Decision Science Letters 3 (2014) 411–430

Contents lists available at GrowingScience

Decision Science Letters

homepage: www.GrowingScience.com/dsl

An evaluation of the software architecture efficiency using the Clichés and behavioral diagrams pertaining to the unified modeling language

Siamak Khaksar Haghani[a*], Yousef Abbasnejad[a] and Ali Harounabadi[b]

[a]*Department of Computer Science, International Kish Branch, Islamic Azad University, Kish, Iran*
[b]*Faculty of Computer Science, Central Tehran Branch, Islamic Azad University, Kish, Iran*

C H R O N I C L E

Article history:
Received October 15, 2014
Accepted January 24, 2014
Available online
February 4 2014

Keywords:
Software Architecture
Unified Modeling Language
Efficiency Evaluation
Colored Petri Net

A B S T R A C T

The software architecture plays essential role for the development of the complicated software systems and it is important to evaluate the software architecture efficiency. One way to evaluate the software architecture is to create an executable model from the architecture. Unified Modeling Language (UML) diagrams are used to describe the software architecture. UML has made it easy to use and to evaluate the necessary requirements at the software architecture level. It creates an executable model from these diagrams; yet, since the UML is a standard semi-formal language for describing the software architecture, evaluating the software architecture is not directly possible through it. Furthermore, in order to evaluate the software architecture, one needs to turn the actual model into the formal model. In this study, first we describe the architecture using the UML. Then, some properties of the software architecture are mentioned using the UML sequence diagram, deployment diagram, use case diagram, and component diagram. The necessary information associated with the qualitative characteristic of efficiency will be margined as clichés and labels to these diagrams. The independent and dependent components will be extracted from the component diagram. Finally, the resulted semi-formal model will be mapped into a formal model based on the colored Petri net and finally the evaluation will take place.

1. Introduction

With an increase in the number of complicated software systems usages, it is essential to develop their principles and methodologies to meet all desired properties. It normally costs less to evaluate these properties in early stages before the design and implementation phase. The software architecture, as the first product, plays essential role for the development of the complicated systems. Therefore, we may evaluate the behavior of the system, i.e. qualitative attributes, such as security, reliability, changeability, response time and efficiency. One alternative solution strategy for evaluating the software architecture is to build an executable model from the architecture. An

* Corresponding author. Tel: +989132839152
E-mail addresses: siamakkh64@yahoo.com (S. Khaksar Haghani)

doi: 10.5267/j.dsl.2014.2.002

executable model from the architecture is considered as a formal description of that architecture that allows us to view and to study the behavior before implementing the architecture (Clements & Klein, 2002; Kogut & Clements, 1995; Wang et al., 2006). The Unified Modeling Language (UML) is a standard, semi-formal language intended for the easy description of the software architecture, and it is used to meet the necessary requirements in software engineering. The utilized techniques in UML can cope with some certain tasks. Since UML is not a formal model, the software systems evaluation is not directly possible through it. Hence, the actual model should be transformed into the formal model in order for us to evaluate the efficiency.

There are various formal models to explain an executable architecture such as the colored Petri nets, queue nets, simulation nets, procedural algebra, etc. Among the pre-mentioned models, the colored Petri nets have received more attentions because of their simplicity and high capability. The colored Petri nets can be useful since they have a powerful mathematical support for modeling the behavior in this field. Efficiency and proper response time are among the important subject matters in designing, developing, and implementing the systems. Many designers are looking for some special circumstances to produce software with high efficiency in less production time, cost and necessary maintenance. One way to tackle this problem is to evaluate and to analyze the software efficiency in the preliminary stages of the software production process. In fact, the main problem here is to find out how to evaluate and how to analyze the system architecture using the documentations prior to the production of a software system. Presenting an effective method to evaluate and to analyze the efficiency based on the software architecture may contribute in driving a software project successfully forward (Clements & Klein, 2002; Kumar & Jaspernete, 2012).

2. Research literature

2.1. The software architecture

The software architecture identifies most parts of software as components; but it does not proceed to the internal parts and data structures. Besides the structure, the architecture looks at the behavior of the system too. In fact, it can be said that the software architecture builds a relationship between the structure and the behavior of the components. The primary concern of the software architecture is on the components and connectors as important elements and parts of the architecture. A set of components with their own characteristics interact with each other via the connectors, which have in turn their own descriptive characteristics. They form the architecture of a software system as a specific configuration (Clements & Klein, 2002; Pettit IV & Gomaa, 2003, 2004; Skene & Emmerich, 2003).

2.1.1 The software architecture parts

The software architecture parts include components, connectors, interface and configuration. Components participate as primary blocks and computational entities in system construction. They perform tasks via their internal computations and external communications. A component communicates with the environment via one or more ports. Connector defines the interactions among the components and describes the rules by which they are governed. A connector connects the ports of two or more components. Interface is the interaction between the components and connectors or external environments. Configuration is sometimes mentioned as topology is a connected graph made up of components and connectors, which describes the architecture structure (Clements & Klein, 2002; Musa, 1993).

2.1.2 The Software Architecture Production Phases

The software architecture production phase includes the following activities:

1. Understanding the requirements: There are several ways to necessary extract the requirements. One very basic way to understand the requirements of a system under production is to determine the change domain and the differences of that system compared with the older similar systems.

2. The architecture creation or selection: Generally, in order to create the architecture, a technique is proposed in which the design method is produced, flexibly. This method is supposed to take a firmer shape during the designing process according to the requirements of the desired system.

3. The architecture documentation is the most important phase of the architecture production. Documentation should be tangible for the common users. However, it should have sufficient details so that it could provide the working program and technical plans to carry out a full analysis of the system.

4. The architecture illustration: The expository model should contain useful unambiguous information that is easy to read for the individuals in various fields. There are many techniques for displaying and illustrating the architecture such as the Petri nets for modeling the system. They receive so much attention due to their simplicity and high capabilities.

5. The architecture evaluation: In order to prevent wasting the time and cost, which occurs due to the improper selection of the architecture, describing a formal architecture that is evaluative prior to the system implementation is very essential.

6. The system implementation based on the architecture: In this phase, it is essential that the system developers are committed to the structure and communicative protocols based on the whole architecture.

7. Ensuring the proper implementation of the architecture: At the end, when the architecture is constructed, it could be transferred into the maintenance phase. In this phase, it has to be guaranteed that the architecture is still committed to its displayed form (Clements & Klein, 2002).

2.1.3 The software architecture evaluation

In order to save the necessary time and cost because of an improper architecture selection, it is important to describe a formal evaluative architecture prior to the system implementation. Generally, the advantages of the architecture can be listed as prioritization of the qualitative attributes, improvements to the architecture documentations quality, improvements to the software architecture quality, reduction of risks, paying less cost for implementing the necessary changes in the system.

2.1.4 Creating an executable model from the software architecture

After the architecture products have been made using a codified step-by-step approach, it is necessary to create an executable model to evaluate the software architecture using the created products. We may utilize the model for logical and behavioral evaluation. An executable model from the architecture is considered as a formal description to study the behavior of the final system prior to the architecture implementation. Therefore, we may try to implement the architecture more confidently and to prevent extra cost and even its failure (Emadi, 2008).

2.1.5 The qualitative properties of the software architecture

The qualitative properties are the irresponsibleness requirements of the system that largely determine the style of the architecture. Architecture is the first step toward software production in which the

qualitative requirements can be traced. The qualitative attributes are true in all phases of design, implementation and transfer. The qualitative properties can be classified into two categories in terms of evaluation.

