Journal of Advanced Computer Science & Technology, 4 (1) (2015) 23-31
www.sciencepubco.com/index.php/JACST
©Science Publishing Corporation
doi: 10.14419/jacst.v4i1.3982
Research Paper

Presentation of an executable model for evaluation of software architecture using blackboard technique and formal models

Fatemeh Majidi [1]*, Ali Harounabadi [2]

[1] Department of Computer Engineering, Ardabil Science and Research Branch, Islamic Azad University, Ardabil, Iran
[1] Department of Computer Engineering, Ardabil Branch, Islamic Azad University, Ardabil, Iran
[2] Department of Computer Engineering, Central Branch, Islamic Azad University, Tehran, Iran
**Corresponding author E-mail: Forough_majidi@yahoo.com*

Abstract

Considering the promotion and complexities of software systems and need to services for repair and updating, success of this system is extremely dependent to its architecture and design. Architectural techniques due to being used by the architectures frequently have specified effect on qualitative characteristics. If this effect is quantitative and measurable for each technique will enable the architect to have more care and convenience for evaluation of design and architectural designs. In this study, at first to show the structure and behavior of software architecture using blackboard technique, component diagram and unified modeling language activity diagram together with required data in relation to nonfunctional needs as stereotypes and labels were used. Later, upon converting the actual model to the formal model, requirements for system performance evaluation on the formal model is provided. Upon analyzing the results, it is observed that the offered method can evaluate the performance of software architecture based on blackboard technique at the design age.

Keywords: *Software Architecture; Blackboard Technique; Performance Evaluation and Time Colored Petri Net.*

1. Introduction

Nowadays, software systems are designed based on abstraction and abstraction is used as a solution for dominating over software complexity. Since increasing the complexity of software systems, design problems have been considered as more important subjects than problems related to algorithms selection and data structure and design and description of general system structure reveals a new type of problem for us, which are referred to as software architecture. Therefore, one of the best solutions for dominating over software complexity is software architecture.
In the software production process, after analysis of requirements and making documentations, probable selections are offered for software system architecture. Architectural design activities have important effect on communication between needs and implementation of system; moreover architecture quality has considerable effect on access to nonfunctional needs of system such as efficiency. If the final selected architecture doesn't meet the requirements, a system of low quality is produced that results in wasting a lot of project resources such as cost and time. To avoid time and cost wasting due to improper selection of architecture, offering a formal description of architecture and its evaluation before beginning the system implementation may be very useful [1].
One of important goals that are followed during analysis of architecture quality is verifying the architecture's access to qualitative features such as performance [2]. In the most software systems, special methods are used for evaluation of qualitative features. Special methods are applicable commonly after architectural implementation means when an executable specimen of system is available. If after applying the special methods, it is revealed that the architecture selected for system may not respond the nonfunctional needs, more time and cost is needed for system architecture changing. In consideration of this subject, we need alternative methods for evaluation of qualitative features which are applicable in initial stages of production process. Establishment of executable models of system architecture is one of

solution that may respond the raised problems. An executable model of architecture is assumed as a formal description of architecture through which may analyze the behavior of final system before architecture implementation and get aware of problems and their in performance and take measure for architecture implementation more confidently and so avoid extra costs and even its failure. In continue, different parts of paper are explained: in second part, a general description of blackboard technique, performance model in unified modeling language (UML) and time color Petri net (TCPN) is presented. In third part, some works related to the subject of this paper are reviewed. In fourth part, the offered model is described. In fifth part, a case study is analyzed for evaluation of offered method and in sixth part, a general conclusion of suggested method is explained.

2. Work field

Fields of this work include unified modeling, time colored Petri nets and blackboard technique which are briefly discussed in consideration of the subject of this study.

2.1. Blackboard technique

In the codified classification of techniques, blackboard is placed in the centralized group. One user is executed on a distinct control set and includes common data which is accessible by these users.
Blackboard is a technique therein independent processing components are referred to as knowledge resource that is operated on the common storage in the name of blackboard. Knowledge resources have no direct interaction with each other, when blackboard is executable, knowledge resources run on the blackboard as an opportunity seeking [3].
This architectural style is always promoting and extending and a structural solution for reaching to the integrity. In plenty of systems particularly systems consisted of prefabricated components, data integrity is provided by blackboard mechanism. Major advantage of this method is that the users are available separate from each other. In addition, common data is an independent part of users. Therefore, this style is scalable and new users may be added easily.

