Journal of Advances in Computer Research
Quarterly ISSN: 2008-6148
Sari Branch, Islamic Azad University, Sari, I.R.Iran
(Vol. 3, No. 2, May 2012), Pages: 53-64
www.jacr.iausari.ac.ir

A New Algorithm for Performance Evaluation of Homogeneous Architectural Styles

Sima Emadi[1]✉, Golnaz Aghaee Ghazvini[2]

(1) Department of Computer Science, Maybod Branch, Islamic Azad University, Maybod , Iran
(2) Department of Computer Science, Najafabad Branch, Islamic Azad University, Esfahan, Iran

emadi@maybodiau.ac.ir; aghaee.golnaz@sco.iaun.ac.ir

Received: 2012/04/17; Accepted: 2012/05/05

Abstract

Software architecture is considered one of the most important indices of software engineering today. Software Architecture is a technical description of a system indicating its component structures and their relationships, and is the principles and rules governing designing. The success of the software depends on whether the system can satisfy the quality attributes. One of the most critical aspects of the quality attributes of a software system is its performance. Performance analysis can be useful for assessing whether a proposed architecture can meet the desired performance specifications and whether it can help in making key architectural decisions. Architecture style is a set of principles which an architect uses in designing software architecture. Since software architectural styles have frequently been used by architects, these styles have a specific effect on quality attributes. If this effect is measurable for each existing style, it will enable the architect to evaluate and make architectural decisions more easily and precisely. In this paper an effort has been made to introduce a new algorithm for investigating performance in homogeneous architectural styles based on Markov chains. Moreover, How to use this Markov chains model for performance evaluation is shown and simulation and the implications of this transformation are described completely. Finally, to represent the usage of our proposed algorithm, a case study is presented as an example.

Keywords: *Software architecture, Markov model, Architectural styles, Performance attribute, homogeneous styles*

1. Introduction

The structure of a program or a computing system is its software architecture. This structure consists of software components, their externally visible properties and their inter relationships [1][2]. The architecture of a software system has been identified as an important aspect in software development; because it provides a formal basis to describe and analyze the software system today. One of the most important quality attributes in software architecture is performance. Software architects take advantage of early performance analysis and measurement approaches for a software system based on components in order to evaluate their systems on the basis of performance specifications which are created by component developers [3]. During the last decades, there have been many approaches for evaluating the performance attributes of component-based systems. These approaches have been classified into formal and

informal models. Classical formal models such as queuing networks [4], stochastic process algebras [5], stochastic Petri nets [6] and automata [7] can be used to model and analyze component-based software systems. However, these approaches do not specifically consider performance evaluation of architectural styles using Markov chain. A combination of architectural styles restricting the features/roles of architectural components and allowing relationships among these components within any architecture conforming to that style is referred to as architectural style [8]. Architects use software architectural styles in designing software architecture. Common styles are Batch-sequential, Pipe and Filters, Call and Return and also Fault tolerance [2]. In a batch-sequential style, components are executed in a sequential manner. This means that only a single component is executed in any instance of time. For example, a bank performs a batch of transactions update to a master file in sequence. A parallel style has a set of components running concurrently; a fault tolerant style has a set of back-up components compensating for the failure of the others; call and return style has some components, calling the other components at an indefinite number of times [2][9][10]. In this paper, a new algorithm for performance evaluation of architectural styles is presented. The algorithm is named "PEAS" (Performance Evaluation of Architectural Styles). It consists of modeling the software architecture as a Discrete Time Markov Chain (DTMC), and the DTMC model is then analyzed to get performance attributes of the systems. The unique ability of the approach is to allow for quantitative analysis for performance attribute, so it will make it very suitable for comparing various software architecture as well as component type. This approach is useful both for analysis at the time of system design and for the evaluation of existing systems .The rest of the paper is divided as follows: section 2 introduces an analytical algorithm to performance evaluation of architectural styles. Section 3 describes how to use this new algorithm for performance evaluation of the homogeneous styles. Section 4 illustrates an Example of component-based system for performance evaluation of the system. Conclusion and future works are presented in section 5.

