Automated reliability prediction from formal architectural descriptions

João M. Franco, Raul Barbosa and Mário Zenha-Rela
University of Coimbra, Portugal
Email: {jmfranco,rbarbosa,mzrela}@dei.uc.pt

***Abstract*—Quantitative assessment of quality attributes (*i.e.*, non-functional requirements, such as performance, safety or reliability) of software architectures during design supports important early decisions and validates the quality requirements established by the stakeholder. In current practice, these quality requirements are most often manually checked, which is time-consuming and error-prone due to the overwhelmingly complex designs. We propose an automated approach to assess the reliability of software architectures. It consists in extracting a Markov model from the system specification written in an Architecture Description Language (ADL). Our approach translates the specified architecture to a high-level probabilistic model-checking language, supporting system validation and quantitative reliability prediction against usage profile, component arrangement and architectural styles. We validate our approach by applying it to different architectural styles and comparing those with two different quantitative reliability assessment methods presented in the literature: the composite and the hierarchical methods.**

***Index Terms*—software architecture, reliability modelling, model checking**

I. Introduction

Software architecture is a discipline of software engineering, supporting the specification of non-functional requirements (*e.g.*, performance, maintainability, security, reliability) during the design stage of the software development cycle. At this software development stage, architectural decisions will largely influence the quality of the software and determine if a particular non-functional requirement defined by the stakeholder is complied by the system under construction. Architecture Description Languages (ADLs) allow one to model, represent and describe a software architecture, thereby improving the artefacts used for communication among designers, developers and stakeholders. ADLs, such as Acme [1], Wright [2], AADL [3], support annotations to specify relevant properties for analysis and validation of quality attributes.

As one of the key metrics for determining the quality of software, reliability prediction is important to assure that a particular architecture provides a correct and accurate probability of failure-free operation within a specified exposure time interval.

Early reliability prediction in the software development life cycle allows architects to reason about how the constituents of the architecture will affect the overall system reliability. In other words, the practitioners can improve, test and validate a software architecture according to the components' reliability, architectural styles, component and connector arrangements and the expected usage profile of the system. Reliability assessment in an early design stage, also provides architects with the assurance that a particular architecture meets the quality requirements established by the stakeholders, preventing additional costs of fixing problems detected late during the life cycle and architectural redesigns of the system.

In current practice, very few of the non-functional requirements are automatically checked. This manual checking activity is prone to errors and time-consuming due to the overwhelming complex designs, as a result of a high number of components, connectors and the interconnections between them, as well as the possible architectural decisions. Therefore, an automated verification, validation and testing of the quality attributes of a software architecture is becoming more needed for practitioners.

In this work, we present an approach that takes a described architecture through an ADL, extracting automatically a stochastic model. The models generated allow the architect with validation and prediction procedures against a quality attribute, the reliability. Our approach also supports experimentation to inform the practitioner on what solution is the most reliable. Particularly, the architect can test the system by varying all the architecture constituents (*e.g.*, architectural styles) and their reliabilities, test and compare it by obtaining the reliability output given by that set.

The contribution of this paper is the automated generation of a stochastic model from an architecture specification. We applied our approach to assess the reliability of the system by generating a mathematical model which allows the architect to validate, verify and test different architectural solutions and select what best suits the stakeholders requirements. Four degrees of freedom are at the disposal of the architect, namely the architectural styles, component reliabilities, interactions between components and the usage profile. Those degrees of freedom can be applied to any architecture, letting practitioners make use of them to find the most reliable solution.

We validate our approach through two different steps. Firstly, we validate our reliability prediction method with the results presented in previous researches by applying the same scenarios [4], [5]. In the second step, we validate our approach by including architectural styles to our reliability prediction method and compare the results obtained from the methods presented in Wang *et al.* [6] with ours.

This paper is organised as follows. Section 2 presents the background and related work. Section 3 introduces the method

adopted in our approach and Section 4 describes the validation process of our approach and compares the results obtained with previous research studies. Finally, Section 5 reveals the limitations and Section 6 discusses our insights about automated reliability prediction before Section 7 concludes.

II. Related Work

The main objective of reliability prediction (as the probability of failure-free operation in a given time span) based on software architecture is to obtain an estimate of the system reliability. Several studies address the reliability assessment from a software architecture description [7]–[10], among the firsts to propose architecture reliability modelling using Markov chains was Cheung [11] and several surveys were presented since then [12]–[15].

According to these surveys and as depicted by Figure 1 the reliability assessment of software architectures can be performed through three different approaches which combine the architecture with the failure behaviour [13]: additive, path-based and state-based models.

