JACET Journal of Advances in Computer Engineering and Technology, 3(1) 2017

Availability evaluation of Software architecture of object oriented Style using coloured Petri nets

Abdolghader Pourali[1]

Received (2016-10-24)
Accepted (2017-01-03)

Abstract — **Software architecture is one of the most fundamental products in the process of software development in the areas of behavioral or non- behavioral features like availability or transformability change. There are different ways to evaluate software architecture one of which is the creation of application model. An executable model of software architecture is an official description of architecture which if used prior to the running of architecture, its final behavior and function will be observed and as a result possible problems could be elevated and promoted. In this study we aimed at availability evaluation in object- oriented style. To ensure the applicability of the style the UML diagrams, especially the sequence diagram, were used to exhibit the architectural behavior. In the later stages, as the UML diagram is inapplicable, the following operations were done. First, metric annotation is used to tag clichés. Then, the studied style diagram was transformed into an applicable one. Afterwards and following the design of petri, using CpnTools, the applicable model based on color petri net was evaluated. In this research the availability evaluation on an ATM for the N=5 users was tested and the results of evaluation showed that the higher the rate of availability (approximately %100) the higher is the rate of usability of the system when needed.**

Keywords: **Software architecture, integrated modeling language, sequence diagram, colored Petri net (CPN), availability, object-oriented style.**

1- Islamic Azad University, Abadan Branch, Abadan, Iran. (pourali_ghader@yahoo.com)

I. INTRODUCTION

In the present era, as information technology is advancing, software architecture researchers are making efforts to develop better software systems and have achieved success in the field. In fact, better software systems are developed by modeling important architecture patterns in the initial phases of the life cycle of software development. Software architecture is a significant index of qualitative characteristics. The selection of patterns that should be modeled and the way modeling is performed and, finally, evaluating the patterns are important decisions upon which software architecture researchers have focused. As a matter of fact, the developer can design the software by software modeling before allocating extra sources, returning to the stage prior to design, namely requirement determination, and making sure of a reliable system. In addition to describing software and dividing it into components, software architecture styles substantially affect the qualitative characteristics of the designed software. Hence, it is extremely necessary to select a suitable architecture style for the system [1] because incorrect decisions cost considerable money and time spent for design in addition to preventing the achievement of desirable qualitative characteristics. Failing to conduct a quantitative analysis of the effect of software architecture styles on qualitative characteristics does not allow architecture styles to be used effectively. The reason lies in the fact that the decisive factor in selecting software architecture is the extent to which it supports the respective qualitative characteristics. Therefore, utilizing a technique or model for the quantitative evaluation of qualitative characteristics in styles

enables system developers to make design decisions with greater accuracy. The evaluation of software system architecture aims at analyzing architecture in order to identify potential risks and make sure that the qualitative requirements are met in the design. As a result, it is absolutely necessary to make decisions as to the selection of an appropriate architecture for the system and its similar subsystems during software system development [1].

Various state-based evaluation techniques have been proposed. In state-based models, the software architecture behavior is modeled as an interaction among components. For example, an evaluation technique was put forward in [2] using discrete-time Markov model that calculates software architecture reliability given the reliability of each component and the probability of transitions. In [3], to conduct quantitative evaluation, a technique based on dynamic Bayesian network was suggested. For the quantitative evaluation of efficiency in software architecture, a method was developed in [4] that functions according to discrete-time Markov model and calculates software architecture efficiency in respect of the time spent in each component and the number of times the components are met during the execution of the program. In [5], to conduct a quantitative evaluation of security on discrete-time Markov model, a technique was put forward that calculates software architecture security on the basis of component vulnerability and the number of times the components are met while the program is executed. Furthermore, to evaluate security, a model was presented in [6] on the basis of statistical Petri nets. In [7], a technique was put forward for the quantitative evaluation of availability, efficiency, and security in software architecture according to colored Petri nets (CPNs) .In [16] Reliability evaluation of a payment model in mobile-commerce using colored Petri net was provided.

