Integration of Architecture Specification, Testing and Dependability Analysis

Swapna S. Gokhale, Joseph R. Horgan
Telcordia Technologies
445 South Street
Morristown, NJ, 07960, USA
swapna,jrh@research.telcordia.com

Kishor S. Trivedi
Dept. of ECE, CACC
Duke University
Durham, NC 27708, USA
kst@ee.duke.edu

ABSTRACT

Software architectural choices have a profound influence on the quality attributes supported by a system. Architecture analysis can be used to evaluate the influence of the design decisions on important quality attributes such as maintainability, performance, and dependability. As software architecture gains appreciation as a critical design level for software systems, techniques and tools to support testing, understanding, debugging, and maintaining these architectures are expected to become readily available. In addition to providing the desired support, data collected from these tools also provides a rich source of information from the point of view of performance and dependability analysis of the architecture. The present paper presents a performance and dependability analysis methodology which illustrates the use of such data. The methodology thus seeks a three way integration of distinct and important areas, namely, formal specification, specification simulation/testing and dependability/performance analysis. We illustrate the important steps in the methodology with the help of an example.

1. INTRODUCTION

As software systems [1] continue to grow in size and complexity, software architecture is increasingly appreciated as a method of understanding and analysis. The architecture of a software system defines its high level structure, exposing its gross organization as a collection of interacting components [4]. Software architecture represents the design decisions that are made in the early phases of a system. These decisions are usually difficult to change or reverse and have a profound influence on the non functional attributes that can be supported by the system. It is becoming increasingly clear that software architecture analysis is the best vehicle to assess important quality attributes such as maintainability, reliability, reusability and performance. Architectural description languages (ADLs) are formal languages that can be used to represent the architecture of a software system. They focus on the high-level structure of the overall system rather than the implementation details of any specific source module. ADLs are intended to play an important role in the development of software by composing source modules rather than by composing individual statements written in conventional programming languages. A number of architectural description languages have been proposed recently, such as Rapide [9], UniCon [13] and WRIGHT [1]. As software architecture design gains prominence, the development of techniques and tools to support understanding, testing, debugging, reengineering, and maintaining software architecture will become an important issue. Li et. al. propose a tool for understanding, testing, debugging and profiling software architectural specifications in SDL [8]. Zhao et. al. propose a technique for architecture understanding based on program dependence graphs for ACME [19]. They also propose a slicing methodology to extract reusable software architectures [20].

Software architectures specified in ADLs such as SDL and LOTOS, and other modeling languages such as UML can also be used for performance and dependability analysis, by constructing quantitative models from these specifications. Wohlin et. al. develop a performance analysis methodology for specifications written in SDL [18]. Marsan et. al. present an extension of LOTOS, which enables a mapping from the extended specifications to performance models [10]. Steppler develop a tool for the simulation and emulation of formation specifications in SDL, with an eye towards analyzing these specifications for the non functional attribute of performance [14]. Heck et. al. describe a hierarchical performance evaluation approach for formally specified communication protocols in SDL [6]. Bondavalli et. al. present a dependability analysis procedure based on UML designs [2]. Wang et. al. discuss a performance and dependability analysis mechanism for specifications in Estelle [17]. The major drawback of the analysis approaches mentioned above is the lack of adequate information to parameterize the quantitative models constructed from the software specifications. As the techniques for software architecture testing become mature and tools become readily available, the data generated during testing can provide a rich source of information for subsequent model parameterization. A similar approach has been demonstrated at the source code level, where execution trace data collected from extensive testing

[1] System and application are used interchangeably in this paper.