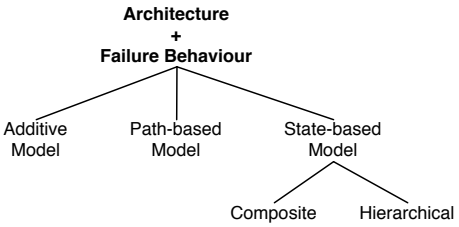


Fig. 1: Approaches to combine the architecture with the failure behaviour

The former estimates the reliability using the component's failure data and does not consider explicitly the architecture of the software. Hence, as the architecture arrangement, used styles and its usage profile are the main focus in our reliability assessment, the additive model will not be considered in this paper.

The path-based model assesses the reliability of the system according to the possible execution paths of the program, which can be obtained experimentally, by testing or algorithmically. In other words, the reliability of each path is obtained as a product of the reliabilities of the components along the path. System reliability is calculated by averaging the path reliabilities, which can be a drawback due to the presence of loops in the architecture, providing only an approximate estimation of the system reliability.

The latter, state-based model, assumes that the transitions between states have a Markov property, meaning that at any time the future behaviour of components or transitions between them is conditionally independent of the past behaviour. In addition, the state-based models can be divided into composite [11], [16] and hierarchical [4] methods. The former combines the architecture of the system and the failure behaviour of its components into a single model. The hierarchical method considers that the architecture and the failure behaviour are detached, more specifically the architecture is modelled by a semi-Markov process and the failure behaviour can be modelled according to a Poisson process [17] or by a time-independent failure rate [18].

Hierarchical methods are simpler to compute than the composite ones. Since the failure behaviour is detached from the architecture, the reliability prediction can be computed directly by applying a mathematical formula, without re-constructing and re-solving the combined models as in the composite method. Therefore, the advantages of hierarchical over composite are essentially about performance; however, the major drawback of the hierarchical method is that it only provides an approximation of the reliability, and hence the reliability metrics obtained using this model are not as accurate as the ones from composite models.

Above methods are theoretical mathematical models to assess the reliability in an early software development stage that are accurate enough to be applied to real case studies. Popstojanova *et al.* [19] studied and tested the test suite of the C compiler to prove the adequacy, applicability and accuracy of software reliability models. The results obtained show that the actual reliability differs only by less than 3% from the theoretical methods, proving that both the composite and hierarchical models are very accurate and applicable for real case studies.

Over the years, common patterns of structural organization of components and connectors have been identified and documented [20]. The so-called architectural styles are commonly used in any architecture, but they impose constraints in reliability assessment: each architectural style maps to a different state-space model and it must be extended to reflect some architectural choices, such as concurrency or fault-tolerance. Abd-allah [21] identified the issues of reliability assessment of architectural styles using reliability block diagrams and Wang *et al.* [6] described the process of mapping a limited number of architectural styles to state space models for reliability analysis. Only few research studies address the reliability analysis on architectural styles. More interest is needed on this topic to analyse the reliability on important styles that were not considered before, such as event-based or blackboard repository.

Martens *et al.* [22] present an approach to quantitatively predict the performance, reliability and cost of a software architecture. Their approach supports a multi-criteria genetic algorithm to find the best trade-off between those quality attributes. Regarding reliability analysis, they performed two types of analysis: transform the software architecture into an absorbing discrete-time Markov chain and a reliability simulation to derive the probability of failure on demand. Their work differs from ours in two aspects: they predicted the system reliability by introducing hardware faults and they did not take into consideration the different architectural styles that may be used.

III. METHOD

Software reliability prediction based on architecture consists of several distinct techniques. In this section, we discuss which of these techniques have been applied to our approach and how our translation procedure is performed. In addition, we explain how the architectural styles have been translated and we state what are the assumptions on which we rely.

A. Reliability Prediction Process

Several different approaches to perform reliability prediction were described in previous research studies, targeting different failure behaviours and different reliability assessment methods. For this reason, we specify what are our assumptions and how our approach is performed to the process of reliability prediction.

1) Architecture and Module identification: Software architecture defines the software behaviour in respect to the manner in which the different modules interact between them. A module is conceived as an independent component of the system, which performs a clear and well-defined function in the system. Moreover, a module may represent an available service (such as a web-service), a function, a class or even a group of classes that have the same functionality or the same interfaces [13], [19].

2) Failure Behaviour: In our approach a failure may occur during the control transfer between two different modules and is known as the Probability Of Failure On Demand. The failure behaviour of the modules 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, including hardware and software failures.

To assure that the architecture complies the reliability requirements, the reliability of each component can be estimated by giving reliability ranges for each component in the design phase or can be obtained using the failure data collected during the testing and operational phases of a previous implemented component.

3) Combining the architecture with the failure behaviour: As stated in Section II, the composite model from the state-based approach is the most accurate for reliability prediction when compared to other solutions. Therefore, our approach uses the composite model through the generation of an absorbing DTMC (Discrete-Time Markov Chain), where we add two absorbing states C and F, which represent the correct output and the failed one, respectively.