Software architecture consists of a set of components, the relationships between them, and features of the components that may be observed from outside. Different methods may be adopted to represent these components in a variety of architecture styles. The architecture style is a set of design rules demonstrating the components and connectors that may be used to develop a system. Software architecture is evaluated in this research by studying software architecture styles and selecting one of the styles under study. The aforesaid style is the object-oriented style. In this style, data representation and the operation that is in connection with it are enclosed in an object. The components of this style are objects that are in connection with each other via calling functions [8].

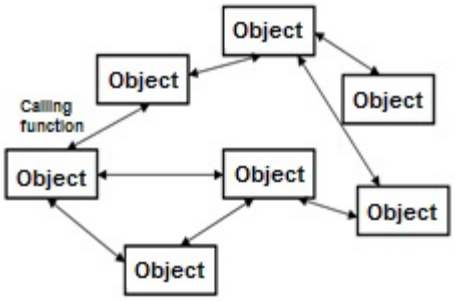

Figure1 – Object-oriented style

Components:

- Object: Components that interact via calling procedures.
- Call-return connectors: They are used by an object to call the procedures of other objects.

To arrive at a precise evaluation of important non-functional and metric requirements, such as availability, an executable model should be put forward. In fact, architecture may be evaluated using this executable model and its problems may be resolved before spending money and time for implementation. UML diagrams are adopted to show the behavior of architecture. The sequence diagram indeed displays system behavior. Whereas UML diagrams may not be executed, excitability should be demonstrated using colored Petri nets. To evaluate the respective metric using Writing Stereotypes, the intended labels should be added so that an evaluation of the architecture would be obtained using CPNTools. In this way, the setbacks would be precluded before the execution and implementation phases. The structure of the rest of the paper is as follows: Section Two addresses the fundamental concepts, statement of the problem, and the presented solution. Section Three reviews the highlights of the study briefly and describes the results. Section Four presents a case study for evaluating

and modeling the presented technique to show its accuracy. Section Five mentions the main, important research results.

II. AVAILABILITY EVALUATION IN SOFTWARE ARCHITECTURE ON OBJECT-ORIENTED

In addition to describing software and dividing it into components, software architecture styles substantially affect the qualitative characteristics of the designed software and amount of support each architecture style provides qualitative characteristics is different. Thus, an appropriate architecture style should be selected in order to attain desirable qualitative characteristics in the system. In general, in IEEE 1990 there are different definitions for quality which are as follows [9].

1- The rate of success a system, a component or a process gains in facing a group of definite requirements.

2- The rate of success a system, a component or a process gains in facing a group of users' expected requirements. From among conventional architecture styles, object-oriented style is chosen in the proposed method. Whereas object-oriented architecture style can use UML modeling language for modeling and architecture products may be generated using UML modeling language, UML is utilized in this technique for architecture documentation. Moreover, colored Petri nets, though simple, are mathematically well-supported, and their powerful support tools such as CPNTools are used to develop an executable model [21].

To develop an executable model, UML diagrams should be at first converted to color Petri nets. Whereas sequence diagram is adopted from among UML diagrams in this paper, UML diagrams are converted to colored Petri nets using the presented algorithm in what follows and the system is evaluated using availability -related Annotations. Using model simulation results and analyzing them, the problems were identified in the planning phase and the products were modified so that huge economic costs and time for implementation would be avoided. Further, the way availability is evaluated on object-oriented style using Petri nets is described.

1. Software Architecture Description

Various models are adopted to describe the architecture of a software system from the viewpoint of software engineers. Architecture itself consists of two parts, namely static and dynamic. The static part of architecture describes software components and the relationship between them. The dynamic part demonstrates the behavior at the time of system execution. Sequence diagrams are utilized to describe system behavior. Sequence diagram describes interactions that are a single behavioral unit which concentrates on exchanging visible information between elements. Information exchange is done via messages and the elements participating in the interaction are displayed with floating lines and messages with arcs [10].

