Experimental Frame Design Using E-DEVSML for Software Quality Evaluation

Bei Cao, Linpeng Huang, Jianpeng Hu
Dept. of Computer Science and Engineering
Shanghai Jiao Tong University
Shanghai,China
Email: caobei.sjtu@gmail.com, Huang-lp@sjtu.edu.cn, mr@sues.edu.cn

***Abstract*—Quality evaluation is a critical aspect in the area of software development. If software quality problems could be found in the early design phase, the cost for software development and maintaining will be reduced. In this paper we propose an evaluation framework including a software error model and its corresponding experimental frame, which is based on Discrete Event System Specification (DEVS), to support the evaluation of multiple quality properties in the design phase. To accelerate the modeling and simulation processes, we further extend E-DEVSML to create model of system under evaluation and its experimental frame, and transform them to executable models automatically. A case study of a ticket booking system is presented to demonstrate that our approach is applicable.**

***Keywords*—System of Systems, DEVS, quality evaluation**

I. INTRODUCTION

Automated software modeling and simulation tools are by far the most promising approach to lowering the cost of software development. For successful project managements, it is very important to validate FRs and evaluate NFRs of systems precisely in early design phase before implementation of the systems [2]. Consequently the executable architectures are commonly defined to be executable dynamic simulations that are automatically or semi-automatically generated from static architecture models or products [1]. In an executable evaluation process, one of the most important issues is the design of the Experimental Frame (EF). An EF is a specification of the conditions under which a system is observed or experimented with [6]. The EF can be viewed as a system that interacts with the system of interest or system under test to obtain data under specified conditions. In this way, for an early evaluation of software quality we should concentrate on the behavior description of the system under test and the design of EF related to corresponding quality concerns, and interactions between simulated system model and EF are consequently captured for analysis and evaluation.

In this paper we propose a generic simulation approach to software quality evaluation based on a Discrete Event System Specification (DEVS) simulation framework to aid architects in the analysis of software quality attributes. First, to accelerate the modeling and simulation processes, we further extend our formal work E-DEVSML [2] to facilitate the design of experimental frame of quality evaluation. Second, an error model is given to depict behavior of the system under test for quality concerns. At the last, we use a case of ticket booking system to demonstrate our approach.

The rest of this paper is organized as follows. Section two extends E-DEVSML for EF modeling. Section three presents our DEVS-based software quality evaluation model. A case study of a ticket booking system is present to explain how our approach worked in section four. In section five, we make our conclusion and discuss the future direction.

II. EF SPECIFICATION IN E-DEVSML

When we use DEVS to simulate a system, a critical part is to design the experimental frame (EF), which controls the simulation process. A DEVS experimental frame is often composed of three parts: acceptor, generator and transducer. The acceptor controls the beginning and end of the simulation, the generator sends requests applied to the system or model, and the transducer observes and analyzes the system output. As creating EF is a necessary part of the simulation process, reducing the complexity of writing EF programs will accelerate the DEVS modeling process. Therefore, we extend our E-DEVSML [2] to support the EF modeling. Thus EF models can be created in E-DEVSML and automatically transformed to executable languages through Xtend [4]. The detail of our method will be introduced in the following sections.

A. Acceptor

The function of an acceptor is to control the beginning and the end of simulation. The DEVS model of an acceptor is shown in Fig.1. An acceptor contains an output port to send start or stop messages. In the beginning of simulation, the acceptor stays in passive state for a certain time. After that time it sends a start message through control port and changes its state to simulating. Then the acceptor sends a stop message after a certain time simulation. The key information that is needed for us is the simulation time, wait time and the output port name which is used for the coupling with other models. The abstract syntax of an acceptor in E-DEVSML is presented in Fig. 1. We define an acceptor with keywords acceptor, extends,waitTime, simTime, and control. A defined acceptor can be extended by another acceptor using keyword extends. To make the layout of the simulation model clear, we define the acceptor as a coupled model, which contain several sub-acceptors. As its external transition function, internal transition function and output function are in the same form which can be encapsulated, we can automatically get these functions through Xtend [4].