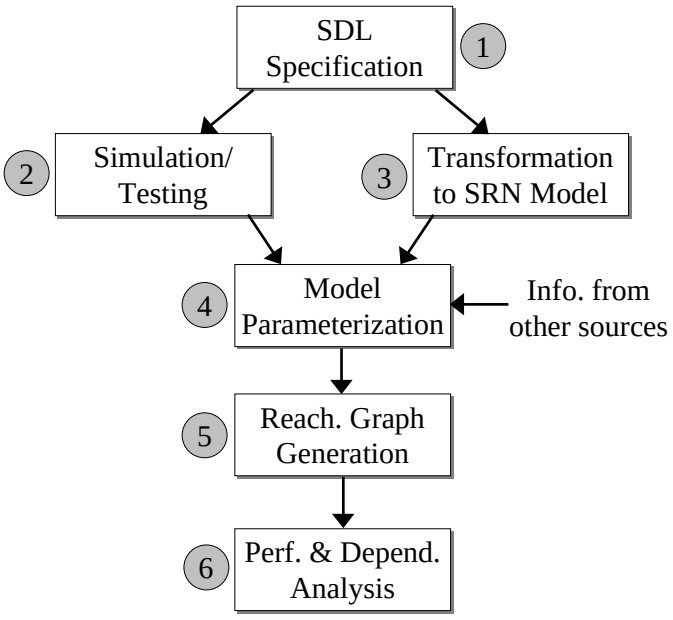

Figure 1: Steps in the methodology

of the application was used to extract and parameterize the architectural model of the system [5]. Our present paper outlines an approach which seeks a three way integration, namely, formal specification/modeling, architecture simulation/testing and performance/dependability analysis. The glue between specification, performance analysis and testing is provided by using the measurements obtained during simulation/testing to parameterize the quantitative model of the system. Our methodology is facilitated by Telcordia Software Visualization and Analysis Tool Suite (TSVAT), developed at Telcordia Technologies for architectural specifications in SDL [8].

We describe the steps involved in the analysis methodology in the extended abstract. The illustration of the methodology to assess the performance and dependability of an application specified in SDL will be presented at the workshop.

2. METHODOLOGY

The methodology centers around a system specified using the Specification and Description Language (SDL) and the quantitative modeling paradigm of Stochastic Reward Nets (SRNs). The steps involved in the methodology can be depicted pictorially as shown in Figure 1. We briefly describe the steps shown in Figure 1 in this section.

2.1 System Specification in SDL

An architecture can be specified using an informal notation (box and arrow diagrams). However, such notations are error-prone and ambiguous. An architecture specified in a formal specification language eliminates the ambiguity and provides a clear basis for analysis. We choose Specification and Description Language (SDL) as a Communicating Extended Finite State Machine (CEFSM) specification language to specify the software architecture. The choice of SDL as an architecture description language is motivated due to the following reasons: i) SDL is an International Telecommunication Union (ITU) standard [15]. Many telecom system software specifications are rendered in SDL, ii) SDL is a formal language with a well-defined semantics. Several commercial off-the-shelf tools can be used to investigate architectural models formalized in SDL, iii) SDL meets the requirements for an executable architectural specification language [9]. SDL allows dynamic creation and termination of process instances and their corresponding communication paths during execution. Hence it is capable of modeling the architectures of dynamic systems in which the number of components and connectors may vary during system execution, iv) SDL can present all four views of software architecture [7]. For example, SDL uses delay and non-delay channels to indicate the relative physical locations of components. The first step in the methodology is the specification of the application to be assessed in SDL [2]. This step is marked as "1" in Figure 1.

2.2 Specification Simulation/Testing

The next step in the methodology (marked "2" in Figure 1) is to simulate/test the system specified in SDL and collect trace data during the simulation process. We use the SDL version of the Telcordia Software Visualization and Analysis Tool Suite (TSVAT) [8], developed to support architectural specification, debugging and testing to collect the trace data. TSVAT contains a suite of tools, χSlice, χVue, χProf, χRegress, and χATAC. We primarily use χATAC in our methodology. The technique underlying this tool suite is the creation of a flow graph of the specification, thus laying out its execution structure. We use the simulator from Telelogic [21] to simulate the SDL specification of the application. The simulator is then instrumented to collect execution traces. The trace file records how many times a given part of the specification, such as a process, a transition, a decision, a state input, or a data flow, has been exercised in each simulation of the specification, or at the end of the testing process. χATAC reports coverage with respect to the following well-known criteria: function coverage, basic transition coverage and decision coverage. Function coverage simply checks that each process of the SDL specification has been executed at least once. A basic transition is simply a statement sequence of the specification that is always executed sequentially, including states and decision constructs (no internal branching constructs). Basic transition coverage checks that each basic transition has been executed at least once, which implies that each statement has been executed at least once. Decisions are conditional branches from one basic transition to another. Decision coverage checks that each such situation, decision matching or input match-

[2]It should be noted that our methodology which proposes a three way integration of architecture specification, simulation/testing and dependability analysis is not limited to the use of SDL. Any architecture specification language which provides capabilities similar to SDL can be used in this methodology.

ing is executed, so that all true and false paths have been taken as well as all input alternatives and decision alternatives.