2.1.6 Efficiency

Efficiency is a qualitative attribute of the software, which indicates how well the software works with regards to the time-related issues. Efficiency is evaluated using the response time or the operational power. The response rate and the scalability are the two efficiency criteria with direct relationship with the system efficiency. The efficiency evaluation in the software development process leads to a reduction in necessary expenses, development risk, etc.

2.2. The Unified Modeling Language

UML is a modeling language used to evaluate and to design object-oriented systems. UML is a language for specification, illustration, construction and documentation of the software systems products, commercial system and other non-software systems.

2.3 The Petri nets

Carl Adam Petri is believed to be the first who proposed the Petri nets theory in 1962. Utilization of the Petri nets to evaluate the software architecture and construction of a formal model received so much attention because of the simplicity and capabilities. The Petri nets are displayed graphically and harbor a defined meaning and a well-defined structure. The Petri nets have a long history and include numerous algorithms to analyze these nets. There are many software programs for simulation of these Petri nets, which allow for easy net design, simulation and analysis of the net performance. The Petri nets have a mathematical framework for analysis, validation, veracity confirmation and efficiency evaluation (Reisig, 1985).

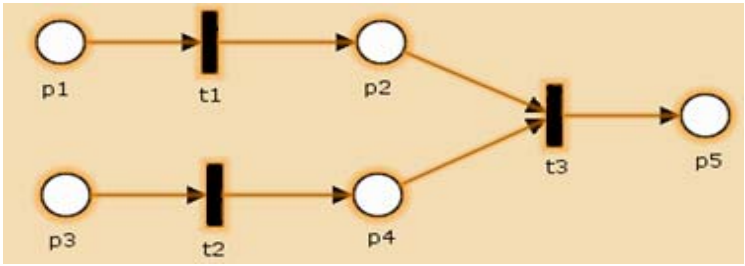

Fig. 1. A simple Petri net (Reisig, 1985)

2.3.1 The colored Petri nets

The colored Petri nets were introduced as a developed model from the Petri nets. In these nets, concepts such as ‘terms’, ‘protection’ and ‘color’ are introduced as well. The colored Petri nets take advantage of the capabilities of simple Petri nets and the programming languages. The data values in these nets are carried by the tokens. The colored Petri nets present precise models from the convoluted asynchronous processing systems CPN= (P, T, C, I-, I+, M_0) which is as follows:

- P is a finite and non-empty set of locations.
- T is a finite and non-empty set of transitions.
- Intersection of two sets P and T is empty (null).
- C is a color function that is a mapping from the P∪T set to the non-empty set.
- I- and I+ are the progressive and recessive confluence functions that are defined on P×T.

● M_0 is a function that is defined on P and describes the primary demarcation in a way that for each p that is a member of P, the $M_0(p) \varepsilon C(p)$ relation is true (Li & Yu., 2001).

■ The colored Petri nets include colored sets, locations, transitions, arcs, variables, protections and the code section.

■ The colored sets of the colored Petri nets: In colored Petri net, each token keeps a color set and only it can take those values defined by the same color set.

■ The locations of the colored Petri nets: In one CPN, a location is represented by an oval. This location includes zero, one or more tokens from a colored set, which is attributed to each location from the CPN. This indicates the token type that location can have.

■ The transitions in the colored Petri nets: In a colored Petri Net, a transition is presented with a rectangle, which have simple behaviors such as moving the tokens from one place to another (*Fukuzawa & Saeki, 2002*).

■ The arcs in the colored Petri nets: Arcs connect the locations to the transitions (input arcs) and transitions to the locations (output arcs). An expression is attributed to each arc and An arc expression keeps a form of a set, which specifies some characteristics: In input arcs, it is the tokens composition that should be presented in the input arc so that the transition could be enabled. In output arcs, it is the tokens composition, which would be created in the output locations when the transition is enabled (*Fukuzawa & Saeki, 2002*).

■ The variables in the colored Petri nets are introduced to provide more flexibility when we model a system, they are defined for a set of colors and used exactly the same way as the colors are used in the arcs.

■ The protections in the colored Petri nets: A protection is a Boolean expression attributed to a transition and adds more conditions for enabling, on variables in the input arc expression. A transition with a protection is enabled when the number of tokens in the input location is similar to the input arc expression and the protection condition is met.

■ The code section in the colored Petri nets: The code section is a piece of code, which accompanies a transition (*Fukuzawa & Saeki, 2002;* Makaruk et al., 2005).

2.4. The efficiency evaluation of the software architecture using the sub-index of properties such as efficiency, time and their clichés

Fig. 2 shows a general efficiency model (Group, 2005), which shows the basic abstractions and the relationships that are used in efficiency analysis. This model maps the domain model class onto the existing clichés in the modeling language. According to this figure, the class attributes will be mapped onto the mapping labels.

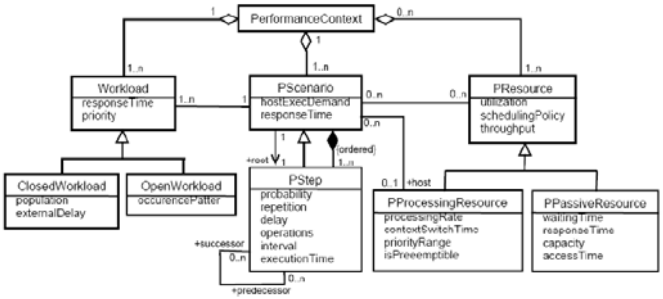

Fig. 2. The general model of efficiency (Group, 2005)

The primary objective of this index is to identify the requirements associated with the timing analysis and the efficiency of UML models. The main clichés under the time and efficiency index are as follows:

<<PAclosedload>> ،<<PAopenload>> ،<<PAresource>> ،<<PAhost>> ،<<PAstep>>

2.4.1 cliché <<PAcontext>>

This cliché models a field of efficiency analysis. The basic classes for which this cliché is used are shown in Table 1. This cliché lacks any labels.

Table 1
Cliché <<PAcontext>> and its labels

Stereotype	Base class
<<PAcontext>>	Collaboration Collaboration Instance Set Activity Graph

2.4.2 The clichés <<PAopenload>>, <<PAcloseload>>

In the efficiency models, each scenario is performed by the users with a specific intensity called the 'workload'. In other words, the workload specifies the requesting intensity for executing a specific scenario and the required response time or estimation for that workload, which incorporates the close or open workload. A closed workload has a fixed number of tasks, which move along the scenario executions. If the number of requests to the system is unlimited, the system has an open workload and shows a population in which the input rate to the system is depicted as predefined patterns with population being a variable. Table 2 and Table 3 show the <<PAclosedload>> and <<PAopenload>> respectively along with their labels.

Table 2
The cliché <<PAcloseload>> with its labels

Stereotype	Base Class	Tags
<<PAclosedload>>	Method Stimulus Message Action State Subactivity State Action Operation Method Reception	PArespTime PApriority PApopulation PAextDelay

Table 3
The cliché <<PAopenload>> with its labels

Stereotype	Base Class	Tags
<<PAopenload>>	Message Stimulus Action State Subactivity State Action Action Execution Operation Method Reception	PArespTime PApriority PAoccurrence

2.4.3 The cliché <<PAhost>>

This cliché models a processing source. In the first phase, this cliché is used in the field of efficiency. Table 4 includes the basic classes and the labels pertaining to the cliché.