Our approach acts in accordance to the following assumptions:

- Every software component can fail. Each module that is mapped from the architecture to the mathematical model has a direct edge to the absorbing state F, which is weighted by its probability of failing (*i.e.*, one minus the assigned reliability to the component).
- The failures are independent between software components. Components in a software system can be viewed as logically independent modules, which can be developed and tested independently from each other [4], [5].
- The transfer of control among modules follows a Markov Process. The transition probability from one component to another is determined through the product of the reliability of that component with the estimated usage profile of the system. Therefore, the control transition is independent of the past history of the system and depends only upon the current state, following the memoryless property of a Markov chain [23].
- System reliability is the probability of reaching the state C. The computation of the system reliability is performed through the probability of transit between all the components in the system and reaching the absorbing state C, which exhibits the correct behaviour of the system or the probability of failure-free of every component in the system.

B. Translation Process

We refer to translation process as the procedure of taking as input a system description in an ADL format and generate automatically a mathematical model, exhibiting the behaviour and the control flow of the system. In particular, we exploit ADL annotations by extending architectural design entities with relevant information for architecture design and analysis [24]. Our approach supports the following annotations to build a complete model of the system:

- *Specification of the control flow of the system*. The architect can specify the flow of transitions that are being held from a component to another, by using annotated ports to distinguish between output and input transition.
- *Identification of architectural styles used in the system*. The identification of what styles are being used in the architecture is a requirement for a faithful translation from the architecture to the mathematical model.
- *Assignment of system usage profile*. This can be achievable by specifying a transition probability annotation to a connector, identifying the usage profile of the system. For each component, all its transition probabilities must sum up to one.
- *Reliability specification for each one of the components in the system.* Each component must be annotated with a probability of failure on demand, which will quantify its reliability value.

The translation process is illustrated by Figure 2. It can be observed that our approach parses the ADL file into an intermediate representation of the constituents of the system along with the proper annotations. The intermediate representation allows any ADL that complies with the ISO/IEC/IEEE 42010 [25] standard to be parsed and analysed. We have successfully used Acme, although other ADLs may be target of future work.

After the file has been parsed, our application translates the architecture by building a mathematical model in a high-level

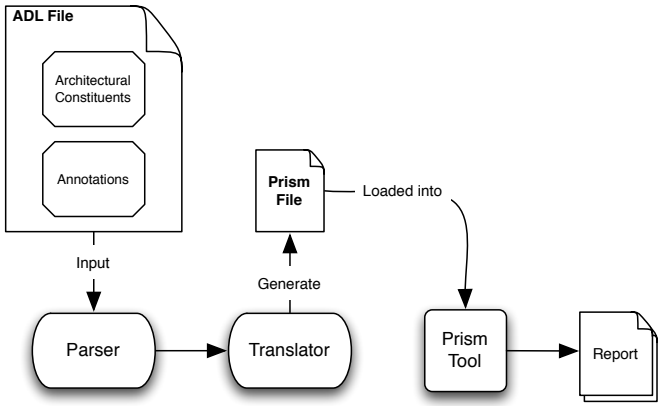


Fig. 2: Translation Process Workflow

formal language, which can be loaded into the probabilistic model checking tool, Prism [26].

Prism allows to perform three types of analysis: verification, simulation and experimentation. The verification process allows testing the correctness of the model, the absence of deadlocks and provides the reliability value of a single execution without testing it for the occurrence of loops. Regarding the simulation process, Prism allows to generate a large number of random paths through the model, evaluating each one of the paths, and using this values to generate an approximate result.

It is also possible to make experimentations on Prism by, for instance, varying the components' number or their reliabilities. Experimentations in Prism are represented by graphical visualizations, providing insights to the architects about what is the test suite that best fits the stakeholder requirements.

C. Architectural Styles

Architectural styles, also known as architectural patterns, are well-documented solutions to commonly occurring problems. In addition, the application of these styles may exhibit known quality attributes, so architects may use one particular style to solve performance or availability problems [27], [28].

In our approach we used the same architectural styles presented by Wang *et al.* [6], in order to compare and validate our approach. We shall now describe each one of the styles and how the translation process was performed.

1) Batch-sequential: In the batch-sequential style, the components are executed in a sequential order, therefore, only one component is executed at any instance of time. This style can be modelled as shown in Figure 3, where $s_1, s_2, \dots, s_k$ are the mapping states from the software components and represent its execution. Upon the completion of their execution, the control is transferred to one of its following components. If there is more than one component to pass the control to, the selection is made through the transition probability from the state i to state j. This probability corresponds to the estimated usage profile of the system, denoted by $P_{i,j}$ and summing up to one for each state.