2. Unified Modeling Language

Integrated modeling language, which was introduced by the object management group, is a standard, quasi-official language to easily describe software architecture [11]. This language has introduced a powerful set of modeling elements as well as predefined diagrams and structures to describe the structural and behavioral features of software architecture and appropriate tools to support it. The software may be described at various levels of abstraction and with different outlooks using various diagrams of integrated modeling language.

3. Sequence Diagram

Message sequence diagram is a language that models the interaction between components and processes as well as between samples and the environment [12]. In this scenario-based model, the relationship between samples, namely, sending and receiving messages, local events, and their order, is described. While providing a detailed definition of the system, this language imposes limitations on the transferred data values and the time of events. Moreover, message sequence diagram has a graphical representation. The lifetime of each sample is shown on the vertical dimension of the diagram and a message is demonstrated as a horizontal arrow between the sender and the receiver [12].

4. Colored Petri Nets

CPNs present a clear, graphical representation of the system along with a mathematical approach. They can be indicative of communication patterns, control patterns, and information flows. These nets provide a framework for analysis, validation, and evaluation of efficiency. Petri nets are based on graphs. It may be unofficially stated that a Petri net is a two-part directional graph composed of two elements, namely place and transition. These nets are status-based and not event based, allowing them to explicitly model status in each case. Petri nets present models of structural and behavioral aspects of a discrete event system [18]. They also provide a framework for analysis, validation, and evaluation of efficiency and reliability [13].

5. Object-Oriented style

This style is a modern version of call and return architectures. The object-oriented paradigm relies on the data package and knowledge of the way the operations are carried out and the data is accessed. Package includes Encapsulation data and hiding the internal keys of data from the environment. Accessing an object is only possible via a particular operation called method. Enclosing increases reusability and variability because topics are separated in this course. For instance, the user of service does not need to know how the service works. The main attribute of the object-oriented paradigm which is indeed what distinguishes it from various types of abstract data, is inheritance and polymorphism. When abstractions of an object build a component that supplies black box services and other components use the first group, the call-based client-server style comes into being.

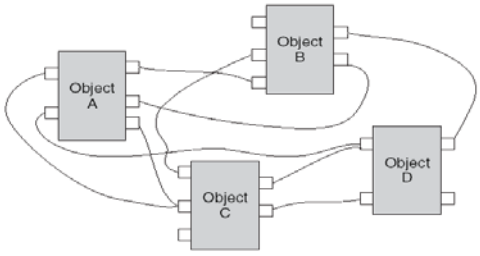

Figure2 – Object-oriented software style from [8]

6. Algorithm for Converting UML Sequence Diagrams to CPN

This diagram lays emphasis on the communication pattern between the components; namely, the interaction between the components. It is plotted in light of the time of sending messages [17]. This paper utilizes a conversion method on the message sender and receiver objects and various types of messages in sequence diagram such as order, selection, parallelism, and iteration. In this case, each message receiver and sender component is converted to place-transition-place. For example, in figures 3, 4, 5 and 6 the conversion of sequence diagram and its equivalence in Petri Net is shown. [14], [17], [18], [21].