2.2. Establishment of performance model in unified modeling language

Software architecture describes the system in high abstraction levels through specifying the structural and behavioral aspects. But, unified molding language diagrams may not be used to evaluate the software architecture, because some architectural features are not executable using them. In consideration of this subject, a strategy was offered by OMG including performance sub index, similar to other indices for supporting the extension process, stereotypes and labeled values that improves the applicability of these features [4].

2.3. Time colored petri net

Time is one of concepts that are significantly important for evaluation of qualitative trait of efficiency. This concept is introduced using a component called “Global Clock” in colored Petri nets. Values selected by this clock indicate the module time.
When this time is an integer number indicates the discrete time and when a number is actual shows the continuous time. A value may be attributed to each token; furthermore time value may be also given to each time. It is referred to as “timestamp” which denotes the first time therein token can be used and omitted through executing the transition from a location contained it. In a time colored Petri net, when a transition is active that tokens are available in input areas of a transition and guard related to transition is running. But, for firing this transition, it is required to be ready at present. Transition is ready when timestamp of input areas tokens are smaller or equal to current time of model.
In order to simulate an event which lasts for r time units in a colored Petri net, the transition corresponding to the event, to produce timestamps for output tokens that are r time units more than global clock therein transition has been occurred. It means that generated tokens are not available to r time units. In the time colored Petri nets, when in the current time of model, no transition is ready; system time is forwarded to at least one transition gets ready [5].

3. Related works

Model-based methods development for evaluation of systems and computer nets is referred to a long time ago. Correct application of these models may provide appropriate attitudes for evaluation of nonfunctional needs. Due to low knowledge level of software architect, evaluation of these features is not applicable for software architecture, because software architecture for describing the software architecture uses specific marks and signs which are not usable for experts evaluating these features. Therefore, a solution must be found to fill the gap between software designers and nonfunctional features evaluation experts. One of solutions is using the tools and markings of software modeling together with options added thereto that may considerably remove this gap.

Fukazawa et al [6] offered a method there in software architecture is described using unified modeling language component diagram, later converted the above diagram to colored Petri net using an algorithm and later, performance evaluation is provided. For this purpose, the component and its connector are converted to a colored Petri net, but its interface is converted to a colored Petri net place.
Petit and Gamma [7] described the software architecture by collaboration diagram and then converted to Petri net. This method is used for evaluation of performance and reliability. In this method, a collection of predetermined molds in colored Petri nets formed based on objects' behavioral roles are used. These behavioral roles are formed based on available objects structuring in COMMET method, but are not dependent to a specific method and used within different application ranges. Later, results obtained for colored Petri net are reflected in unified modeling language diagrams and the designer may improve the design quality and consequently improve the performance and reliability of system.
Gyarmati et al [8] offered a model therein software performance engineering (SPE) is used for evaluation of performance specifications of software architecture and fabrication and analysis of software executive model resulted from ordinal diagram of unified modeling language. In this method, class diagram and unified modeling language placing is used for describing the software architecture completely, but is not used in the conversion process. Architectural descriptions are converted to the developed queue net to evaluate the performance specifications.
In this paper, three major objectives are under consideration as follows:
-Evaluation of information system performance based on blackboard technique;
-An algorithm for converting blackboard technique to component diagram;
-Converting UML diagrams to the formal models based on features available in blackboard technique for evaluation of its performance.

4. The proposed method

The main method in this paper is presenting an executable model for evaluation of software architecture using blackboard technique. For this purpose, at first software architecture based on blackboard technique is described based on unified modeling language that that is noted by index related to performance feature. In continue, to present an executable model, unified modeling language diagrams are converted to time colored Petri net and then are evaluated.

4.1. Description of software architecture based on blackboard technique using unified modeling language diagrams

In this article, to describe the software architectural structure and behavior, use case, component and activity diagrams are used. In continue these diagrams and notations related to performance are explained.