The work described in this paper was supported by the National Natural Science Foundation of China under Grant No.61232007 and No.91118004.

(DOI reference number: 10.18293/SEKE2015-169)

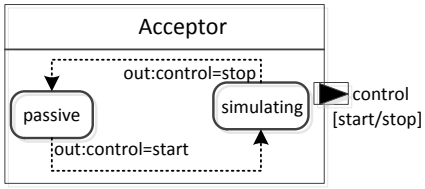

Fig. 1: Acceptor model

B. Generator

The DEVS model of a generator is shown in Fig. 2. A generator contains two ports: an input port to receive control messages for start or stop the generator, and an output port to generate requests. Requests are generated in a random way following a probability distribution. The abstract syntax of a generator is presented in Fig. 4. We define a generator with keywords generator, extends, control, out, distribution. We also defined some classical distribution types in the grammar, including poisson distribution, normal distribution and uniform distribution. We can choose an appropriate distribution suited for actual situation or define a function by ourselves.

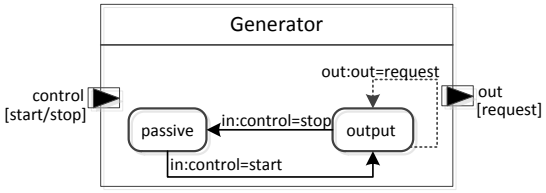

Fig. 2: Generator model

C. Transducer

A transducer receives messages generated by the simulation system and analyzes them through some calculations. The DEVS model of a transducer is shown in Fig. 3. A transducer contains several ports: an input port to receive control messages for start or stop the transducer, a set of input ports to receive messages generated by the simulation system, and a set of output ports to send the analysis results. We define the grammar of a transducer with keywords transducer, extends, vars, control, in, out in Fig. 4. Every input port of a transducer is binding with some codes which describes the behavior when a message comes to this port. Every output port of a transducer is binding to a variable which records the analysis result. The transducer is also defined as a coupled model and can be extended by another transducer.

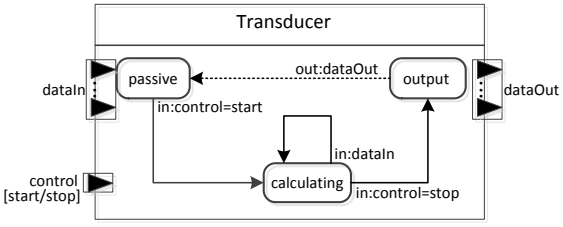

Fig. 3: Transducer model

E-DEVSML.xtext

```
Acceptor:
'acceptor' name=ID ('extends' superType=[Acceptor])?
'{'(subAC +=SubAcceptor)*'}';

SubAcceptor:
name=ID '{''waitTime' wt=DOUBLE  'simTime' st=DOUBLE
'control' str=STRING'}';

Generator: 'generator' name=ID
('extends'superType=Generator )?
'{' (subG +=SubGenerator)*'}';

SubGenerator: name=ID
'{''control' in=STRING 'out' out=STRING
'distribution' '{' distributionType=Dtype
'('exp=DOUBLE','sigm=DOUBLE')'|'double' name=ID
'('')' code=Code '}' '}';

Dtype:
type=('Poisson' | 'Normal' | 'Uniform');

Transducer: 'transducer' name=ID
('extends' superType=[Transducer])?
'{' (subT += SubTransducer)*'}';

SubTransducer: name=ID '{'
'vars''{'(variables += Variable)*'}'
'control' start=STRING
("in" port=STRING code=Code)*
("out" port=STRING var=[Variable])*
'}';
```

Fig. 4: The Grammar of EF Defined in EBNF

III. SOFTWARE QUALITY EVALUATION MODEL

In this section, we propose our DEVS-based software quality evaluation model. The quality evaluation model is composed of two parts: Error Model (EM) and EF. An EM describes the dynamic behavior of software or software component with parameters generated in the runtime. The EF controls the simulation process and calculates the quality metrics.

