliability assessment of software systems. Performance Evaluation **45**(2) (2001) 179–204
7. Gokhale, S., Wong, W., Trivedi, K., Horgan, J.: An analytical approach to architecture-based software reliability prediction. Proceedings. IEEE International Computer Performance and Dependability Symposium. (1998) 13–22
8. Filieri, A., Ghezzi, C., Leva, A., Maggio, M.: Self-adaptive software meets control theory: A preliminary approach supporting reliability requirements. Automated Software Engineering (ASE), 2011 26th IEEE/ACM International Conference on (2011) 283–292
9. Franco, J.M., Barbosa, R., Zenha-Rela, M.: Automated Reliability Prediction from Formal Architectural Descriptions. In: 2012 Joint Working IEEE/IFIP Conference on Software Architecture and European Conference on Software Architecture, IEEE (August 2012) 302–309
10. Kwiatkowska, M., Norman, G., Parker, D.: Stochastic model checking. Formal Methods for Performance Evaluation (2007) 220–270
11. Kwiatkowska, M., Norman, G., Parker, D.: Prism: Probabilistic model checking for performance and reliability analysis. ACM SIGMETRICS Performance Evaluation Review **36**(4) (2009) 40–45
12. Franco, J.M., Barbosa, R., Zenha-Rela, M.: Reliability analysis of software architecture evolution. In: Dependable Computing (LADC), 2013 Sixth Latin-American Symposium on. (2013) 11–20
13. Becker, S., Koziolek, H., Reussner, R.: Model-based performance prediction with the palladio component model. In: Proceedings of the 6th international workshop on Software and performance. WOSP '07, New York, NY, USA, ACM (2007) 54–65
14. Martens, A., Koziolek, H., Becker, S., Reussner, R.: Automatically Improve Software Architecture Models for Performance , Reliability , and Cost Using Evolutionary Algorithms. Population (English Edition) (2010)
15. Edwards, G.: Automated Analysis and Code Generation for Domain-Specific Models. Software Architecture (WICSA) and European Conference on Software Architecture (ECSA), 2012 Joint Working IEEE/IFIP Conference on (2012)
16. Roshandel, R., Banerjee, S., Cheung, L., Medvidovic, N., Golubchik, L.: Estimating software component reliability by leveraging architectural models. Proceeding of the 28th international conference on Software engineering - ICSE '06 (2006) 853
17. Hecht, M., Lam, A., Vogl, C.: A Tool Set for Integrated Software and Hardware Dependability Analysis Using the Architecture Analysis and Design Language (AADL) and Error Model Annex. 2011 16th IEEE International Conference on Engineering of Complex Computer Systems (April 2011) 361–366
18. Labri, I.X.I., France, B., Rauzy, A.: AltaRica Constraint automata as a description language. European Journal on Automation (1999)
19. ISO/IEC/(IEEE): ISO/IEC 42010 (IEEE Std) 1471-2000 : Systems and Software engineering - Recomended practice for architectural description of software-intensive systems (July 2007)