4.1.1. Effect of use case diagram and its related notations

Use case diagram describes the interaction between system and environment. Case use diagram includes a collection of agents, case uses and relationship between them. This diagram is used for exhibiting the nonfunctional needs and function load applied on the system for description of software architecture. Notations related to this diagram are referred to the agents; causes which indicate a sequence of infinite requests out of system are noted by <<PAopenload>> stereotype which include PAoccurrence label that denotes the time between two consecutive requests and agents which indicate a sequence of finite requests of system are noted with <<PAclosedLoad>> stereotype including two PAexdelay and PApopulation labels. Each one respectively indicates the time range between a completed request and subsequent interaction in system and total system requests [9], [10]. Fig. 1 shows the noted case use diagram.

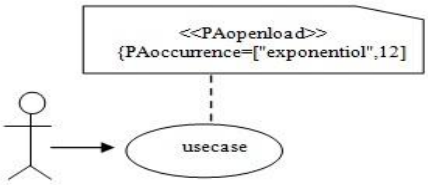
4.1.2. Effect of component diagram and its related notations

Component diagram describes the software system. In this paper, this diagram is used for describing the architectural structure based on blackboard technique. Moreover, a component includes interfaces that each interface defines a collection of component performed operation. Notations related to performance of this diagram are related to the interfaces and components. In this diagram, each component is noted with <<PAhost>> stereotype that specifies the software resource used in this project. This stereotype includes PAschedpolicy label that specifies the system schedule policy. Each interface is noted by <<PAstep>> stereotype that specifies the tasks performance time by the component together with PAdelay and PAdemand labels [11], [10]. Figure (1) shows the noted component diagram. In addition, PArate label indicates the processing rate of processing source related to respective component.

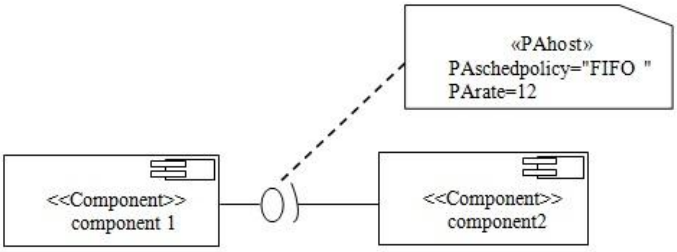
4.1.3. Effect of activity diagram and its related notations

Activity diagram describes the software system behavior. This diagram is a graphic exhibition that shows the control flow from one activity to another. Notations related to performance in this diagram are related to transfers [12]. Each transfer is noted with <<PAstep>> stereotype which demonstrates the service provided in a component and according to its location and includes PAhost and PAdemand labels that each one denotes component location and service request, respectively.

a) Notation of case use diagram of unified modeling language



b) Notation of unified modeling language component diagram



c) Notation of unified modeling language activity diagram

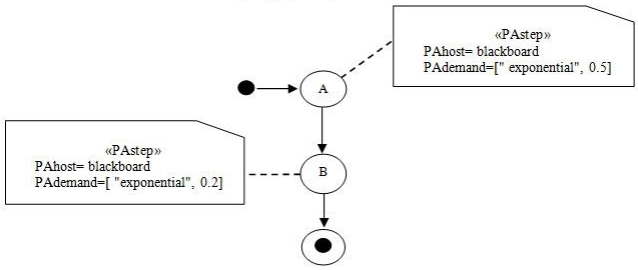


Fig. 1: Notation of Unified Modeling Language Diagrams

4.2. The offered algorithm for converting modeling language diagrams to time colored petri net

The offered algorithm in this paper for converting unified modeling language to time colored Petri net is raised for incorporating an executable model for evaluation of software architecture that includes following stages:
First stage: Description of architectural structure based on blackboard technique
For this purpose, CR cards shown in Fig. 2 are used that indicate static structure of software architecture and relationship between them. Then, through converting these cards to component diagram and determination of performance specifications in this diagram, blackboard-based architectural structure is described.