Fig.5 describes the behavior of a DEVS-based Error Model of system under evaluation with seven states: passive, active, executing, failed, recovering, error, and reboot. The passive state represents the model is waiting requests from req input port. When the request come the state changes to active. Then a message will be sent through state port. EM is in executing phase when it is processing a request. The failed state represents that a failure event happened. The recovering state represents that the model is recover from a failure. The error state represents that a fatal error event happened which leads to system reboot. The reboot state represents that the system is rebooting. An EM is in passive state before the simulations beginning. If a request comes, the state will change to active and an activated message will generate through state port to activate the failure generator and error generator. Then the state change to executing. If the request queue is not empty, the EM will keep processing the request. When a request is finished, the execution time will be sent through et port. If the request queue is empty, the state will go back to passive. If a failure comes in the executing state, the state will change to failed. After a downtime, the system starts to recover. EM comes to recover state. After a recover time, the state returns to executing. If an error comes in the executing state, the state will change to error. The system starts to reboot immediately. After a reboot time, the state returns to executing. Some messages will be produced in an internal transition as shown in Fig.5. We will capture these messages with our experimental frame.

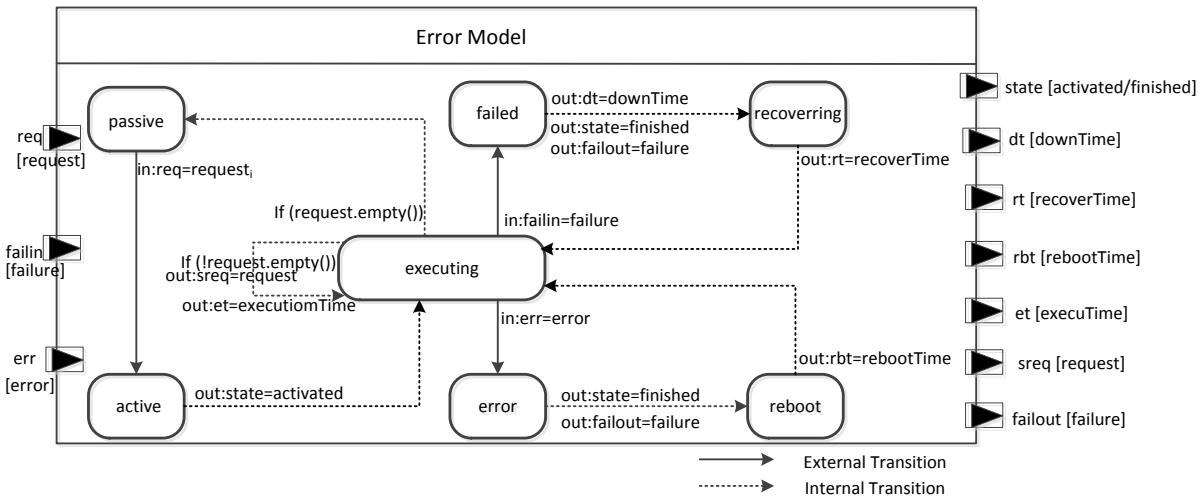

Fig. 5: Error model

We defined the corresponding EF of the error model with one acceptor, three kinds of generators and three kinds of transducers. The acceptor controls the start and the stop of the simulation. Generators are composed of request generators, failure generators, and error generators. The request generator sends request to the req port, the failure generator sends failure to failin port. The error generator sends error to the err port. Transducers are composed of performance transducers, reliability transducers and availability transducers. The performance transducer receives message from the sreq port and counts the total number of requests finished by the system. The reliability transducer receives message from failout port and counts the failures happened in the simulation process. The availability transducer receives message from port dt, rt, rbt, and et. It counts the available time and unavailable time of the system.