Table 4
The cliché <<PAhost>> with its labels

Stereotype	Base Class	Tags
<<PAhost>>	Classifier Node ClassifierRole Instance Partition	PAutilization PAschdpolicy PArate PActxtswT PAprioRange PApreemptable PAthroughput

2.4.4 The cliché <<PAstep>>

This cliché models an inactive source in an efficiency analysis scenario. The scenarios will be combined with the previous and next relationships with the help of levels. A level is a basic operation that is defined by a sub-scenario. Meanwhile, each level has a limited time for execution, the probability of execution, repetition count and an arbitrary time interval between each two repetitions. The basic classes and their defined labels are shown in Table 5.

Table 5
The cliché <<PAstep>> with its labels

Stereotype	Base Class	Tags
<<PAstep>>	Message Stimulus Action State Subactivity State	PAdemand PArespTime PAprob PArep PAdelay PAextOp Painter Val

3. Research background

According to Motameni et al. (2008), it is possible to turn the fuzzy activity diagram into the fuzzy Petri net using an algorithm. This diagram is applied to illustrate the actions existing in the fuzzy nondeterministic systems which is indicative of the dynamic aspect of the software architecture. Also the resulted Petri net applies it to analyze and evaluate the runtime bevahior of the software architecure.

Zhang (2006) indicated that the software architecture can be described with an architecture description language called XSADL4PE. This language adds the responsibleness and irresponsibleness requirements to the software architecture characteristics and then evaluates the software architecture efficiency by adding the random process algebra, the architecture description language and using the 3COCASETOOL tool.

According to Balsamo et al. (1998), it is possible to describe the software architecture using the CHAM model and then we can analyze it using the queue net. Marco (2003) suggested that in this method, the fotware architecture is described using the message sequence diagram and then it turns into the efficiency model automatically based on the queue net. After that, designers will evaluate the

system efficiency by evaluating the queue net. Emadi (2008) stated that in recent years, evaluating the irresponsibleness requirements and the qualitative attributes in the preliminary levels of the software development process especially in the architecture phase from the engineering and designing phases have received so much attentions. In research groups, the evaluation of the efficiency requirements in the preliminary levels of the software development and in the software architecture level has been studied and various methods have been proposed. Hadipour Sanati (2005) suggested that using the index 1+4 concept and the CPN/Design tool, it is possible to change these indices into the colored Petri nets. Focusing on the qualitative attributes of the time efficiency, resource efficiency, memory and file security and also the response time, we can proceed to evaluating the software architecture.

Shirazi (2008) indicated that we could describe the software architecture using the UML diagrams and then using the CPN Tools and the conducted simulations, we can predict the efficiency metrics. Sharafi (2005) declared that we could describe the software architecture uing the UML 2.o software and evaluate the software architecture efficiency by developing the general model of the team automata and introducing an efficiency model on the developed team automata. Motameni et al. (2009) suggested that we could change the fuzzy state diagram, one of the diagrams existing in the F-UML model, into the fuzzy colored Petri net using an algorithm. Using the fuzzy state diagram, we can display the static aspect of the nondeterministic systems software architecture. Changing it into the fuzzy Petri net create an executable model from the software architecture with which we can structurally analyze the software arhictecture.

4. The proposed method for evaluating the software architecture efficiency

The colored Petri net model is composed of *T* transition, which shows the resources are working and *C* different colors which are indicative of the customers classes. Every resource can offer its services with numerous methods. So, the service rate is different for sifferent services. We show this difference in the service rate with different colors that we attribute to the tokens. If the workload of the system is closed, the agent is defined with the cliché <<PAclosedload>> which in this case an extra transition will be defined in CPN. This extra transition indicates the elapsed time between an interaction and the start of its next repetition.

Since numerous agents have different workloads in the system, the CPN model includes various sub-models that are independet from one another each of which will have their own workloads. Besides, the requests related to a sub-model can have numerous classes each of which is shown with a different color in the CPN. Each color is unique and can not be repeated in other classes. As it was mentioned earlier, the open Petri net is a net that has an input and an output to the outside environment which corresponds to the use case diagram with the cliché <<PAopenload>>. But, the closed Petri net lacks an input and an output to the outside environment which its corresponsing use case has the cliché <<PAclosedload>>.

In the CPN model, the following definitions were employed:
- The system resources set: $RESOURCES= \{R_1,R_2,...,R_t\}$
$Identity\ [R_i] =i\ ,for\ each\ R_i\ \varepsilon\ RESORCES$

- For each resource, the attribute COUNT [R] is defined that indicates the number of requests made from R.
- The set of methods that use the resource R: METHODS = {mthd1, mthd2, …, mthdm}
- Therefore, we will have: $COUNT\ [R] = m$
- For each resource $R\ \varepsilon\ RESOURCES$ and for each $Mthd\ \varepsilon\ METHODS$ in the set (1-4), we define the attribute *INDEX [mthd],* which is a unique number in the interval *[1, 2, ..., COUNT[R]]*. This unique number in each level indicates the number of methods reference to a specific resource from the beginning.

$$\{mthd \varepsilon METHODS \mid resource\ (mthd) = R\} \quad (1)$$

We attribute a natural number to each of the system methods in the sequence diagram in order of execution from the top to the bottom. As we specify the first executed method with 1, the second one with 2 and etcetera. We assume that the total number of executed methods in the system equals 'n': $TOTALmthd = \{mthd_1, mthd_2, \ldots, mthd_n\}$.

With regards to the components in the component diagram and the interactions existing between the collaborative components, we can define the interactions set in terms of synchronization as the following ordered pairs:

$$DEPENDENCIES = \begin{cases} & z=0 \text{ if mthdj is independent} \\ (\text{mthdj,z}) & z=1 \text{ if mthdj is dependent, } (1 \le j \le n) \end{cases}$$

If an operation is carried out independently by a component and without expecting any response from the collaborative component, the value of z equals 0. Also, if an operation is dependently performed and it depends on receiving some response from the collaborative component, the value of z will equal ato 1. Input in the open Petri net will enter a transition that the first transition in the sequence diagram used that resource.

4.1 The algorithm for converting the UML semi-formal model to the CPN formal model

For the agent 'x' with the cliché <<PAopenload>>, the following attribute values should be specified:

$$COUNT\ [R] \quad \forall\ R \varepsilon RECOURCES \quad (2)$$
$$INDEX\ [mthd] \quad \forall\ mthd \varepsilon METHODS \quad (3)$$
$$C = MAX_{R \varepsilon RESOURCES}\ \{COUNT\ [R]\}\ and\ T \quad (4)$$

With regards to the sequence diagram, the related number for each method out of the total number of the executed methods should be specified$(1 \le j \le n)$.

With regards to the component diagram, the independence or dependece component of the method execution (0 or 1) should be specified:

$$(mthd_j, z) \quad \forall\ mthd \varepsilon TOTALmthd \quad (5)$$

In order to show the customers service rate with class 'c' in the transition 'i', we will use *ServiceRate[i,c]*. *'m'* is a method in the sequence diagram.

$$ServiceRate\ [i, c] = rate\ [R] / demand\ [mthd] \quad (6)$$

where $i = Identity\ [resource[m]]$

$$c = INDEX\ [mthd] \quad (7)$$

We consider the vector $\lambda[c]$ to show the customers input rate with class 'c' which is defined as the Eq. (8):

$$\lambda[c] = ArrivalRate\ [x] \quad (8)$$

The input rate will be calculated with regards to the use case with the label PAoccurence which led to the utilization of the sequence diagram.