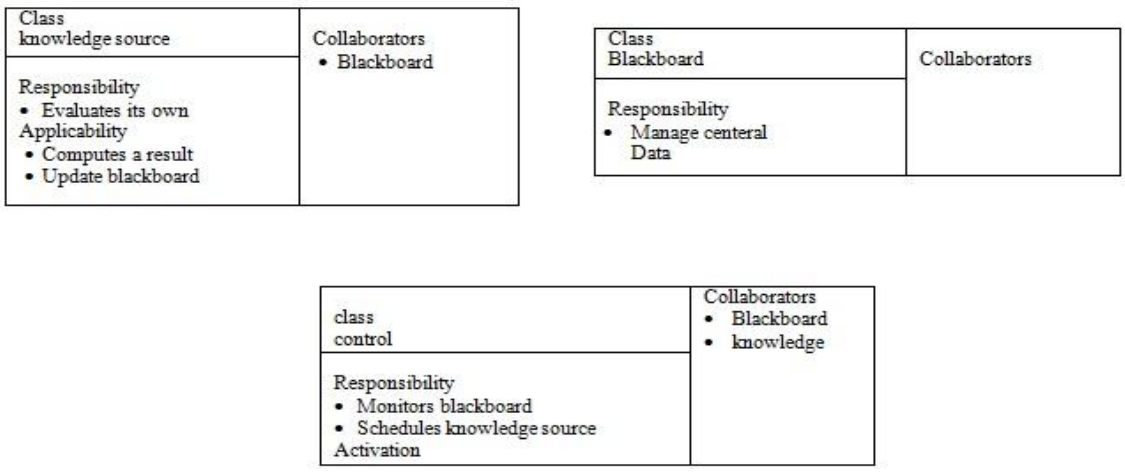


Fig. 2: CR Cards

Second stage: Description of blackboard-based architectural behavior
In this paper, to describe the software architecture behavior, case use and unified modeling language activity diagrams are used. In the suggested course of action, case use diagram is used to exhibit the functional needs and function load applied on the system during software architecture description. Activity diagram describes also the system behavior and shows the control process from one activity to another one.
Third stage: Evaluation of the blackboard-based architecture
Whereas various agents have varied function loads in the system, T-CPN model contains following models which are independent from each other and each one will has their own functional load. Furthermore, requests related to one sub model may have several classes that each one is shown with different colors in T-CPN. Each color is unique and may not be repeated in other classes.
Open Petri net contains input and output to the external environment and shown by <<PAopenload>> stereotype that is used in the case use diagram. Whereas several methods can use a source in the system, we have following definitions in T-CPN model:
If it is assumed that sources are exhibited as RES= {res1, res2 ... res n} for each source, res ϵ RES is defined as a feature called [count [res]]. [Count [res]] denotes total requests that request service from res, index feature is a unique index for identification of sources. Places which use res resource are shown by {ACTION={ action1,action2,..., actionn}, it is obvious that count[res]=a for each source labels total requests in {action ϵ ACTION | resource(action)= res} set by a unique number in range [1,2,...,count[res]]. This unique number is shown by index [action] feature.
If agent x is noted by <<PAopenload>> stereotype, feature values are determined as below:
Count [res] $\forall$ res ϵ RES
Index [action] $\forall$ actionϵ ACTION

$C = MAX_{res} \epsilon \{COUNT [res]\}$ and T. (1)
To show the customers service rate with class r, SR [i,r] in transfer i is used. M is an action in activity diagram.
SR [i,r]= rate[r]/demand [action] where i= index [resource [m]]. (2)
r = index [action]. (3)
λ[r] for is considered for showing the customer input rate with class r that is defined as below:
λ [r] = arrival rate [x] (4)

The input rate is used based on labeled case use that resulted in use of activity diagram.

4.3. Computation of response time

Performance metrics such as response time, queue length etc. may be evaluated using the said evaluation method. To compute the response time, time interval between request and first received response by the other side must be analyzed. In fact:
$T_R = T_S + T_D$ (5)
T_R: Response time
T_S: Service time
T_D: Delay time
Delay time may be defined as delay time in processing queue
To analyze the queue length, tokens number in place must be calculated.

5. Implementation and evaluation of offered model

In this paper, hotel reservation system was assumed as case study, so that this system was implemented on blackboard technique and ultimately is evaluated using the offered method. Blackboard technique is shown in Figure (3). Figure (4) shows the case study with CR cards. Fig. 5, 6 and 7 show case use, component and activity diagrams of unified modeling language of hotel reservation system. Fig. 6 show activity diagram of hotel reservation system. In this scenario, firstly the user declares its request on hotel reservation and the system during some stages responds by its agents in consideration of the user request. For evaluation of nonfunctional needs (such as performance), diagrams shown in fig. 5, 6 and 7 are converted to time colored Petri net model. Final model of time colored Petri net is exhibited in Fig. 8. To evaluate the performance (such as response), 4 requests are input to the system by users and upon their execution on time colored Petri net, valuable results are obtained for evaluation of nonfunctional needs on SA level. Table 1 shows the response time related to users.