The absorbing states *C* and *F*, express the correct and the failed output, respectively. Each state has a probability of failing, so there is a non-null probability of transit to the *F* state, which is equal to one minus its reliability value, denoted by $R_1, R_2, \dots, R_k$. The system reliability is computed from the transitions across all the states and reaching the correct output state, the *C* state (probability of failure-free operation).

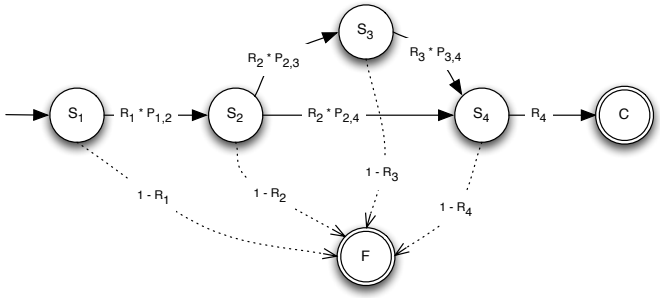


Fig. 3: Batch-sequential style state model

2) Parallel and Pipe-filter: Parallel and pipe-filter styles are commonly used to model systems that exhibit concurrent executions to improve performance. In these styles, the work is partitioned and each component works on a small subtask to complete a larger task. Wang *et al.* joined these two styles, because they both share the same behaviour, although they differ on the processing environment. The parallel style assumes multi-processor while the pipe-filter is usually applied to a single processor.

The state space model is depicted in Figure 4, where architectural components that represent a single execution are mapped to states s_1 and s_3, in the same way as the batch-sequential style. The inherent concurrent executions to the parallel and pipe-filter styles are mapped to different states, as shown by states $s_{21}, s_{22}, \dots, s_{2k}$. These states are wrapped in the s_2 state, which is responsible for the synchronization process of when a transition to these states occur and when the concurrent executions of all parallel states are completed.

In case of failure of one of the concurrent states, the computation of the subtask will not be completed and the system will go into a failed state.

3) Fault-tolerance: A fault-tolerance style may be applied to a system in order to obtain higher reliability or few failures in a specified time period. This style is composed of a set of components that try to compensate the failures of each other. More specifically, only one component is performing computation, the active component. When it fails, one of the redundant components takes over the failed, becoming the active one. The system only enters in a failed state, when all the components in the fault-tolerant set fail. Our approach supports different reliability values for each one of the components in the set, as they may involve different data structures or algorithms to improve the system reliability.

The state model of this style is illustrated by Figure 5, where the fault-tolerant set wrapped in the state s_2 is shown.

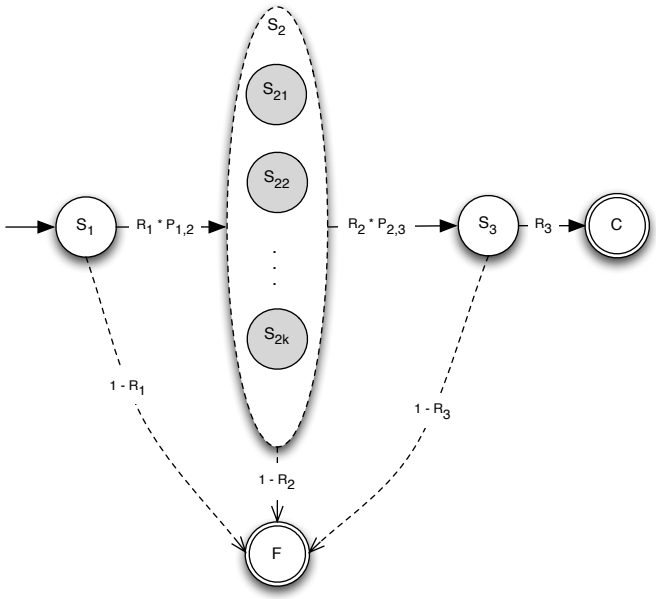


Fig. 4: Parallel and pipe-filter style state model

The active state is depicted with a gray background and the redundant states with a white one.