If the agent ‘x’ with the cliché <<PAclosedload>> is margined, we should add the following changes to the algorithm:

- We consider the number of transitions one unit more than the resources existing in the deployment diagram.

$$T = |R| + 1 \tag{9}$$

The added transition will be specified with 0 and is used to consider the delay. The fire rate of this transition for all customer classes is as what follows (‘s’ is a class from customers):

$$ServiceRate[0,s] = 1 / extdelay[x] \tag{10}$$

Regarding the label PApopulation of the desired agent, the value N which equals the total number of requests existing in the system, will be calculated. The above label in the Petri net is shown with the attribute ‘population’ as the following:

$$N = population\ [x] \tag{11}$$

Combining the two states of open and closed workloads in the above algorithm, a complete algorithm will be created. The number of CPN sub-models in it equals the total number of agents in the two labels, namely <<PAopenload>> and <<PAclosedload>>.

5. A case study

In this section, we used the online shopping system as the case study. In online shopping, the main event is as follows: first, the customer finds their required item or service on a desired webpage and adds them to their shopping basket. Then, they select their desired bank via the interbank network cards. The desired form to the bank terminal appears. On this page, the customer should insert the card number, secondary password, CVV2 code and the security code shown in the bank terminal form. Bank verifies the inserted information. If the inserted security code is not valid, the S1 lateral event occurs. If the card related information is not correct, the S2 lateral event occurs. In case of the validity of all information, bank verifies the account balance connected to the customer card. If the withdrawal account balance is less than the customer’s requested amount, the S3 lateral event occurs. If the acconut balance is enough, the customer’s requested amount will be deducted from it and a message will be sent meaning the transaction has been successfully done and the new balance will be issued for the customer.

The S1 lateral event: A message will be displayed to the customer based on the invalidity of the security code and a new security code will be displayed on the screen.
The S2 lateral event: A message related to the incorrectness of the information related to the card will be issued for the customer. Usually, in this event, due to security considerations and hackers success rate reduction, it is not exactly mentioned which inserted information related to the card is incorrect.
The S3 lateral event: Inadequateness of the withdrawal balance of customers account is announced and the transaction will be aborted.

In this case study, the customer is in connection with the system as an agent. The web page plays the role of an interface between the system and the customer. Bank is of a control class that controls the card-related information and makes decisions on perfoming or aborting the transaction. The customer

account is of entity class that contains information such as the information related to the bank card, account number and account balance.

In order to describe the static structure of the system, we employed the component diagram extracted from the CRC cards. In order to describe the physical resources of the system, we utilized the deployment diagram and also we used the use case and sequence diagrams in order to describe the system's dynamic structure and the existing interactions. Moreover, the dependent and independent components were extracted from the component diagram.

5.1 The case study diagram and its cliches

The case study diagram is considered as a ground for efficiency and is margined with the cliché <<PAcontext>>. We Assume that our net is an open Petri net and has input and output to the outside environment. Therefore, we use the cliché <<PAopenLoad>> for which we use the label PAoccurence in the use case diagram which indicates the time interval between two successive requests of the user. In order to show the time, we use an attribute that indicates the type of input (e.g. exponential or infinite). Fig. 3 shows the margined use case diagram pertaining to the online shopping.

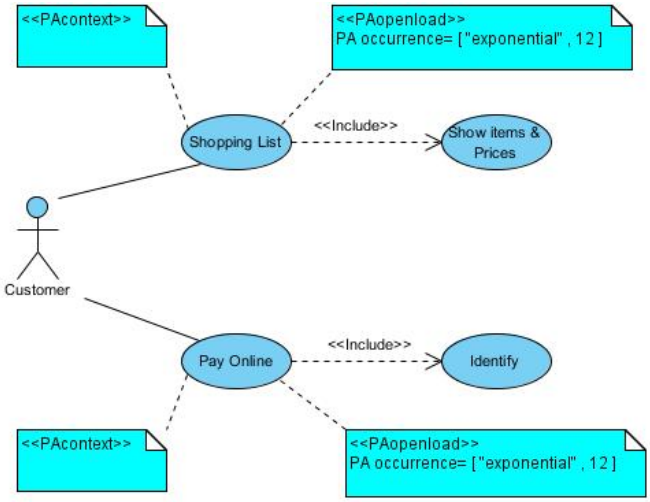

Fig. 3. The margined use case diagram pertaining to the online shopping

5.2 The sequence diagram and its cliches

In order to add the efficiency information in the sequence diagram, we use the cliché <<PAstep>>. Among this cliché labels, we use the labels PAhost and PAdemand. PAhost refers to the name of the requesting resource and PAdemand refers to the requesting rate of the service from the resource. Fig. 4 shows the sequence diagram in which the efficiency information is added. In this diagram, all of the system interactions are shown.

5.3 The deployment diagram and its cliches

For this diagram, we use the cliché <<PAhost>> which indicates the resource name (server) used in the system. This cliché has numerous labels too from which the PArate and PAschedPolicy labels are used here. PArate and PAschedPolicy are the process rate of the resource processor and the timing policy that the processor has for allocating the resource, respectively. First, we define the value of each of these labels for these resources so that the deployment diagram could be margined with the information related to the efficiency. Fig. 5 shows the deployment diagram related to the online shopping system.