2. An algorithm for performance evaluation of architectural styles

In this section, the 'service time' parameter which is one of the most important performance parameters has been selected. The following algorithm is offered for quantitative evaluation of this parameter in homogeneous architectural styles. The architectural styles can be used for evaluation of performance through the following steps:

- Step1: Defining the architecture with state diagram.
- Step 2: Mapping the state diagram to Markov model.
- Step 3: Creating the transition probability matrix.
- Step 4: Calculating the number of visits of each state in Markov model.
- Step5: Finding performance parameter.

In following, these steps will be described in details:

- **Step1: Defining the architecture with state diagram:** The dynamic behavior of the system is defined by using the state diagram. Supposing that the system has a limited number of components, different components of the system and transfer of current control between the components is defined by the state diagram. The diagram for the state used for this purpose is the UML state diagram.

- **Step 2: Mapping the state diagram to Markov model:** Markov model is a finite state machine with the feature that the probability of transfer from one state to another in it merely depends on the current state of the system rather than the previous states [11][12]. Here the state diagram in the previous step which defines the dynamic behavior of the components is mapped to Markov model. With regard to Markov's feature, this mapping can be a one to one or many to one mapping. In other words, the states of several components may be interdependent while being executed, so they are mapped to one state in Markov model (many to one mapping) and there is a possibility that the components states of the system are independent of each other, in that case they are mapped in separate states (one to one mapping).

- **Step 3: Creating the transition probability matrix:** In order to analyze Markov model, the probability of transfer between various states should be calculated. If Markov model system has m state, a P_{m*m} matrix is considered in which each $P_{i,j}$ shows the probability of transfer from state i to state j in Markov model and can be derived from equation 1. It is worth mentioning that the probability of transfer between two states follows Markov feature; that is transfer from Si to S_j is merely dependent on the current state.

$$P_{i,j} = \begin{cases} t(i,j) / \sum_{n=1}^{m} t(i,n) & state\ \ S_i\ \ reaches\ \ to\ \ state\ \ S_j\ \ for 1 \le i,j \le m \\ 0 & otherwise \end{cases} \quad (1)$$

Where $t(i,j)$ is the number of transfers that occur from state i to state j and $\sum_{n=1}^{m} t(i,n)$ is the sum of transfers that may occur from state i to other states.

- **Step 4: Calculating the number of visits of each state in Markov model:** In this step, it is necessary to compute how many times each state is visit since there is a possibility that the execution control is allocated to a certain state more than once, and this can cause the system to be in a certain state several times while the program is being executed. For example, if in a specific state of Markov model, a component calls another component in a different state several times, it will cause the state in which the called component exists to be met several times. In case Markov model has m states, equation 2 is used to find out the number of visits for each state S_j; in other words, for each state in Markov models this equation be calculated [13]:

$$V_j = q_j + \sum_{k=1}^{m-n} P_{k,j} V_k \quad (2)$$

In the above equation, q_j represents probability that the beginning state in Markov model is S_j state. $P_{k,j}$ shows the probability of transfer from state S_k to state S_j in Markov model. V_k shows the number of visit S_k in Markov model. m is the total number of existing stats in DTMC or Markov model. n is the number of states which does not have any transfer to other states.

- **Step5: Finding performance parameter:** Architecture styles of various components depict the combination and interactions between the components. Usually, interactions between the components which are established by connectors allocate the

specifications of various architecture styles to each of them. For example, in the Batch-sequential style, processing steps or components are independent of each other and are executed sequentially, while in the parallel style, several components are executed concurrently. After the completion of the execution of parallel components, they concede the control to the next component or components.

3. Performance evaluation of homogeneous styles by PEAS algorithm

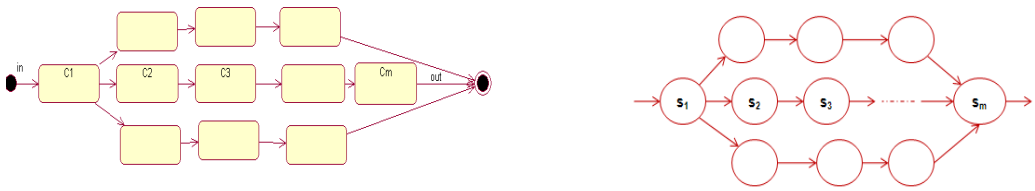
In this stage, efficient parameters are calculated on the basis the specifications of each style as well as the interaction between components. Obtaining relationships for the calculation of efficiency in a style is different from another style, and this comes from the fact that the interaction between components in various styles differs.