IV. Case Study

In this section, we use a ticket booking system to explain how to design an experimental frame in E-DEVSML to evaluate software quality. Eclipse Xtext is used to specify E-DEVSML and transform E-DEVSML models to Java codes for DEVS-suite [5] which is a popular DEVS simulator. We will run the simulation and get the evaluation results in DEVS-suite.

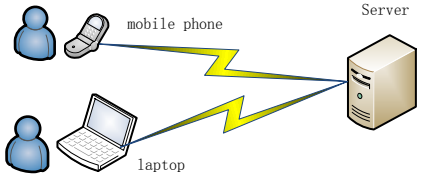

Fig. 6: A ticket booking system

A. A ticket booking system

The example ticket booking system (Fig. 6) is composed of two clients (laptop and mobile phone) and a server. Customers may use these clients to request a service, such as querying available tickets, booking movie tickets, online payment. The server answers the requests and sends the results to the clients. The server may stop providing the service if some failures or errors happen, e.g., it cannot handle any more requests or the system crashed. The clients may also experience some failures, e.g., the mobile phone lost the signal, the laptop crashed. We create DEVS models in E-DEVSML and evaluate the quality of the laptop and phone and compare the results.

TABLE I: Experimental settings

	mobile phone	laptop	server
etime(s)	uniform (0.3, 0.6)	uniform (0.2,0.4)	uniform (0.1,0.15)
rtime(s)	1	2	0.5
rbtime(s)	10	30	20
dtime(s)	0.3	0.2	0.1
failure rate	0.06	0.05	0.02
error rate	0.015	0.01	0.008
Request frequency	funiform (1.5, 3)	uniform (1, 2)	depends on clients

B. Experiment settings

Parameters related to the quality aspects must be set before the simulation based on the behavior of previous systems, historical data and software experiences. To make the case easy to understand, we set these parameters in table I according to our experiences and set the simulation time to 3 days. In actual situation, the experimental settings could be more complex. In table I, etime represents the time to process a request, rtime represents the time needed to recover from failed state, rbtime represents the reboot time of the software system. dtime represents the down time. The word uniform represents the uniform distribution. As the clients depend on the server, the failure of server will cause the failure of clients. Although we only care about the quality properties of clients, we still have to consider the parameters of the server.

C. E-DEVSML modeling

To evaluate the quality of the system, we should create the evaluation model first. We define the ticket booking system in E-DEVSML with experimental settings.Our evaluation model