The execution traces collected during the testing of the specification can then be used to extract branching probabilities of the various decisions in the specification. If the simulation process of the specification is guided by an operational profile [11] of the system, then these branching probabilities would be characteristic of what would be observed in the field. Regression test suites from earlier releases of the product and/or expert opinion could also be used to guide the design of test cases for simulation.

2.3 Translation from SDL Specification to a SRN Model

The SDL representation of the architecture of the system is then translated to a stochastic reward net (SRN) model for performance and dependability analysis. This step is marked "3" in Figure 1. Stochastic reward nets (SRNs) are a generalization of generalized stochastic Petri Nets (SPNs), which in turn are a generalization of stochastic Petri Nets (SPNs) [16]. Stochastic Reward Nets (SRNs) provide the same capabilities as Markov Reward Models [12] which can be used to compute various reliability, safety and performance measures of interest. We define rules to translate an SDL specification at the process level to a stochastic reward net (SRN) model.

2.4 SRN Model Parameterization

The next step in the process is to parameterize the SRN model. This step is marked "4" in Figure 1. The parameters of the SRN model can be broadly classified into five categories, depending on the sources of information used for the parameterization:

- Execution time parameters: These parameters are associated with the execution of the tasks and decisions in the SDL specification, and are heavily dependent on the implementation details. SDL specifications can also be used to generate code in a semi-automatic fashion, and the measurements obtained from the execution of this partial code can be used to determine the distributions and the values of these parameters.
- User inputs: These parameters model the inputs resulting from the actions of the user. These inputs are expected by the system at various stages of execution. The distributions and the actual values of these parameters can be derived from historical data, or can be based on the knowledge of the experts who are intimately familiar with the system and its field usage. In the event that the system is the first of its kind and not much information is available about the system, these parameters can be guestimated.
- Branching probabilities: These reflect the probabilities of occurrence of the various outcomes of a decision. These values can also be derived from historical data, or can be based on expert knowledge. The trace data collected during the simulation/testing of the SDL specification can be used to determine these branching probabilities. The SDL version of Telcordia Software Visualization and Analysis Tool Suite (TSVAT) facilitates the collection of such trace data.
- Inputs from other components/processes: Most real life software systems are inherently distributed, and hence require interactions between the various components of the system. Since the SRN model is constructed from process level specification, it is natural for some of the parameters of the model to draw from the execution of the other processes in the system.
- Failure/repair parameters: These parameters model the failure/repair behavior of the processes and/or each task within a process and are necessary to compute various measures such as the reliability, availability and safety of an application. This information can be obtained by consulting with experts who are familiar with the application or can be guestimated.

2.5 Reachability Graph Generation

The next step in the analysis methodology is the generation of a reachability graph, and the step is marked "5" in Figure 1. The reachability graph of a Petri net is the set of states that are reachable from the other states. The reachability graph is generated using the tool SPNP (Stochastic Petri Net Package) [3], which is also used for performance and dependability analysis. The Stochastic Petri Net Package is a versatile modeling tool for the solution of stochastic Petri net (SPN) models . The SPN models are described in the input language for SPNP called CSPL (C-based SPN language). The CSPL is an extension of the C programming language with additional constructs which facilitate easy description of SPN models. The SPN models specified to SPNP are actually "SPN Reward Models" or Stochastic Reward Nets (SRNs), which are based on the "Markov Reward Models" paradigm. This provides a powerful modeling environment for dependability (reliability, availability, safety), performance and performability analysis.

2.6 Dependability Analysis

The parameterized stochastic reward net (SRN) model of the application can then be used for performance and dependability analysis. The Stochastic Petri Net Package (SPNP) can be used to compute various measures of interest such as the reliability, availability and safety [3]. The performance and dependability analysis step is marked "6" in Figure 1.

3. CASE STUDY

We illustrate the steps described in the previous section with the help of a PBX system specified in SDL. Figure 2 shows the block level specification of the PBX system in SDL.