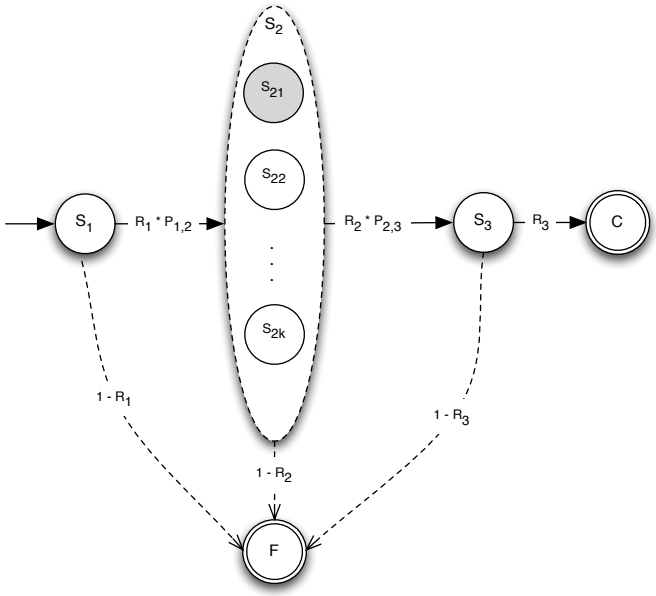


Fig. 5: Fault-tolerance style state model

4) Call-and-return: In the call-and-return style, a caller component may request services provided by other called components. When the services are requested, the caller component holds its execution, until the called one fulfils the requests. After that, the caller component resumes its execution where it left. This style is often used on remote procedure calls and it can be translated to the state space model as follows: a state represents an execution of a component and a transition takes place when the execution in each state is completed or the execution encounters a service request transferring temporarily the control to the called component.

Figure 6 depicts a call-and-return style, where state s_1 is the caller and s_2 is the called component. After calling the state s_2, the caller state will transfer the control to the state s_3, which, if everything went as expected, the control will be transferred to the absorbing component *C*. As it can be seen in Figure 6, the transition $P_{1,2}$ does not consider the reliability of the caller component, because, as stated in Wang *et al.* [6], s_1 will be visited only once before transferring the control to state s_3 regardless the number of times the state s_2 is called.

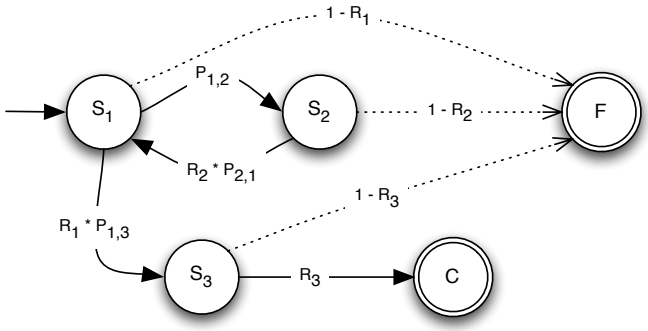


Fig. 6: Call-and-return style state model

IV. Validation

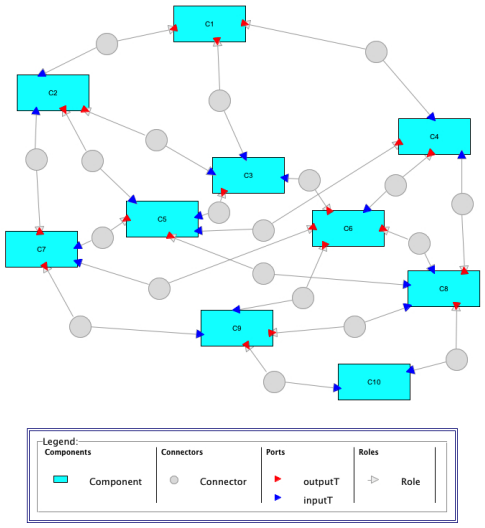
The validation procedure of our approach was divided in two different steps. First we validated our reliability prediction method by applying a software architecture widely used in the literature and comparing the accuracy of our results to the ones stated in the cited publications. The second step of our validation procedure, contemplates the validation of the reliability by applying architectural styles, so we compared our reliability values against the results obtained from the method presented by Wang *et al.* [6].

A. Reliability Prediction

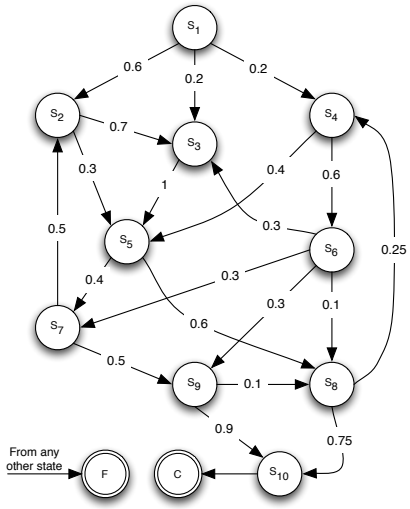
To validate the accuracy of our reliability prediction method we built a software architecture in Acme, depicted in Figure 7a, which was adapted from the examples by Cheung [11], Lo *et al.* [5] and Gokhale *et al.* [4]. The architecture includes all the required annotations to enable the complete parsing and translation procedure, generating an accurate mathematical model. In Figure 7b is depicted the state model generated by our approach.

All the compared publications use the same architecture, although they use different reliability values for each component, as depicted in Table I. So, we applied the same software architecture, usage profile and reliability values from each one of the publications to compare and validate our results.