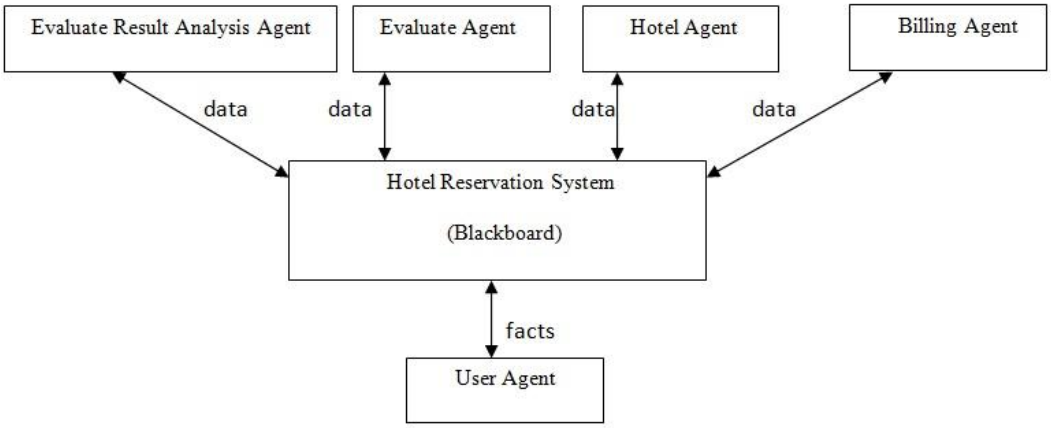


Fig. 3: Abstract Model of Hotel Reservation System

Class: Hotel reservation system	Collaborators
Responsibility • manage centeral data • update	

Class: user agent	Collaborators: Hotel reservation system
Responsibility • evaluate its own applicability • compute a result • update	

Class: evaluate agent	Collaborators: Hotel reservation system
Responsibility • evaluate its own applicability • compute a result • update	

Class: hotel agent	Collaborators: Hotel reservation system
Responsibility • evaluate its own applicability • compute a result • update	

Class: Evaluate result analysis agent	Collaborators: Hotel reservation system
Responsibility • evaluate its own applicability • compute a result • update	

Class: Billing agent	Collaborators: Hotel reservation system
Responsibility • evaluate its own applicability • compute a result • update	