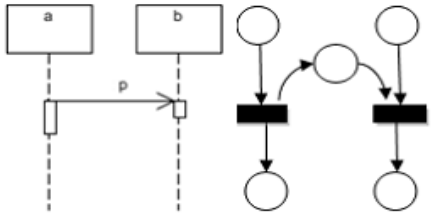

Figure3 – Non-synchronous message and the equivalent colored Petri net

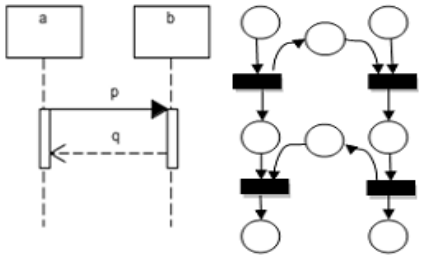

Figure4 – Synchronous message and the equivalent colored Petri net

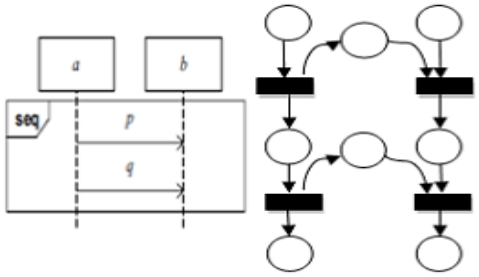

Figure5 – Order structure and the equivalent colored Petri net

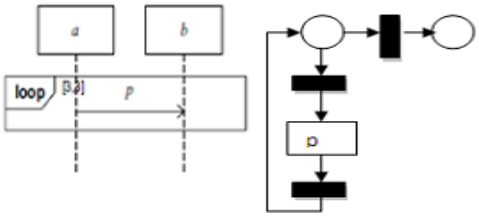

Figure6 – Iteration structure and the equivalent colored Petri net

7. Annotations associated with availability

This paper introduces a description of a set of profiles to evaluate availability. These profiles are added as clichés and labels to integrated modeling language diagrams that describe architecture. The profiles provide little quantitative information on integrated modeling language models. With this information, quantitative analysis may be conducted. In terms of availability, special elements and structures are used in sequence diagram to model system load. In the first case, it is supposed that the time elapsed for sending messages is not very important. A condition, which indicates the probability of the message being sent, is assigned to each message in the diagram. Moreover, a number of messages to which conditions are assigned can be sent from the same point and can be executed in parallel. In case the messages are in an iteration structure, they may be sent several times. This diagram is used to evaluate efficiency, work load, and delay of each message [15], [21].

8. Calculating the availability Metric

To calculate source availability, the average time between source error and its modification is available at a time interval. On this basis, the respective metric may be obtained. Availability may be calculated according to Equation (1) [15], [20].

$$\text{Availibility} = \frac{\text{MTTF}^2}{\text{MTTF} + \text{MTTR}^3} \quad \text{Equation (1)}$$

Both parameters involved in Equation (1) depend on architecture. Average error time typically increases when architecture with high error tolerance is developed, whereas error tolerance is attained by the iteration of important processing elements and connections in the architecture. The next parameter is the average modification time. The better the architecture, the average failure time increases and adding processing elements affects architecture when a crisis occurs.

III. DEVELOPING AN EXECUTABLE MODEL

Colored Petri nets and CPNTools are employed to develop an executable model. In fact, an executable model of architecture is an official description of architecture with which the final behavior may be evaluated, the problems and inefficiencies may be identified, actions may be taken for implementing architecture with higher confidence, and additional costs and even failure may be avoided before implementing architecture. So far, the way a model converted to an executable model is known and the process is observed, the plotted model available should be modeled in the simulator.

Using the algorithm for converting UML diagram to Petri nets, the Petri net of the account withdrawal scenario is as follows. Assuming customer to be equal to C1 and card reader and keyboard, ATM screen, account, cash dispenser, and central database to be respectively equal to C2, C3, C4, C5, and C6, we have:

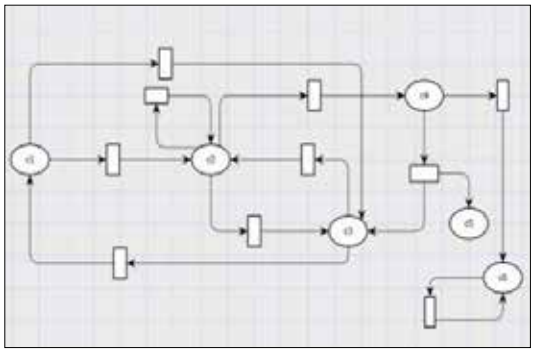

Figure7 – Executable model of account withdrawal scenario

2- Mean Time –To- Failures
3- Mean Time- To- Repair

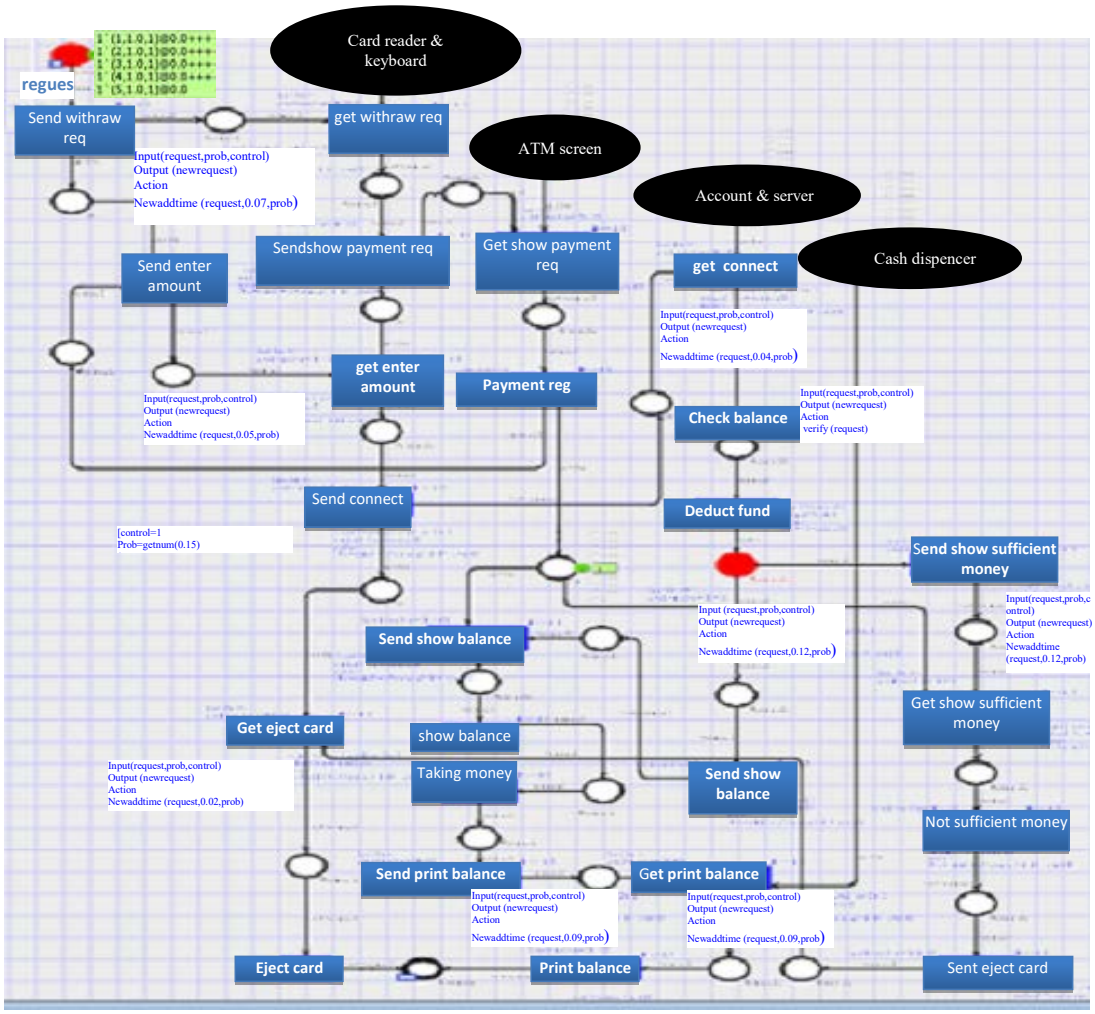

Figure8 – Overview of the presented model

IV. CASE STUDY

In the presumed model, a comprehensive example that is as small as possible is selected so that it would represent a small instance of real, huge examples, while being not highly complicated. To show that the method is executable and accurate in this paper, account withdrawal using an ATM is investigated because it is not very complex. Using the proposed method and simulating the executable model in CPNTools, the availability metric is studied. Figure 8 is an overview of the proposed model which is modeled in light of the described assumptions. The results are investigated in what follows.