Finally, we loaded the mathematical model into the probabilistic model-checking tool, Prism, and obtained the reliability values for the specified architecture. In Table II we compare



(a)



(b)

Fig. 7: Architecture described in Acme (a) and the associated state model (b)

TABLE I: Component reliabilities

R_i	Gokhale *et al.* [4]	Lo *et al.* [5]
1	0.999	0.99
2	0.980	0.98
3	0.990	0.99
4	0.970	0.96
5	0.950	0.98
6	0.995	0.95
7	0.985	0.98
8	0.950	0.96
9	0.975	0.97
10	0.985	0.99

the reliability values with the ones from the literature. Cheung *et al.* [11] used a composite method through an absorbing DTMC to predict the reliability of an architecture. Lo *et al.* [5] made use of a hierarchical method to predict the reliability and Gokhale *et al.* [4] presented the results using both methods of the state-based approach, the composite and hierarchical methods.

As a result, Table II shows that our approach provides reliability values that match to the ones presented in the literature. More specifically, our reliability values are exactly the same as the ones provided by the composite methods and they are close to the values obtained from the hierarchical. Since, as stated in the Section II, the hierarchical method only allows us to obtain an approximation of the reliability values and the maximum difference to our results is 0.35% which is below 1%, the significance level.

TABLE II: Validation of the reliability prediction method

		Literature	Our Approach	Difference
Cheung *et al.* [11]		0.8512	0.8512	0.00%
Lo *et al.* [5]		0.8482	0.8512	0.35%
Gokhale [4]	Composite	0.8299	0.8299	0.00%
	Hierarchical	0.8280		0.22%

B. *Architectural styles*

In the second part of our validation procedure, we certify that our approach generates a correct mathematical model and provides an accurate reliability value when an architectural style is applied.

The input architectures used to test the validity of our approach are the ones presented in Section III-C. They were modelled in Acme and we used our approach to generate the mathematical model. Finally, they were loaded into the Prism model checker tool, to verify and simulate the architecture. We tested the fault-tolerant style with one active and two redundant components and the parallel style with three parallel components.

The comparison between the results obtained from our approach and the ones achieved through the methods presented by Wang *et al.* [6], are exposed in Table III.

TABLE III: Validation of the architectural styles

Style	Wang [6]	Our approach	
		Reliability	*Diff.*
Batch-sequential	0.9248722	0.9248722	0.0%
Parallel	0.8945088	0.8945088	0.0%
Fault-tolerance	0.9503923	0.9503923	0.0%
Call-and-return	0.9317644	0.9317631	$\sim$ 0.0%

Considering the values provided by previous research studies, our results are identical, proving that our approach generates accurate and correct mathematical models when using architectural styles.

V. Limitations

The existing limitations in our approach are discussed below, alongside with references to tools or research studies that address those same limitations.

- *Components reliability must be known*. Our approach requires that architects know beforehand a value, or at least a range, for the reliability of the components in early phases of the software development life-cycle, which can be difficult. L. Cheung *et al.* [29] address this uncertainty by using hidden Markov chains to determine the component failure probability. Other studies [4], [5], [30] compute the sensitivity of the system's reliability by varying the components reliability and the usage profile.
- *Usage profile has to be defined*. During design time, inter-component transition probabilities can be estimated by consulting with experts who are familiar with the system or by defining various possible scenarios and determine what will be the expected usage profile of the system. If the reliability analysis is to be employed during the operational phase, the usage profile can be extracted from the source code by using profilers [31] or test coverage tools [32].
- *State-space explosion*. Model checking tools face a common problem, the combinatorial growth of the state-space. This occurs when the model has a large number of states and a great number of transitions between those states exceeding the memory available. In our approach we have not faced this problem, not only because we used simple architectures, but also because the translation procedure to the Prism language is optimized by using the smallest number of states and transitions possible. Groote *et al.* [33] explain how to reduce the size and avoid extremely large models, preventing the occurrence of state-space explosion.

VI. Implications for practice

In this section we share several insights from our work, contributing with answers to future research on the topic of reliability prediction of software architectures.