Fig. 4: Exhibition of Hotel Reservation System with CR Card

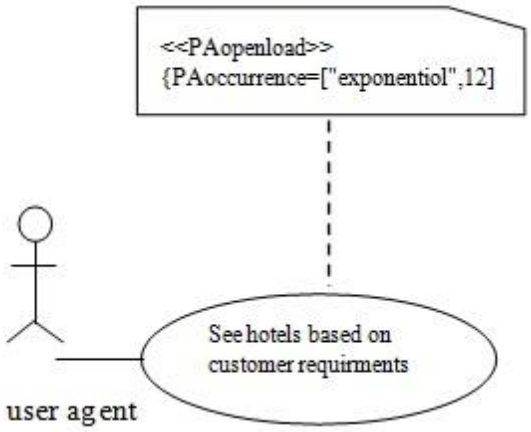


Fig. 5: Case Use Diagram

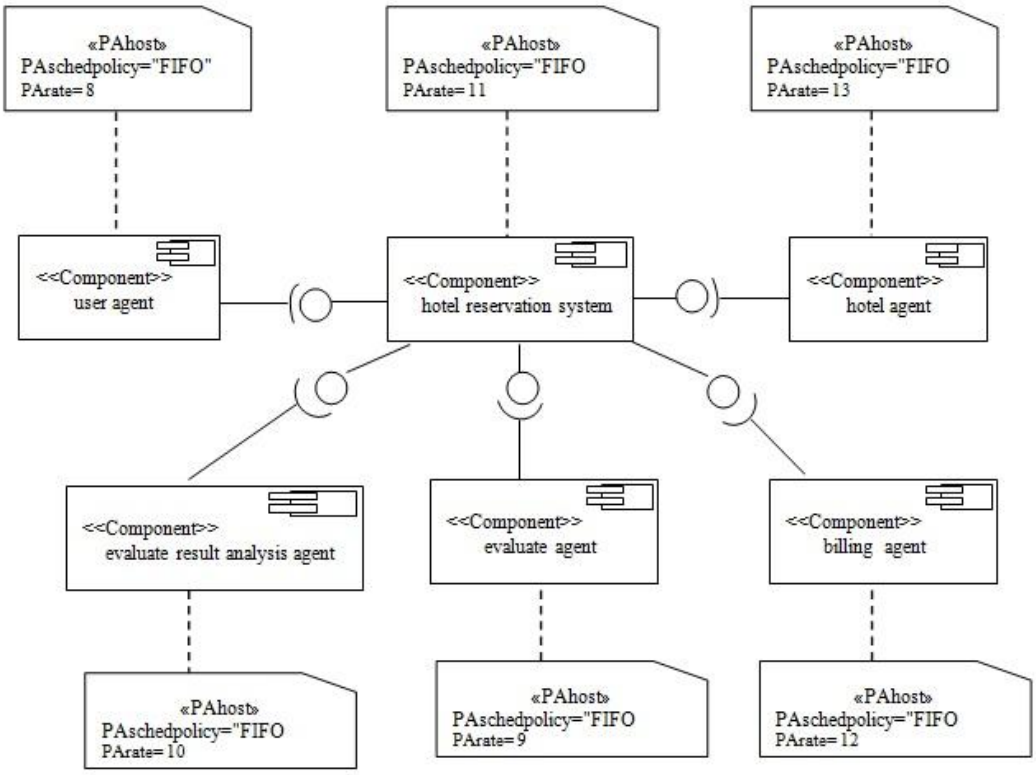


Fig. 6: Notation of Performance in Component Diagram

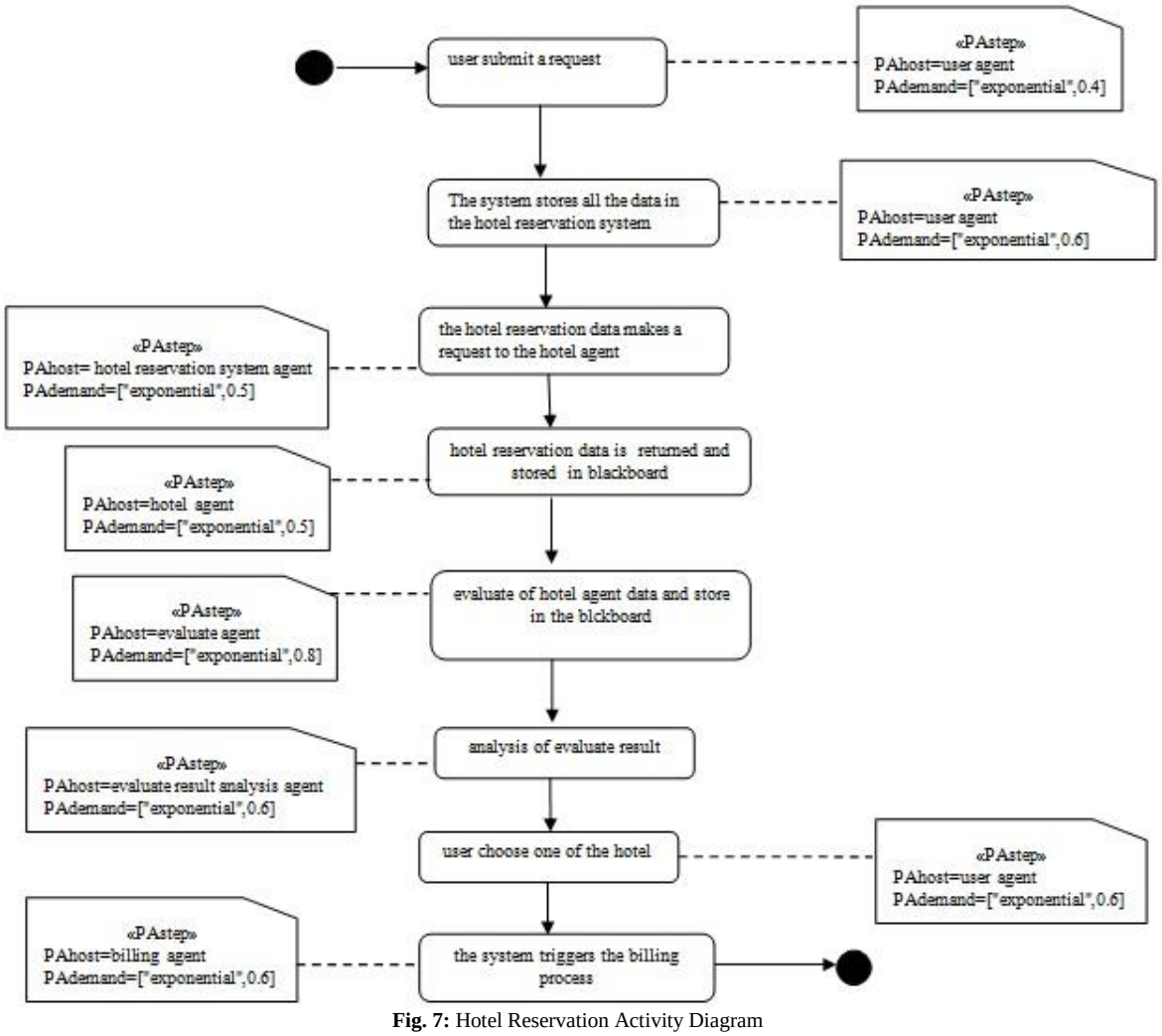


Fig. 7: Hotel Reservation Activity Diagram

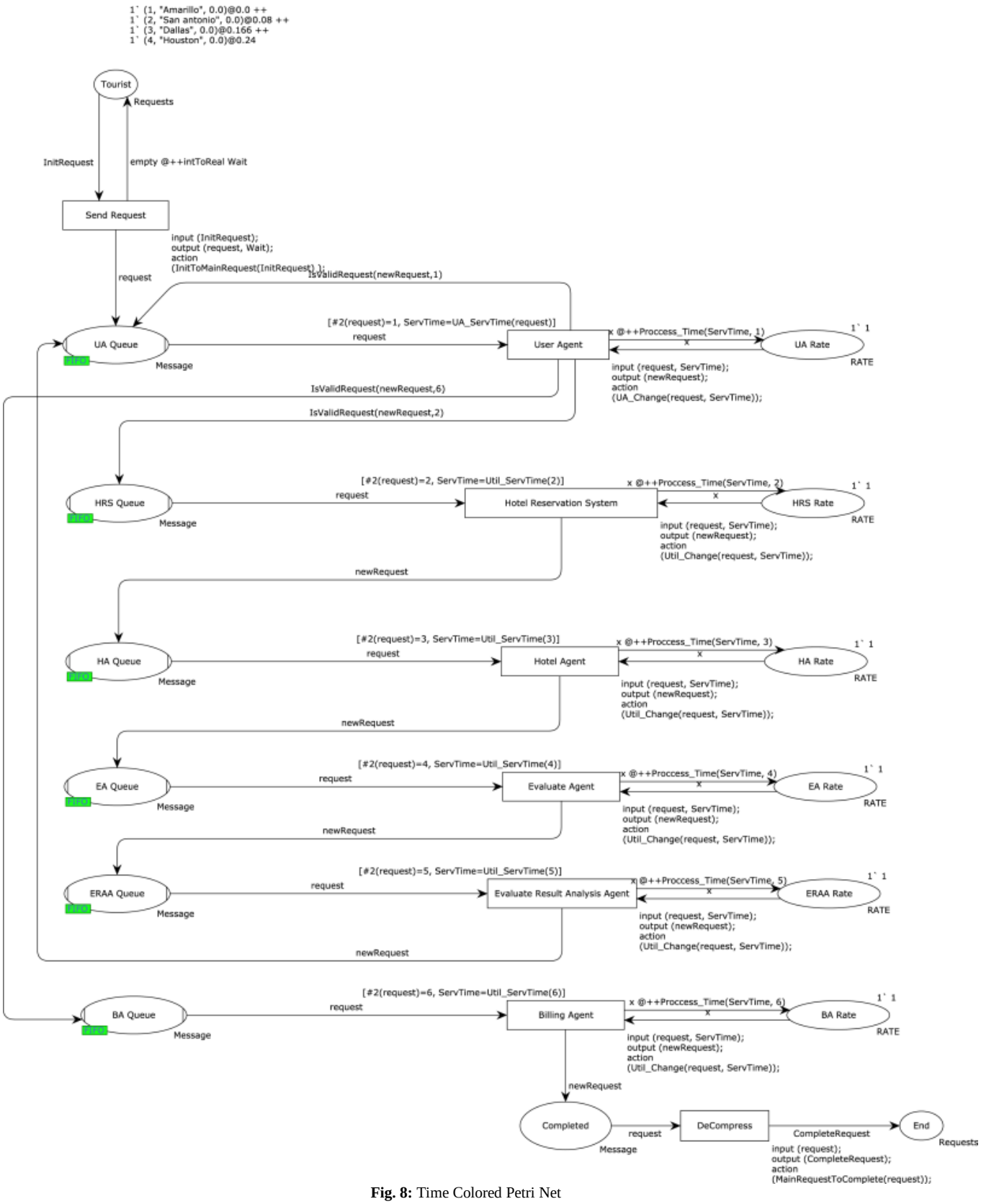


Fig. 8: Time Colored Petri Net

Table 1: Response Time

Number of request	Response time
1	0.16861
2	0.56993
3	0.52483
4	0.295

6. Conclusion

In this paper, an executable model was offered for evaluation of software architecture based on blackboard technique and using formal models. Presenting a formal model based on actual model causes to deal with nonfunctional needs of software for development of systems along with software process models. Summary of simulation provides the requirements for model evaluation before implementation.

References

[1] L. Dobrica, E. Niemela, "A Survay on Software Architecture Analysis Methods", *Software Engineering, IEEE Transactions on*, Vol. 28: 7, (2002), pp. 638-653, Romania.