To convert UML diagrams to Petri nets, sequence diagram is employed from among UML diagrams. Figure 9 exhibits the sequence diagram of account withdrawal in an object-oriented style. In this diagram, the Annotation of system availability is done. The account withdrawal model diagram should now be converted to an executable model. To do so, the proposed algorithm is adopted. The executable model of the diagram under study is plotted and the resulting model is tested in several stages using labeled clichés. The results may be seen in Table 1.

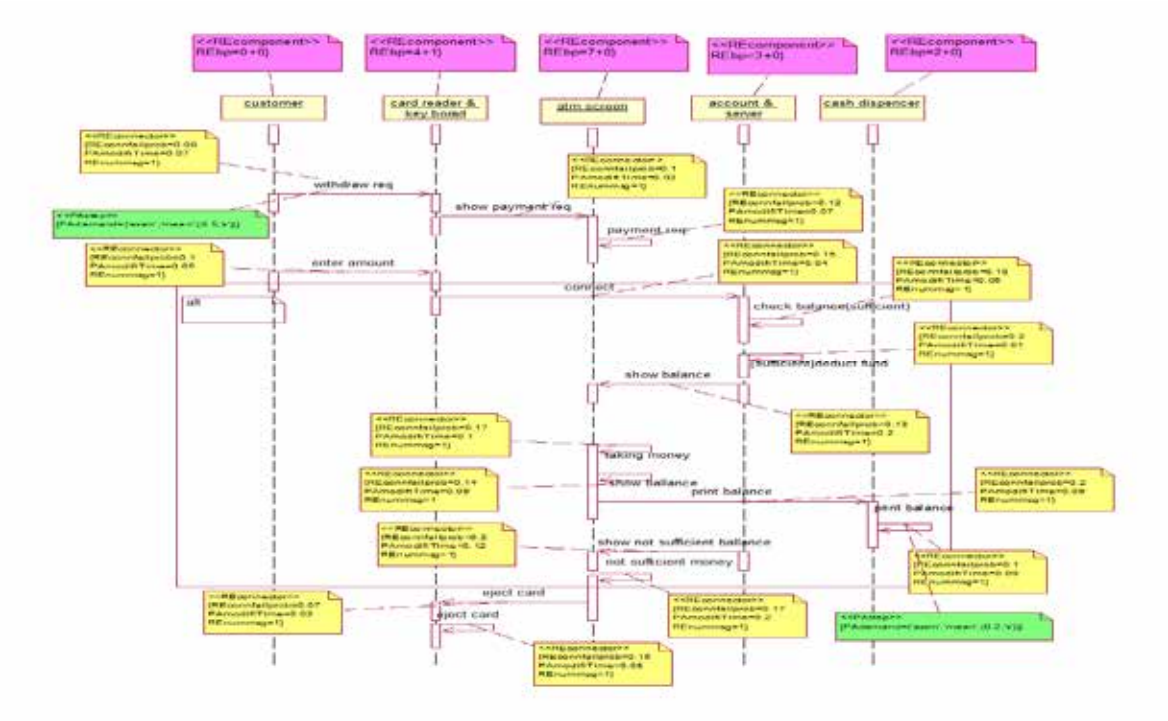

Figure9 – Account withdrawal sequence diagram along with availability labels

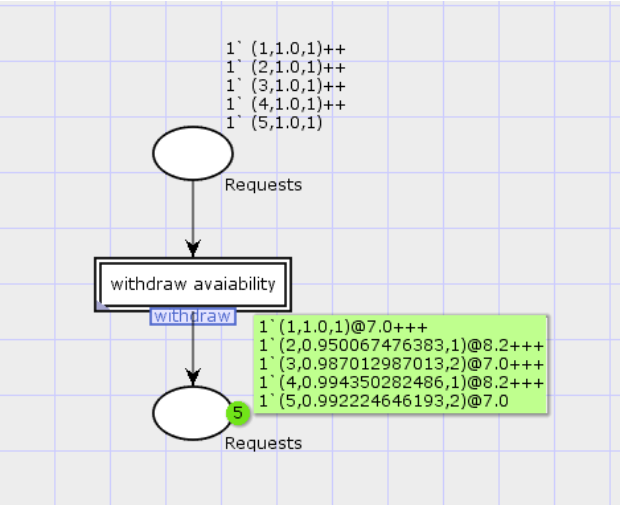

Figure 10 – Execution results with 5 users

Table 1 – Results of executing the proposed method

User No.	availability	Execution Time
1	1	7
2	0.950067476	8.2
3	0.987012987	7
4	0.994350282	8.2
5	0.992224646	7

The results of model execution with five users in an executable area may be seen in Figure 10.

Average availability by executing the model of the proposed method with 5 users may be seen in Table 2.

Table 2 – Total average of the proposed method

Total average		
Number of Users	availability	Execution Time
5	0.984731078	7.48

Average availability=0.984731078
Average unavailability=-0.015268922