- *Different architectural styles will provide different reliability values.* We were able to predict different reliability values for distinct architectural styles, even though the architecture arrangement is the same. This is achieved by using annotations to describe which styles were applied in the architecture. In Section IV we show the validity of this assumption by applying two different architectural styles (fault-tolerance and parallel styles) to the same architecture description, obtaining different reliability values.
- *Correctness of the generated mathematical model.* Mathematical models from the architecture description represent correctly the system's behaviour and provides accurate reliability values as shown in the validation phase, Section IV.
- *Application to real case studies.* One of our concerns was if the generated models could be applied to real case studies. Goseva-Popstojanova *et al.* [19] determined the applicability of the reliability prediction methods provided by the literature on a real case study and the results show that the theoretical methods only differ on 3% from the real values.
- *Influence of our work on the current techniques.* The automated generation of mathematical models from the architecture specification is an important topic, since until today the verification and testing procedures were manually built. This manual activity is prone to errors, time-consuming and almost impossible to achieve on complex and large architectures.
- *Application to other quality attributes.* In the generated mathematical model we have only predicted the system's reliability, but it can be applied to other system's quality attributes, such as cost and performance. This is addressed as future work, in which we could join work with other studies that have predicted the system's performance. Their work focus on building a mathematical model with specific annotations to load it on a probabilistic model checking tool. This way, architects would be able to assure a more thorough quality of the designed software architecture, complying the requirements of the stakeholders.

VII. Conclusion

In this paper we presented an approach to automate the reliability prediction of software architectures considering the application of different architectural styles. Our approach takes advantage of the progress made to ADLs in the last two decades, leveraging the formalization of ADLs to extract a mathematical model. This would not be possible without a formal representation and description of software architectures.

Our work tried to overcome some of the shortcomings identified by the surveys of reliability prediction methods on software architectures [12], [14], [15], such as the lack of support for tools and for variability, weak reliability analysis and weak validation of the methods. Hence, our approach addresses most of these issues by providing a tool that supports strong variability with four degrees of freedom and strong validation of our results by comparing them with the methods presented in the literature. In addition, by providing a mathematical model that can be used for verification, simulation and experimentation, architects are now capable of performing a strong reliability analysis.

An automated generation of a mathematical model from an ADL saves software architects the effort to manually build it by providing a correct, accurate and error-free formal model, which describes the system behaviour. In addition, software architects are now able to find alternatives to the architecture thanks to the degrees of freedom provided by our approach. Thus, architects can vary the number of components, their reliabilities, architectural styles and the usage profile of the system, obtaining information about what test suite best fits the stakeholders' requirements regarding the reliability of the system.

In the next steps to extend our work, we will perform a sensitivity analysis to gather information about how the system behaves relatively to the uncertainty of the reliability of a particular component. Thus, it allows pointing out in the architecture what are the reliability hotspots that are influencing the most the reliability of the whole system. We plan to add more styles to our prediction procedure, enriching knowledge to the research community, since there is a notorious lack of studies on this topic. In addition, we plan to join our work with other studies that have predicted different quality attributes, such as performance or maintainability, to have a more complete prediction of the system quality. Therefore, our approach can help avoid undesired or infeasible architectural designs and prevent extra costs in fixing late life cycle undetected problems.

ACKNOWLEDGEMENTS

This research was supported by a grant from the Carnegie Mellon|Portugal project AFFIDAVIT (PT/ELE/0035/2009). The authors would also like to thank the contribution from David Garlan and Bradley Schmerl.

REFERENCES

[1] D. Garlan, R. T. Monroe, and D. Wile, "Acme: An architecture description interchange language," in *Proceedings of CASCON'97*, Toronto, Ontario, November 1997, pp. 169–183.

[2] R. Allen, "A formal approach to software architecture," Ph.D. dissertation, Carnegie Mellon, School of Computer Science, January 1997, issued as CMU Technical Report CMU-CS-97-144.

[3] P. H. Feiler, D. P. Gluch, and J. J. Hudak, "The Architecture Analysis & Design Language (AADL): An Introduction," Software Engineering Institute, Tech. Rep., 2006.

[4] S. S. Gokhale and K. S. Trivedi, "Reliability Prediction and Sensitivity Analysis Based on Software Architecture," *Reliability Engineering*, 2002.

[5] J.-H. Lo, C.-Y. Huang, I.-Y. Chen, S.-Y. Kuo, and M. R. Lyu, "Reliability assessment and sensitivity analysis of software reliability growth modeling based on software module structure," *Journal of Systems and Software*, vol. 76, no. 1, pp. 3–13, Apr. 2005.

[6] W.-L. Wang, D. Pan, and M.-H. Chen, "Architecture-based software reliability modeling," *Journal of Systems and Software*, vol. 79, no. 1, pp. 132–146, Jan. 2006.

[7] V. Cortellessa, "Early reliability assessment of UML based software models," *Electrical Engineering*, pp. 302–309, 2002.

[8] S. Yacoub, B. Cukic, and H. Ammar, "Scenario-based reliability analysis of component-based software," *Proceedings 10th International Symposium on Software Reliability Engineering (Cat. No.PR00443)*, pp. 22–31, 1999.

[9] F. Brosch, B. Buhnova, H. Koziolek, and R. Reussner, "Reliability prediction for fault-tolerant software architectures," in *Proceedings of the joint ACM SIGSOFT conferenceQoSA and ACM SIGSOFT symposiumISARCS on Quality of software architecturesQoSA and architecting critical systemsISARCS*. ACM, 2011, pp. 75–84.