The PBX system consists of two distributed blocks. The block *CallHandler* controls the call processing functions and the block *ResManager* involves inventory control and remote database access. The *CallHandler* block receives three signals over channel $c1$, namely, *offhook*, *dig* and *hangUp*. The first signal announces the caller's intent to place a call,

the second signal is a digit of a telephone number and the hangUp signal is sent either after the call is complete, if there is a change in the caller's intention to pursue the call, due to the unavailability of the resources required to place a call, or due to the unavailability of the callee to accept the call (callee is busy). The *CallHandler* block outputs the signals *dialT*, *busyT*, *connt*, *ringT* and *ring* over the channel $c2$. Communication between the *CallHandler* block and the *ResManager* block occurs over channels $c3$ and $c4$. Channel $c3$ is used to communicate the request and release messages for resources and channel $c4$ is used to send reply messages regarding the availability of the resources to proceed with the call. Channels $c3$ and $c4$ are delaying channels which indicates that the two blocks can be implemented on different CPUs with a non-negligible delay. This reflects the reality that the database information can be stored remotely. The process level specification of the *CallHandler* and *ResManager* blocks as well as the other steps in the methodology will be presented at the workshop.

4. CONCLUSIONS AND FUTURE

In this paper we present a methodology which integrates three distinct areas in architecture design: specification, simulation/testing, and performance/dependability analysis. Our future research includes extending the methodology to analyze other non functional attributes such as maintainability and flexibility.

5. REFERENCES

[1] R. Allen. *"A formal approach to software architecture"*. PhD thesis, Dept. of Computer Science, Carneige Mellon University, Pittsburgh, NC, 1997.

[2] A. Bondavalli, I. Mura, and I. Majzik. "Automated dependability analysis of UML designs". In *Proc. of Second IEEE International Symposium on Object-Oriented Real-Time Distributed Computing*, 1998.

[3] G. Ciardo, J. Muppala, and K. S. Trivedi. "SPNP: Stochastic Petri Net Package". In *Proceedings of the International Workshop on Petri Nets and Performance Models*, pages 142–150, Los Alamitos, CA, December 1989. IEEE Computer Society Press.

[4] D. Garlan and M. Shaw. *Advances in Software Engineering and Knowledge Engineering, Volume 1, edited by V. Ambriola and G. Torotora*, chapter An Introduction to Software Architecture. World Scientific Publishing Company, New Jersey, 1993.

[5] S. Gokhale, W. E. Wong, K. S. Trivedi, and J. R. Horgan. "An analytic approach to architecture-based software reliability prediction". In *Proc. of Intl. Performance and Dependability Symposium (IPDS '98)*, pages 13–22, Durham, NC, September 1998.

[6] E. Heck and D. Hogrefe. "Hierarchical performance evaluation based on formally specified communication protocols". *IEEE Trans. on Computers*, 40(4):500–513, April 1991.

[7] P. B. Krutchen. "The 4+1 view model of architecture". *IEEE Software*, pages 42–50, November 1995.

[8] J. J. Li and J. R. Horgan. "A toolsuite for diagnosis and testing software design specifications". In *Proc. of Dependable Systems and Networks (DSN2000, FTCS31)*, New York, NY, June 2000.

[9] D. Luckham, L. A. Augustin, J. Kenney, J. Veera, D. Bryan, and W. Mann. "Specification and analysis of system architecture using rapide". *IEEE Tran. on Software Engineering*, 21(4):336–355, April 1995.

[10] M. A. Marsan, A. Bianco, L. Ciminera, R. Sisto, and A. Valenzano. "A LOTOS extsnsion for the performance analysis of distributed systems". *IEEE/ACM Transactions on Networking*, 2(2):151–165, April 1994.

[11] J. D. Musa. "Operational profiles in software-reliability engineering". *IEEE Software*, 10(2):14–32, March 1993.

[12] A. Reibman, R. Smith, and K. S. Trivedi. "Markov and Markov Reward Model Transient Analysis: An Overview of Numerical Approaches". *European Journal of Operational Research*, 40:257–267, 1989.

[13] M. Shaw, R. DeLine, D. V. Klein, T. L. Ross, D. M. Young, and G. Zelesnik. "Abstractions for software architecture and tools to support them". *IEEE Trans. on Software Engineering*, 21(4):314–335, April 1995.