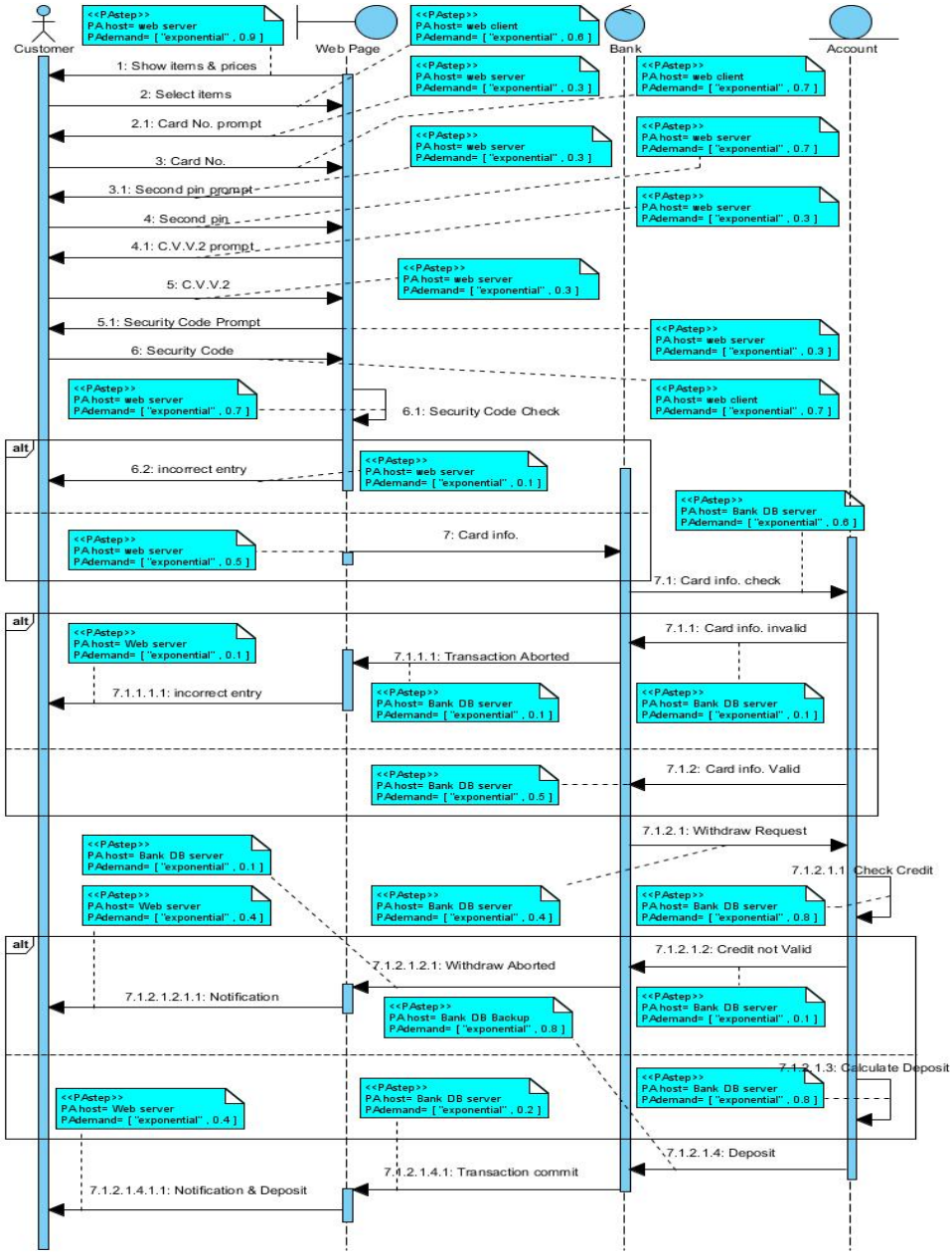

Fig. 4. The sequence diagram together with the efficiency information related to the online shopping system

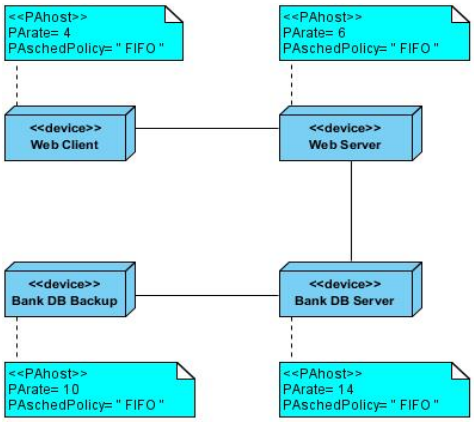

Fig. 5. The margined deployment diagram pertaining to the online shopping system

5-4 The CRC cards and the component diagram for describing the static structure of the system

The CRC cards related to the online shopping system are shown in Fig. 6. Each card includes the component name, its responsibilities and collaborators. Next, we will consider a component in the diagram for each card and map the relationships between the components. In this case study, we have four components that are: 'customer','webpage',bank','account'. Each component has responsibilities and they get help from the collaborating components for some of these responsibilities.

Then it specifies the information added to the component diagram, the service center type, the rate of the services provided by the component and their timing policy for extracting the tasks from the waiting line (queue). Therefore, the cliché <<PAhost<< is used. In order to specify the timing policy, we use the label PAschedpolicy and we have to use the cliché <<PAstep>> along with the label PAdemand in order to specify the required time for executing a component. Fig. 7 shows the component diagram with the margined efficiency information.

Component name: Customer	
Responsibilities	**Collaborators**
Add items to list	Web page
Remove items from list	Web page
Enter card info	Web page
Enter security code	Web page

Component name: Web page	
Responsibilities	**Collaborators**
Show items list	----------
Receive card info	Customer
Receive security code	Customer
Check security code	----------
Send card info	Bank
Receive transaction status	Bank
Show notifications	----------

Component name: Bank	
Responsibilities	**Collaborators**
Receive card info	Web page
Check card info	Account
Receive card info validity status	Account
Withdraw request	----------
Receive credit validity	Account
Receive deposit	Account
Transaction status	----------

Component name: Account	
Responsibilities	**Collaborators**
Receive card info check request	Bank
Check card info validity	---------
Receive withdraw request	Bank
Check credit validity	---------
Notify credit status	---------
Calculate deposit	---------

Fig. 6. The CRC cards of the online shopping system

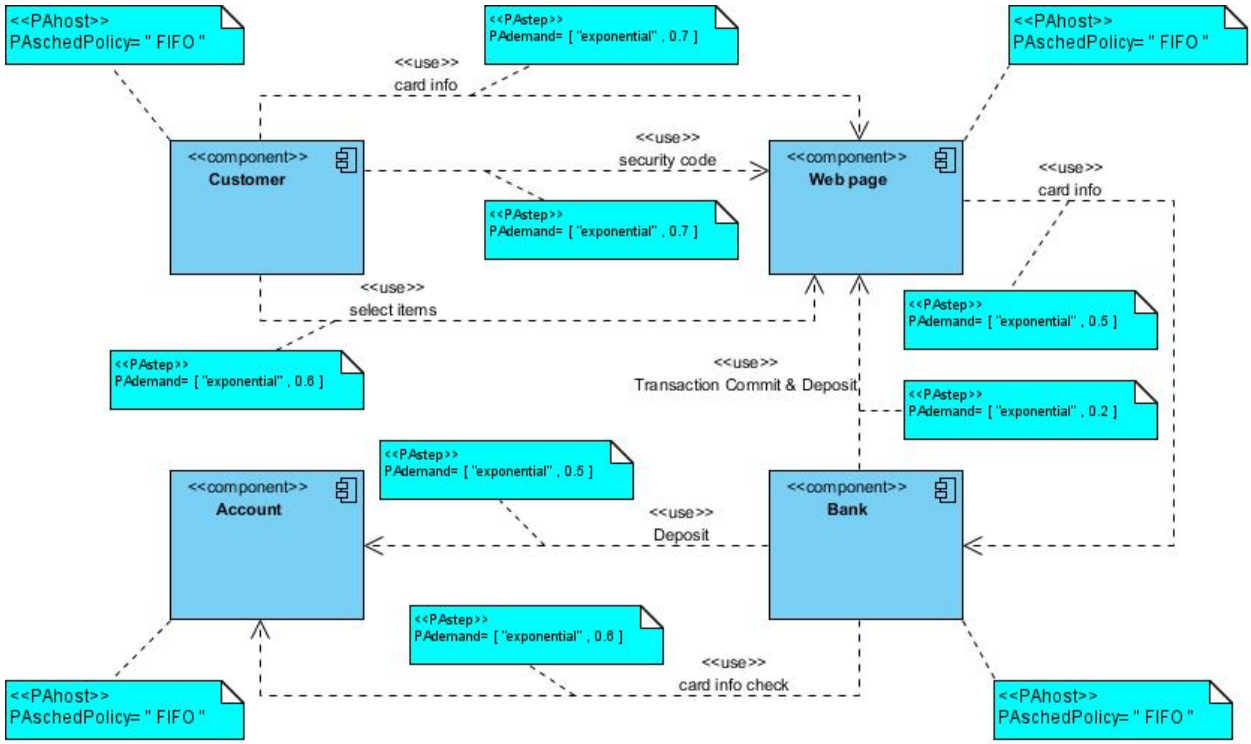

Fig. 7. The component diagram together with the efficiency information pertaining to the online shopping system