Given model simulation in CPNTools software, system requirements and features may be evaluated. Figure 11 shows the results of the colored Petri net of account withdrawal. According to the evaluation results, the studied system has an availability of 0.984731078 which means the system with the probability of 0.015268922 in the time of need could not be used (it means it can't provide service).

V. CONCLUSION

This paper addressed availability evaluation at the software architecture level and the use of object-oriented style without considering hardware context features. Then, the presentation of an executable, CPN-based model was brought under spotlight. These nets enjoy a strong mathematical background. This paper mainly aims to put forward an executable model to evaluate availability at the software architecture level on object-oriented style. Sequence diagram was adopted in this paper to describe architecture. The parameters of the metric under study were

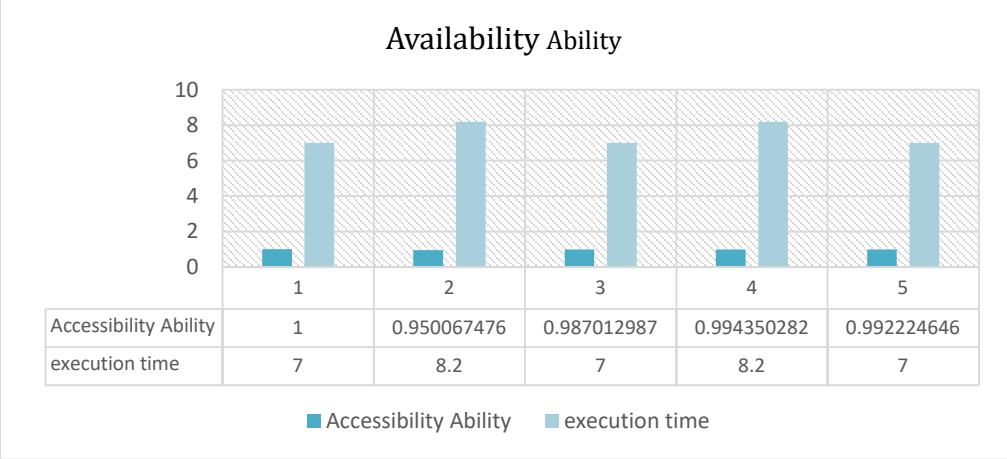

Figure 11 – Availability diagram obtained from execution by 5 users

Annotation to this diagram. In the end, an executable model was obtained with the aid of Petri nets using the proposed method. What this paper actually achieved is the development of an executable model that can investigate availability in object-oriented style. Using the results obtained from simulating this model and analyzing them. First, the problems were identified in the planning phase and the products were modified accordingly, thereby preventing huge economic costs as well as a long time for implementation. Second, the results of availability evaluation show the amount service provided by the system when needed.