[10] R. Reussner and Heinz W., "Reliability prediction for component-based software architectures," *Journal of Systems and Software*, vol. 66, no. 3, pp. 241–252, Jun. 2003.

[11] R. C. Cheung, "A user-oriented software reliability model bell telephone laboratories, naperville, illinois 60540," *Computer Software and Applications Conference, 1978. COMPSAC '78.*, 1978.

[12] A. Immonen and E. Niemelä, "Survey of reliability and availability prediction methods from the viewpoint of software architecture," *Software and Systems Modeling*, vol. 7, no. 1, pp. 49–65, Jan. 2008.

[13] K. Goševa-Popstojanova and K. Trivedi, "Architecture-based approach to reliability assessment of software systems," *Performance Evaluation*, vol. 45, no. 2, pp. 179–204, 2001.

[14] S. S. Gokhale, "Architecture-Based Software Reliability Analysis : Overview and Limitations," *IEEE Transactions On Dependable And Secure Computing*, vol. 4, no. 1, pp. 32–40, 2007.

[15] D. Pengoria, S. Kumar, and M. S. Se, "A Study on Software Reliability Engineering Present Paradigms and its Future Considerations," *Computing*, 2009.

[16] R. Reussner, "Reliability prediction for component-based software architectures," *Journal of Systems and Software*, vol. 66, no. 3, pp. 241–252, Jun. 2003.

[17] B. Littlewood, "Software reliability model for modular program structure," *IEEE Transactions on Reliability*, vol. R-28, no. 3, pp. 241–246, August 1979.

[18] J. Laprie and K. Kanoun, "Software reliability and system reliability," *Handbook of software reliability engineering*, pp. 27–69, 1996.

[19] K. Goseva-Popstojanova, M. Hamill, and R. Perugupalli, "Large Empirical Case Study of Architecture based Software Reliability," *Reliability Engineering*, 2005.

[20] M. Shaw and D. Garlan, *Software Architecture: Perspectives on an Emerging Discipline*. Upper Saddle River, NJ: Prentice Hall, 1996.

[21] A. Abd-allah, "Extending reliability block diagrams to software architectures," Dept. of Computer Science, Univ. Southern California, Tech. Rep. USC-CSE-97-501, 1997.

[22] A. Martens, H. Koziolek, S. Becker, and R. Reussner, "Automatically Improve Software Architecture Models for Performance , Reliability , and Cost Using Evolutionary Algorithms," *Population (English Edition)*, 2010.

[23] C. M. Grinstead and L. J. Snell, *Grinstead and Snell's Introduction to Probability*, 4th ed. American Mathematical Society, 2006.

[24] D. Garlan, R. T. Monroe, and D. Wile, "Acme: Architectural description of component-based systems," in *Foundations of Component-Based Systems*, G. T. Leavens and M. Sitaraman, Eds. Cambridge University Press, 2000, pp. 47–68.

[25] 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.

[26] M. Kwiatkowska, G. Norman, and D. Parker, "Prism: Probabilistic model checking for performance and reliability analysis," *ACM SIGMETRICS Performance Evaluation Review*, vol. 36, no. 4, pp. 40–45, 2009.

[27] D. Garlan and M. Shaw, "An Introduction to Software Architecture," *Knowledge Creation Diffusion Utilization*, no. January, 1994.

[28] L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice (2nd Edition)*, 2nd ed. Addison-Wesley Professional, Apr. 2003.

[29] L. Cheung, R. Roshandel, N. Medvidovic, and L. Golubchik, "Early prediction of software component reliability," in *Proceedings of the 30th international conference on Software engineering*. New York, New York, USA: ACM, 2008, pp. 111–120.

[30] K. Goseva-Popstojanova and S. Kamavaram, "Software reliability estimation under uncertainty: Generalization of the method of moments," *High-Assurance Systems Engineering, IEEE International Symposium on*, vol. 0, pp. 209–218, 2004.

[31] S. L. Graham, P. B. Kessler, and M. K. McKusick, "gprof: a call graph execution profiler," *SIGPLAN Not.*, vol. 39, no. 4, pp. 49–57, Apr. 2004.

[32] Q. Yang, J. J. Li, and D. Weiss, "A survey of coverage based testing tools," in *Proceedings of the 2006 international workshop on Automation of software test*, ser. AST '06. New York, NY, USA: ACM, 2006, pp. 99–103.

[33] J. F. Groote, T. W. D. M. Kouters, and A. A. H. Osaiweran, "Specification guidelines to avoid the state space explosion problem," *Fundamentals of Software Engineering (FSEN)*, April 2011.