5.5 Extracting the colored Petri net from the online shopping system's UML diagrams

Now, we should map our desgined diagrams onto the colored Petri net. To do this, first we should design a directed graph called the middle graph. Each node in this graph includes 3 items. R or the resources which will correspond to the locations in the colored Petri net. Indices are indicators of colors and the ordered pair $(mthd_j, z)$ indicates whether the j-th method is dependent or indpendent, depending on the extracted z value according to the suggested algorithm. Fig. 8 shows the middle graph's structure. The colored Petri net is designed from the middle graph. You can see the the middle graph's structure in figure (8) which was designed according to the sequence diagram and the component diagram. The number related to R varies from 1 to 4 which will be assigned to the system resources. The index 'INDEX' is the counter of the total number of reference to a specific reource in each level. Therefore:

$$R = \begin{cases} \text{Web Client} & 1 \\ \text{Web Server} & 2 \\ \text{Bank DB Server} & 3 \\ \text{Bank DB Backup} & 4 \end{cases}$$

On the other hand, the parameter 'z' in the ordered pair (*mthdj, z*) in each level indicates the dependence and independece of the j-th mehtod. For example, if we have (*mthd5 0*), this means that executing the method number 5 is independent and if we have (*mthd5,1*), this means that executing the method number 5 is dependent to recieving another method from the collaborating component. If a method is dependent, it should be delayed until we receive the response method from the collaborating component. The delay time can be extracted from the label PAdemand of the cliché <<PAstep>> which is margined to the response method. With regards to the sequence diagram, totally 23 methods are executable from the beginning to the end of the system activity. Looking at the component diagram of Figs. (5-9), we can understand that 7 methods out of the total 23 are dependent and the rest of the methods are independent. The dependent methods in some of the nodes existing in

the middle graph which have the value z=1 in their ordered pair (*mthd5, z*) are distinguished from other methods that are dependent (z=0). The extracted delay time should be applied as a delay in tokens movement with various colors (stamp time) in the timed colored Petri net. Also, the protective expression of the dependent method's transition should take the boolean value 1 from the required response method. When multiple response methods should be produced by a component, in a way that the methods' delay times are different, the delay time related to the firing of the component related transition should be equal to the minimum delay time of the methods produced by the component. For example, if the method 1's delay equals 2 time units, the method 2's delay equals 4 time units and the method 3's delay equals 5 time units, the transition firing delay related to the component will at least be equal to 2 time units.

In addition, the token that should be fired with more delay will have the *PAdemand – PAdemand MIN* with them as their stamp time.

$$delay(TransitionR(RESPONSE)) = MIN(PAdemand(mthd1,mthd2,\ldots,mthdm)),\ \forall R \in RECOURCES$$
$$guard(TransitionR(DEPENDENT)) = 1$$

The components that are completely independent and do not need a response from the collaborating component in no part in system's life will not need a protective boolean expression as well. Their corresponsing transition is always executable. The advantage of separating the independent components from the dependent ones is that with the decrease in response method's delay time, the whole system's response time decreases as well. This is one of the effective factors in increasing the efficiency. This is possible when the component corresponds to the resource and via an increase in the resource processig rate, especially through augmenting the processing rate of resources that are responsible for producing the highest number of response methods for other components. As it can be seen from the component diagram, the component 'Client Web' produces the highest number of response methods for other components. In other words, it has the highest number of methods dependece. Ergo, the reponse time of the whole system decreases with an increase in this component's processing time.

5.5.1 Calculating the request rate for the online shopping system

The input in the open Petri net enters a transition that is used by the first message in the sequence diagram. The input rate to the Petri net (λ) equals the value of the label 'PAoccurence' in an agent that uses this sequence diagram to perform its own user case. According to the use case diagram, the value of the label 'PAoccurence' for the main event 'Pay online' equals 12. With regards to this value, the value 'λ' will be 1/12.

$$PAoccurrence = [\text{“exponential”}, 12] \rightarrow \lambda = 1/12$$

In the next step, we will define the number of colors existing in the net. This is simply possible through checking the middle graph. With regards to the middle graph depicted in figure (8), the number of colors equal the number of references to a resource that had the highest number of references, i.e. 10 colors. Therefore, the color set for the colored Petri net is defined as $C = \{c_1, c_2, c_3, c_4, c_5, c_6, c_7, c_8, c_9, c_{10}\}$.

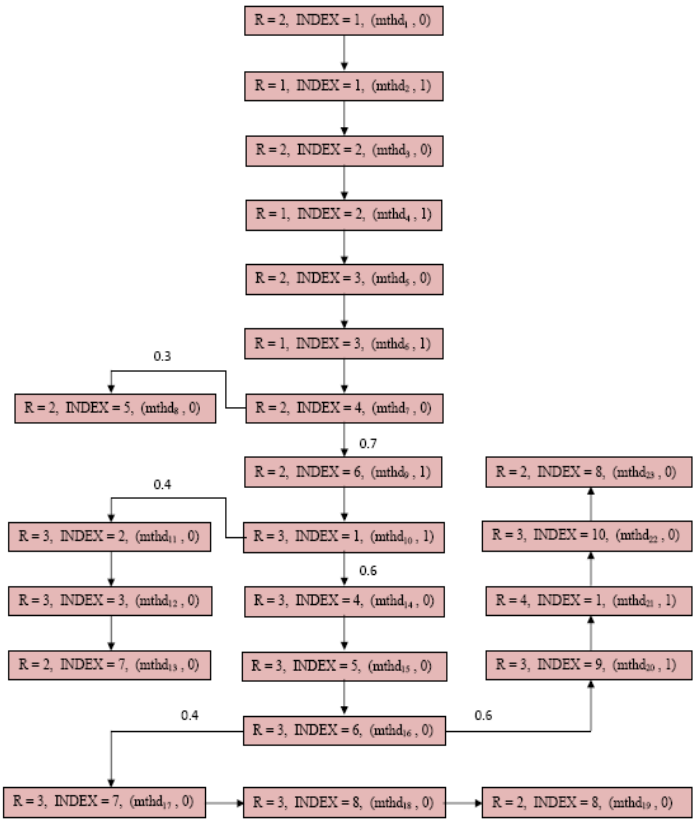

Fig. 8. The middle graph structure (the interface between the semi-formal model and the formal model)

In this level, we extract the label PAdemand value in each reference to the resource from the sequence diagram from the top to the bottom and we will write them per each resource. Table (6) shows these values pertaining to each resource.

Table 6
The label PAdemand values for resources

PAdemand	c_1	c_2	c_3	c_4	c_5	c_6	c_7	c_8	c_9	c_{10}
Web Client	0.6	0.7	0.7	----	----	----	----	----	----	----
Web Server	0.9	0.3	0.3	0.7	0.1	0.5	0.1	0.4	0.4	----
Bank DB Server	0.8	0.1	0.1	0.5	0.4	0.8	0.1	0.1	0.5	0.2
Bank DB Backup	0.8	----	----	----	----	----	----	----	----	----

5.5.2 Calculating the service rate for the online shopping system

Having the PAdemand values which are depicted in Table 6 and PArate which is specified in the deployment diagram, we can calculate the service rate (fire rate) using the Eq. (11):

ServiceRate =rate [R]/ demand [method] (11)
ServiceRate$_1$=(4/0.6,4/0.7,4/0.7,1,1,1,1,1,1,1)=(6.6,5.7,5.7,1,1,1,1,1,1,1)
ServiceRate$_2$=(6/0.9,6/0.3,6/0.3,6/0.7,6/0.1,6/0.5,6/0.1,6/0.4,6/0.4,1) =(6.6,20,20,8.5,60,12,60,15,15,1)
ServiceRate$_3$=(14/0.8,14/0.1,14/0.1,14/0.5,14/0.4,14/0.8,14/0.1,14/0.1,14/0.5,14/0.2)=(17.5,140,140,28,35,17.5, 140,140,28,70)
SreviceRate$_4$ =(10/0.8,1,1,1,1,1,1,1,1,1) =(12.5,1,1,1,1,1,1,1,1,1)

We calculated the service rate values per each resource. These values are summarized in Table 7.