[14] M. Steppler and B. Walke. "Performance analysis of communications systems formally specified in SDL". In *Proc. of the Workshop on Software and Performance*, Santa Fe, NM, 1998.

[15] International Telegraph and Telephone Consulative Committee. *"SDL User Guideliness"*. International Telecommunication Union, November 1989.

[16] L. A. Tomek and K. S. Trivedi. *Software Fault Tolerance, Edited by M. R. Lyu*, chapter Analyses Using Stochastic Reward Nets, pages 139–165. John Wiley and Sons Ltd., New York, 1995.

[17] C. Y. Wang and K. S. Trivedi. "Integration of specification for modeling and specification for system design". In *Proc. of Fourteenth Intl. Conference on Applications and Theory of Petri Nets*, pages 24–31, 1993.

[18] C. Wohlin and D. Rapp. "Performance analysis in the early design of software". In *Proc. of Intl. Conference on Software Engineering for Telephone Switching Systems"*, pages 114–121, 1992.

[19] J. Zhao. *New Technologies on Computer Software, M. Li, Editor*, chapter "Using Dependence Analysis to Support Software Architecture Understanding", pages 135–142. International Academic Publishers, September 1997.

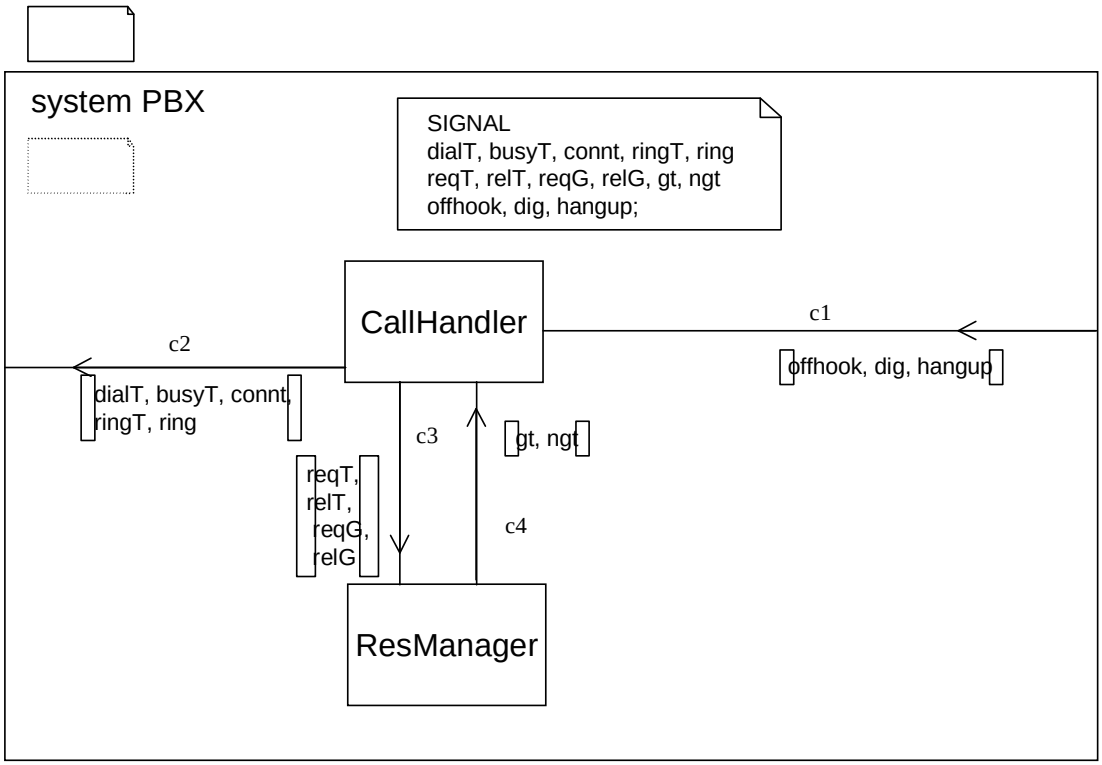

Figure 2: Block level SDL specification of a PBX system

[20] J. Zhao. "A slicing-based approach to extracting reusable software architectures". In *Proc. of 4th European Conference on Software Maintenance and Reengineering*, pages 215–233, Zurich, Switzerland, February 2000.

[21] *http://www.telelogic.com.*