[2] Technical Report IEEE, *Recommended Practice for Architectural Description of Software Intensive Systems*, IEEE Standards Department, the Architecture Working Group of the Software Engineering Committee, (2000), PP: 1471-2000.

[3] P. Clements, L. Bass, D. Garlan, J. Ivers, R. Little, R. Nord, J. Stafford, *Documenting Software Architectures: Views and Beyond*, Second Edition, Publication City/Country New Jersey, Addison Wesley, (2010).

[4] Object Management Group (OMG), *UML Profile for Reliability, Schedulability, Performance and Time Specification*, (2002).

[5] L. Wells, S. Christensen, L. M. Kristensen, K. H. Mortensen, "Simulation Based Performance Analysis of Web Servers", *Proceedings of the 9th international Workshop on Petri Nets and Performance Models (PNPM'01) IEEE Computer Society*, Washington, USA, (2001), pp:59-68.

[6] K. Fukuzawa, M. Saeki, "Evaluating Software Architecture By Coloured Petri Nets", *in SEKE02 14th International Conference on Software Engineering and Knowledg Engineering*, Italy, (2002), pp:263-270.

[7] R. G. Pettit, H. Gomaa, "Improving the Reliability of Concurrent Object Oriented Software Designs", *proceeding of the 9th IEEE international workshop on object oriented real time dependable systems*, (2004).

[8] E. Gyarmati, P. Strakendal, *Software Performance Prediction using SPE*, Master Thesis Software Engineering, Department of Software Engineering and Computer Science Blekinge Institute of Technology, Ronneby Sweden, (2002).

[9] S. Balsamo, M. Marzolla, R. Mirandola, "Efficient Performance models in Component-Based Software Engineering", *Proce. Of the 32nd Euromicro Conference on Software Enginnering and Advanced Applications*, (2006), pp: 64-71.

[10] S. Merseguer, S. Bernardi, J. Campos, S. Donatelli, "A Compositional Semantics for NML State Machines Aimed at performance Evaluation", *proce. Of the 6th International Workshop on Discrete Event Systems*, (2002), pp: 295-302.

[11] J. Merseguer, J. Campos, E. Mena, "Analysing Internet Software Retrieval System: Modeling and Performance Comparison", *Wireless Networks: the Journal of Mobile Computation and Information*, vol. 9: 3, (2003), pp.223-238.

[12] Object Management Group (OMG) Unified *Modeling Language (UML)*. Version 2.0, (2005).