Table 7
The service rate values per each resource

Service Rate	c_1	c_2	c_3	c_4	c_5	c_6	c_7	c_8	c_9	c_{10}
Web Client	6.6	5.7	5.7	1	1	1	1	1	1	1
Web Server	6.6	20	20	8.5	60	12	60	15	15	1
Bank DB Server	17.5	140	140	28	35	17.5	140	140	28	70
Bank DB Backup	12.5	1	1	1	1	1	1	1	1	1

In order to draw a colored Petri net, with regards to the relationship between the resources which is depicted by the middle graph and using the sequence, deployment and component diagrams, we consider one location in the colored Petri net per resource. Then, considering the sequence diagram, we specify which method from which resource is related to which method from another resource. After that, per each relationship, a pointed arrow will be drawn from the source to the destination.

In the CPN net model, the color change related to the token entered into a transition will be specified by the type of token existing in the destination transition (the next location of transition). Hence, in this state, the token color will change into the color set 1, namely C_1. With the same reasoning, we can specify the set of colors passed through each transition for all the other transitions. Finally, Fig. (10) will be obtained as a colored Petri net in this case study and Fig. 11 will show the related Petri net after the execution.

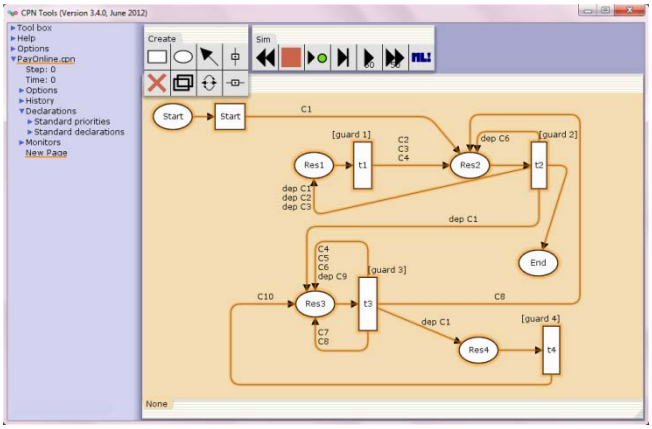

Fig. 10. The colored Petri net of the online shopping system

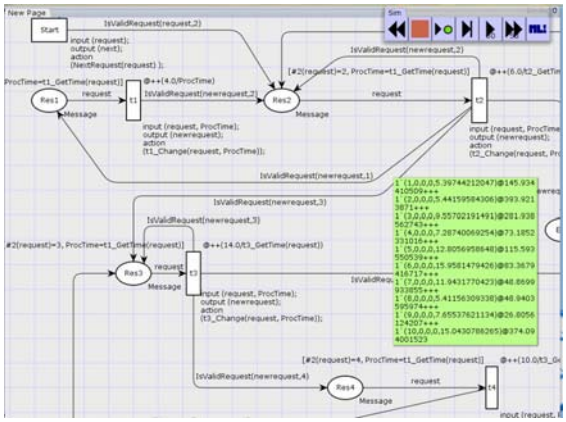

Fig. 11. The colored Petri net of the online shopping system after the execution

5.6 A comparison of the suggested method

In this section, we attempt to compare the suggested method with Emadi's technique in terms of having some of the desired parameters. The result of this comparison is shows in Table 8.

Table 8
A comparison of methods

	Comparison Parameters					
Method	The type of Nonoperational requirements	The components dependence effect on response time	The fuzzy coverage	The uncertainty coverage	The structure coverage	The formality rate
Emadi technique	Efficiency & Reliability	–	–	+	Static & Dynamic	Formal
The proposed	Efficiency	+	–	–	Static & Dynamic	Formal

6. Conclusion and future solutions

In this study, we have attempted to evaluate the software architecture efficiency. In the first phase, we illustrated the software architecture using a semi-formal model. Unified Modeling Language is being utilized for the total process of the software development, which makes it easy to describe the software architecture. This language provides different diagrams with, which we can describe various aspects of the system such as responsibleness, static structure, and dynamic behavior of each component in the system and their interaction with each other and the physical details existing in the system. In addition, we can add the information related to the irresponsibleness requirements as clichés, labels and limitation to the model, which enables us to develop the language in a controlled way so that it can conform to the desired user domain. In the second phase, we turned the semi-formal model into a formal model. We chose the Petri nets from among the pre-mentioned models; because the colored Petri nets are used to describe the software systems by considering the efficiency attributes with demarcations related to the software description. Besides, these nets use a graphical structure to describe the systems based on strong and simple mathematical principles that also support the graphical structure. In the third phase, we attempted to evaluate the software architecture using the formal model. After evaluating the efficiency criteria in the simulated model, one can make the best architecture decisions for the actual model that is about to be built. In this study, the mode-based approaches that are mentioned in order to evaluate the efficiency in the architectural level are reviewed. We can say that almost none of the pre-mentioned approaches studied the effect of independence and dependence of the components at the time of response and efficiency.

As suggested works, one can model and evaluate other parameters and the reliability of the software architecture using other clichés and labels existing in the efficiency and time characteristics index and/or choose other UML diagrams and assess other metrics using this index. In addition, it is possible to apply the fuzzification operation on other net elements such as tokens, locations and even arcs in order to use the fuzzy colored Petri nets optimally.

References

Emadi, S. (2008). Presenting a model in order to study the formal capability of the software architecture, doctorate dissertation, department of computer, Islamic Azad University, Tehran Science and Research Branch, Tehran, Iran.

Akbari E., Noorian R. & Motameni H. (2010). Mapping sequence diagram in fuzzy UML to fuzzy Petri Net. *Iranian Journal of Optimization*, 3, 492-505.

Aziz M. H., Erik L. J., Parnichkun M., and Saha C., (2010), Classification of fuzzy Petri Nets, and their applications. *World Academy of Science, Engineering and Technology*, 72, 394-401.

Balsamo, S. & Grassi V. (2002), Quantitative analysis of software architectures. *Research Report, CS-2002, Department of Information, University Ca Foscari di Venezia*.

Balsamo, S., Inverardi, P., & Mangano, C. (1998). An approach to performance evaluation of software architectures. *Workshop on Software and Performance, WOSP'98, Santa Fe, New Mexico*.

Balsamo, S., Di Marco, A., Inverardi, P., & Simeoni, M. (2004). Model-based performance prediction in software development: A survey. *Software Engineering, IEEE Transactions on*, *30*(5), 295-310.

Balsamo, S. & Marzolla, M. (2003). A simulation based approach to software performance modeling. *ACM SIGSOFT Software Engineering Notes, Proceedings of the 9th European Software Engineering Conference Held Jointly with 11th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, 28(5), 363-366.

Balsamo, B., & Simeoni, M. (2001). Deriving performance models from software architecture specifications. *Proceedings of the 15th European Simulation Multi Conference (ESM2001), SCS-Society for Computer Simulation*.