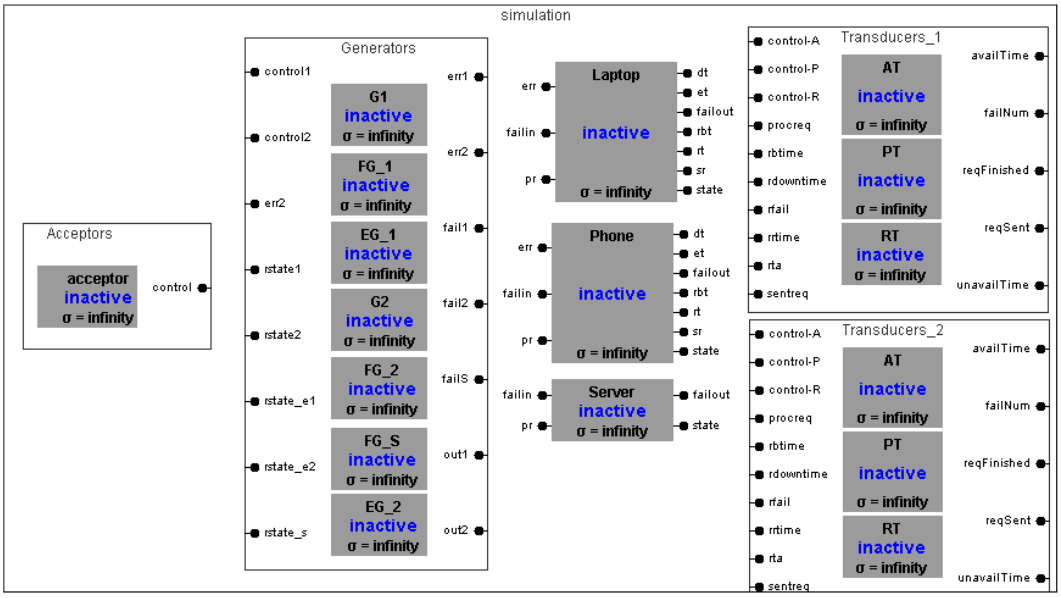

Fig. 7: Simulation model

TABLE II: Simulation results

	mobile phone	laptop
turnaround time	51319.3	50192.7
Finished requests	112155	170407
Sent requests	115274	172835
fail number	3119	2428
unavailable time	4054.7	5341.6

TABLE III: Quality properties of different clients

	Phone	Laptop
Reliability (MTBF)	83.1 s	106.8 s
Availability (available time/total time)	98.44%	99.10%
Performance (Average turnaround time)	2.31s/request	1.52s/request

is composed of three atomic models, an acceptor, a coupled generator, and two coupled transducer. We define the couplings and input ports and output ports of the simulation system in a coupled model. As E-DEVSML can be transformed to executable language through Eclipse Xtend, we define some mapping rules through Xtend and transform our E-DEVSML model to Java code which can be run on the DEVS-suite [5]. The simulation model after transformation is shown in Fig 7 .

D. Simulation results Analysis

By tracking the output ports of transducers, we can get the quality properties of the system. After the simulation, we can get the turnaround time, finished requests, sent requests, fail number and unavailable time. The results we get from these out ports are shown in table II. We can calculate the classical quality properties of the system through the data we get. For example the mean time between failures (MTBF), the availability and performance. Table III presents some quality properties calculated by transducer. By comparison of the quality properties of mobile phone and laptop system, we found that the laptop provide better service than the mobile phone.

V. Conclusions and Future Work

In this paper, we further extend our E-DEVSML with EF specification. To describe the behavior of the system, the error model presented here is general and representative, making the modeling and simulation of software system for quality evaluation more convenient.Our work in this paper has the following features: 1) the error model of system under evaluation considers not only the functional requirements, but also takes into account non-functional concerns, including failures and errors. 2) EF specification in E-DEVSML improves the modeling efficiency and the reusability of DEVS models. 3) The automated code generation improves the automation of model based evaluation process and reduces the burden of analyzers. Although E-DEVSML provides strong modeling abilities, the text based modeling approach also brings in complexity. Our future work is to enable transformation between some graphical modeling languages and E-DEVSML for EF modeling.

References

[1] Hu J, Huang L, Cao B, et al. Extended DEVSML as a Model Transformation Intermediary to Make UML Diagrams Executable[C]. SEKE, 2014.

[2] Hu J, Huang L, Cao B, et al. Executable Modeling Approach to Service Oriented Architecture Using SoaML in Conjunction with Extended DEVSML[C].Services Computing (SCC), 2014 IEEE International Conference on. IEEE, 2014: 243-250.

[3] Sharma V and Trivedi K. Quantifying software performance, reliability and security: an architecture-based approach. J Syst Software2007; 80: 493-509.

[4] Bettini L. Implementing Domain-Specific Languages with Xtext and Xtend[M]. Packt Publishing Ltd, 2013

[5] Kim S, Sarjoughian H S, Elamvazhuthi V. DEVS-suite: a simulator supporting visual experimentation design and behavior monitoring[C].Proceedings of the 2009 Spring Simulation Multiconference. Society for Computer Simulation International, 2009: 161.

[6] B.P. Zeigler, H. Praehofer, T.G. Kim, Theory of Modeling and Simulation, Academic Press, 2000.