REFERENCES

[1] Bass, L, Clements, P. and Kazman, R. (2010) "Software Architecture in Practice: Addison- Wesely Professional". 2nd edition. Vol. 52, pp. 291-432.

[2] Cheung, C. (2009) "A user-oriented software reliability model," Software Engineering, IEEE Transactions on, pp. 118-125.

[3] Roshandel, R., Medvidovic N. and Golubchik, L. (2007) "A Bayesian model for predicting reliability of software systems at the architectural level," Software Architectures, Components, and Applications, pp. 108-126, 2007.

[4] Goshala, S.S., Wong, W E., (2011) "An analytical approach to architecture-based software performance and reliability prediction," Performance Evaluation, vol. 58, pp. 3, 91-432

[5] Sharma, V.S.and Trivedi, K.S. (2009) "Quantifying software performance, reliability and security: An architecture-based approach, " Journal of Systems and Software, vol. 80, pp. 493-509.

[6] YANG, N. And QIAN, Z. (2010) "Quantifying Software Security Based on Stochastic Petri nets, " Journal of Computational Information Systems, vol. 6, pp. 3049-3056.

[7] Fukuzawa, K. and Saeki, M. (2002) "Evaluating software architectures by coloured petri nets, " in Proceedings of the 14th international conference on Software engineering and knowledge engineering, pp. 263-270

[8] Garlan D. and Shaw, M. (2009) "An introduction to software architecture".

[9] http://www.iso.org/, ISO/IEC 25010: (2011)", Systems and software engineering – Systems and software Quality Requirements and Evaluation (SquaRE) – System and software quality models" ," (2009).

[10] Balsamo, S. Marco, A. D. (2004) "Model-Based Performance Prediction in Software Development: A Survey", IEEE Transaction On Software Engineering, Vol 30, NO. 5.

[11] Shin, M., Levis, A., Wagenhals, L. (2007). "Transformation of UML-based System Model to Design/ CPN model for Validating System Behavior In proc Of Compositional Verification of UML Models", Workshop on Compositional Verification of UML'03 conference, USA, pp.126-145.

[12] Allen, R., Douence, A. (2007). "Specifying Dynamism in Software Architectures", Journal of Systems Engineering, Vol. 6, No. 4, pp .52-94

[13] Jensen, K. (2013). "Colored Petri Nets: Basic Concepts, Analysis Methods and Practical Use", EATCS Monographs on Theoretical Computer Science, Vol. 29, No .2, pp70-120.

[14] Cheung, R., Roshandel, N., Medvidovic, (2012). "Early Prediction of Software Component Reliability", ICSE' 08, Leipzig, Germany, pp. 111-120.

[15] "Generating Availibility Data System", North American Electric Reliability Corporation. July (2011). Pp. 7, 17. Retrieved 13 March 2014.

[16] Pourali, A., Malakoti, M., Yektaie, M.H.,(2014) ." Reliability evaluation Of a payment model in mobile e-commerce using colored Petri net" .(JACST), pp. 221-231.

[17] Email, S., And Shams, F., (2010)." Modeling of component diagrams using petri nets", Indian Journal of Science and Technology, Vol. 3 No. 12, pp. 1151-1161.

[18] Spiteri Staines, T., (2013). " Transforming UML Sequence Diagrams into Petri Nets", Journal of Communication and Computer, 10, PP 72-81.

[19] Jensen, k., Kristensen, L.M., (2009) . "Coloured Petri Nets, Modelling and Validation of Concurrent Systems", Springer, July 2009.

[20] Franco, J.M., Barbosa, R., Zenha-Rela, M., (2014)," Availability Evaluation of Software Architectures through Formal Methods", ", IEEE,, Conference: 23-26 Sept. 2014.

[21] Lian-Zhanga, Z, Fan-Sheng K., (2012) "Automatic Conversion from UML to CPN for Software Performance Evaluation", International Workshop on Information and Electronics Engineering, Elsevier Ltd, Procedia Engineering 29 (2012) PP. 2682 – 2686.