Banks, J., & Nicol, D. (1999). *Discrete-Event System Simulation*. Prentice-Hall.

Bass, L., Clements, P., & Kazman, R. (1998). *Software Architecture in Practice, 2/E*. Pearson Education India.

Bondavalli, A., Majzik, I., & Pataricza, B. (2003). Stochastic sependability analysis of system architecture based on UML designs. *Lecture Notes in Computer Science*, 2677, springer.

Clements, P., & Klein, K. (2002). *Evaluating Software Architecture Methods and Case Studies*. Addison Wesely.

Cortellessa, V., Marco, A. & Inverardi, P. (2003). Comparing performance models from a software designer perspective. *Technical Report TR SAH/042*.

Cortellessa, V., Di Marco, A., & Inverardi, P. (2004). Three performance models at work: a software designer perspective. *Electronic Notes in Theoretical Computer Science*, *97*, 219-239.

Fukuzawa, K., & Saeki, M. (2002, July). Evaluating software architectures by coloured petri nets. In *Proceedings of the 14th international conference on Software engineering and knowledge engineering* (pp. 263-270). ACM.

Galster, M., Eberlein, A., & Moussavi, M. (2006, June). Transition from requirements to architecture: A review and future perspective. In *Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing, 2006. SNPD 2006. Seventh ACIS International Conference on* (pp. 9-16). IEEE.

Grassi, V., Mirandola, R., & Sabetta, A. (2007, February). A model-driven approach to performability analysis of dynamically reconfigurable component-based systems. In *Proceedings of the 6th international workshop on Software and performance* (pp. 103-114). ACM.

Group, O.M. (1993), UML Profile, for Schedulability, Performance, and Time, OMG document http:/www.omg.org, 2005.

Gyarmati, E., & Strakendal, P. (2002). *Software Performance Prediction-Using SPE* (Doctoral dissertation, Master Thesis Software Engineering, Department of Software Engineering and Computer Science Blekinge Institute of Technology).

Hadipour Sanati, H. (2003). Presenting an executable model from the software architecture with the purpose of efficiency evaluation, Master's thesis, the electrical and computer department, Shahid Beheshty University, Tehran, Iran.

Happe, J., & Firus, V. (2005, July). Using stochastic petri nets to predict quality of service attributes of component-based software architectures. In *Proceedings of the Tenth Workshop on Component Oriented Programming (WCOP2005)*(Vol. 94).

Harounabadi, A. (2006). Utilizing the FUZZY-UML in order to model the irrational systems. The 7th conference on the fuzzy systems, Ferdosi University of Mashad.

Harounabadi, A. (2010). Modeling and evaluating the information systems using the fuzzy colored Petri nets, 16th annual national conference of Iran computer assembly, Sanati Sharif University of Iran.

Haroonabadi, A., Teshnehlab, M., & Movaghar, A. (2008). A novel method for behavior modeling in uncertain information systems. *World Academy of Science, Engineering and Technology*, *41*, 959-966.

Hopcroft, J., & Uliman J. (1997). *Introduction to Automata Theory, Languages and Computations*. Addison Wesley.

Kim, H., Kang, S., Baik, J., & Ko, I. (2007, July). Test cases generation from UML activity diagrams. In *Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing, 2007. SNPD 2007. Eighth ACIS International Conference on* (Vol. 3, pp. 556-561). IEEE.

Jasmine, K. S., & Vasantha, R. (2007, August). Identification of software performance bottleneck components in reuse based software products with the application of acquaintanceship graphs.

In *Software Engineering Advances, 2007. ICSEA 2007. International Conference on* (pp. 34-34). IEEE.

Kant, K., & Srinivasan, M. M. (1992). *Introduction to computer system performance evaluation*. McGraw-Hill College.

Kogut, P., & Clements, P. (1995). Feature analysis of architecture description languages. *In Proceedings of the Software Technology Conference*, Salt Lake City.

Kumar, B., & Jaspernete J. (2012), UML profiles for modeling real-time communication protocols. *Journal of Object Technology*, 9(2), 178-198.

Li, X., & Yu, W. (2001). Object oriented fuzzy Petri net for complex knowledge system modeling. In *Control Applications, 2001. (CCA'01). Proceedings of the 2001 IEEE International Conference on* (pp. 476-481). IEEE.

Makaruk, H., Owczarek, R., & Sakhanenko, N. (2005). Systematic method for path-complete white box testing. *arXiv preprint cs/0503050*.

Marco A. D., (2003), Starting from message sequence chart for software architecture early performance analysis. *Proceding of 2th International Workshop on Scenarios and State Machines: Models, Algorithm (ICSE).*

Motameni, H., Daneefar, I., Bakhshi, J., & Nematzadeh, H. (2009). Transforming fuzzy state diagram to fuzzy Petri net. *Journal of Computer Engineering*, *1*(1), 29-44.

Motameni, H., Movaghar, A. & Bakhshi, J. (2008). Mapping to convert activity diagram in fuzzy UML to fuzzy Petri Net. *World Applied Sciences Journal*, *3*(3), 514-521.

Musa, J. D. (1993). Operational profiles in software-reliability engineering. *Software, IEEE*, *10*(2), 14-32.

Nematzadeh, H., Deris, S. B., Maleki, H., & Nematzadeh, Z. (2009). Evaluating reliability of system sequence diagram using fuzzy Petri net. *International Journal of Recent Trends in Engineering*, *1*(1), 142-147.

Pettit IV, R. G., & Gomaa, H. (2003, October). Improving the reliability of concurrent object-oriented software designs. In *Object-Oriented Real-Time Dependable Systems, 2003. WORDS 2003 Fall. Proceedings. Ninth IEEE International Workshop on* (pp. 262-269). IEEE.

Pettit IV, R. G., & Gomaa, H. (2004, June). Modeling behavioral patterns of concurrent software architectures using Petri nets. In *Software Architecture, 2004. WICSA 2004. Proceedings. Fourth Working IEEE/IFIP Conference on*(pp. 57-66). IEEE.

Poukamali, M. (2006). Improving the software architecture evaluation techniques, Master's thesis, Amirkabir University of Technology, Tehran, Iran.

Reisig, W. (1985). Petri nets: an introduction, volume 4 of EATCS monographs on theoretical computer science.

Sharafi, M. (2006). Presenting a technique to extract and evaluate the irresponsibleness properties based on the formal descriptions of the software architecture. PhD thesis, Islamic Azad University, Science and Research Branch, Tehran, Iran.

Shirazi, N. (2008). Presenting a technique to evaluate and analyze the efficiency based on the software architecture using the colored Petri nets, Master's thesis, Islamic Azad University, Tehran-South branch, Tehran, Iran.

Skene, J., & Emmerich, W. (2003, October). A model-driven approach to non-functional analysis of software architectures. In *Automated Software Engineering, 2003. Proceedings. 18th IEEE International Conference on* (pp. 236-239). IEEE.

Wang, W. L., Pan, D., & Chen, M. H. (2006). Architecture-based software reliability modeling. *Journal of Systems and Software*, *79*(1), 132-146.

Xiao, L., Gang, W., Lin, X. & Maosheng, D. (2005). Reliability analysis of digital protection's software based on architecture. *IEEE/PES Transmission and Distribution Conference and Exhibition: Asia and Pacific Dalian*, China.

Zhang, S. (2006, April). Integrating non-functional properties to architecture specification and analysis. In *Information Technology: New Generations, 2006. ITNG 2006. Third International Conference on* (pp. 112-117). IEEE.