In continuation, the stages of algorithms already discussed are enforced on the sequential style, and for other styles, the results of algorithm enforcement are discussed in brief.

3.1 Batch-Sequential Style

In a batch-sequential style, components are executed sequentially. In this type of architecture style, only one component is executed at any instance of time, the control flow is transferred to only one of its successors upon the completion of a component [9].

- **Step1: Defining the architecture with state diagram:** In this style, in each time only one component is executed, therefore in the UML state diagram, a separate state is considered for each component. One of the examples of this style is modeled in figure1.a, where c_1, c_2, …, c_m are software components in a software system. Component c1 transfer flow control to one of its branches subsequent components.



(a) (b)

Figure 1. (a) State diagram for batch-sequential style (b) Markov model

- **Step 2: Mapping state diagram to Markov model:** The transformation from the state diagram to Markov model can be viewed as a mapping of one component to one state. The Markov model is shown in figure1.b.

- **Step3: Creating the transition probability matrix:** In order to calculate the transfer probability between the states, equation 1 is used. After the calculations are carried out, p matrix for sequential style is constructed as follows. As can be seen, the elements of one of the side diameters of the matrix are one and the other elements of the matrix are zero.

$$P = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

- **Step 4: Calculating the number of visits of each state in Markov model:** Suppose the initial state is state S_1 in Markov model, we can calculate the number of visits in each state. In continuation, for instance, some V_j for states S_1, S_2, and S_3 have been calculated.

$$V_1 = q_1 + \sum_{k=1}^{m-1} P_{k,1} V_k = 1$$
$$V_2 = q_2 + \sum_{k=1}^{m-1} P_{k,2} V_k = P_{1,2} V_1 = 1$$
$$V_3 = q_3 + \sum_{k=1}^{m-1} P_{k,3} V_k = P_{2,3} V_2 = 1$$

- **Step5: Finding performance parameter:** In the Batch-sequential style in each period only one component is being executed and the execution control will remain with the same component until the processing is completed. Hence, the time for components' response do not overlap. If we suppose:

  - The service time required to service one request by a software component is shown by T (i).

  - m is the number of sequential components.

  We calculate the total service time given by equation 3 [13]:

$$Service - time - seq = \sum_{i \in m} V(i).T(i) \quad (3)$$

3.2 Parallel style

Software architects to achieve higher speed in computation use parallel style. Parallel style has multiple components running concurrently; this feature has reduced the service time of this style. Parallel style for a particular system has been used repeatedly by architects, resulting in the identification of the efficiency related to the style. Hence, if the effect of parallel style on the performance attributes is available quantitatively, design decisions will be easier for architects.

- **Step1: Defining the architecture with state diagram:** State diagram of this style is shown in figure 2.a, where components c_1, c_2... cm in the dotted oval are running concurrently. These components cooperatively work on a partition of outputs produced by previous component and synchronously release the control to the next subsequent component.

- **Step 2: Mapping state diagram to Markov model:** Figure 2.b shows the Markov model of figure 2.a. A set of cooperative concurrent components in software architecture are modeled to one single state in state diagram.

(a) (b)

Figure 2. (a) State diagram for parallel style (b) Markov model

- **Step3: Creating the transition probability matrix:** In order to calculate the transfer probability between the states, equation 1 is used. After the calculations are carried out, p matrix for parallel style is constructed as follows.

$$P = \begin{array}{c} \\ 1 \\ 2 \\ 3 \end{array} \begin{array}{c} \begin{array}{ccc} 1 & 2 & 3 \end{array} \\ \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix} \end{array}$$

- **Step 4: Calculating the number of visits of each state in Markov model:** Considering the fact that all the parallel components are modeled to only one state s_{pl}, the visit count to state s_{pl} is calculated separately as below:

$$V_j = q_j + \sum_{k=1}^{m-n} P_{k,j} V_k$$

$$V_1 = q_1 + \sum_{k=1}^{3-1} P_{k,1} V_k = 1$$

$$V_2 = q_2 + \sum_{k=1}^{3-1} P_{k,2} V_k = P_{1,2} V_1 = 1$$

$$V_3 = q_3 + \sum_{k=1}^{3-1} P_{k,3} V_k = P_{2,3} V_2 = 1$$

- **Step5: Finding performance parameter:** Finally for all parallel components, the service time given by equation 4 can be calculated [13]:

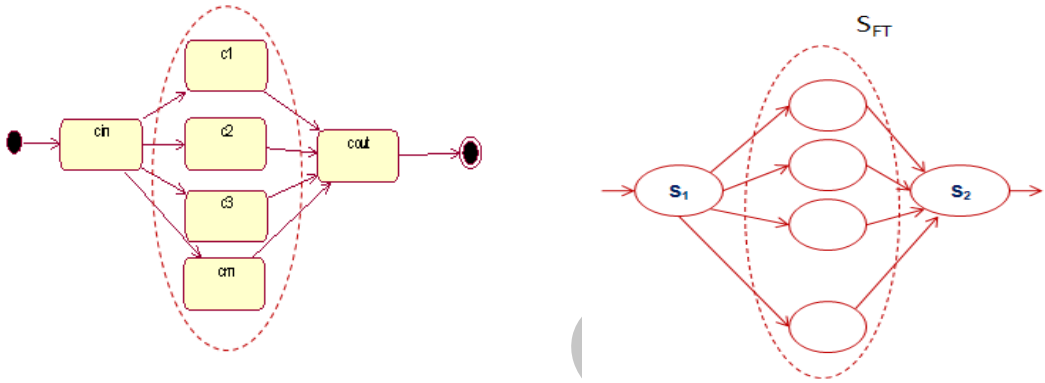
$$Service - time - Parallel = v(s_{pl})[\underset{i \in m}{MAX}(T(i))] \qquad (4)$$

3.3 Fault tolerant style

A fault tolerant architectural style is often used to improve availability of software systems. This style consists of some fault components, including a primary component and a set of backup components. These components (primary and backups) are placed in parallel, so if one component fails, the backup components still provide services and compensate the failure of the primary component. Now, in case the new primary component also fails, another back-up component will replace it and this goes on as long as there is a back-up component to take over its responsibilities, if the primary

component fails. Note that, the transition probability to or from a set of fault components is equal to 1, and implementation of the fault tolerant components are to the same algorithms or data structures. So the response time of these components is identical.

- **Step1: Defining the architecture with state diagram:** State diagram of the fault tolerant style is shown in figure 3.a. Those dotted components c1...cm is set of fault tolerant components, in which components from c2…cm are backup components for primary component c1. In this example, we have merely considered fault tolerant components; and other sequential components seen in the picture are not taken into consideration.



(a) (b)

Figure 3. (a) State diagram for Fault tolerant style (b) Markov model

- **Step 2: Mapping state diagram to Markov model:** Mapping this architecture to Markov model is similar to the parallel style. In this change a set of fault components is changed into a state model. This means that, a set of fault tolerant components c1…ck is mapped into s_{ft} and other components which are not fault tolerant components are modeled into separate states. State model is presented in figure 3.b.

- **Step3: Creating the transition probability matrix:** In order to calculate the transfer probability between the states, equation 1 is used. After the calculations are carried out, p matrix for fault tolerant style is constructed as follows.

$$P = \begin{array}{c} \\ 1 \\ 2 \\ 3 \end{array} \begin{array}{c} \begin{array}{ccc} 1 & 2 & 3 \end{array} \\ \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix} \end{array}$$

- **Step 4: Calculating the number of visits of each state in Markov model:** Considering the fact that all the fault tolerant components are modeled to only one state s_{ft}, the visit count to state s_{ft} is calculated separately as below:

$$V_1 = q_1 + \sum_{k=1}^{3-1} P_{k,1} V_k = 1$$

$$V_2 = q_2 + \sum_{k=1}^{3-1} P_{k,2} V_k = p_{1,2} V_1 = 1$$

$$V_3 = q_3 + \sum_{k=1}^{3-1} P_{k,3} V_k = p_{2,3} V_2 = 1$$

- **Step 5: Finding performance parameter:** Now, to calculate the time spent by fault components, for c_1 and c_{k+1} which are executed sequentially, the service time is calculated like the sequential style but as for fault tolerant components, the service time will be calculated in a different manner. In this set of fault components, if only one of the components performs the operations appropriately and does not fail, there will be no need for other components.
  The probability that at least one component out of the set of m fault components has performed its function correctly will be shown by:

$$1 - (\prod_{i \in m} (1 - p_i))$$

  Where, p_i is the probability of each component performing appropriately.

In the end, the response time for fault tolerant components is calculated by the following equation:

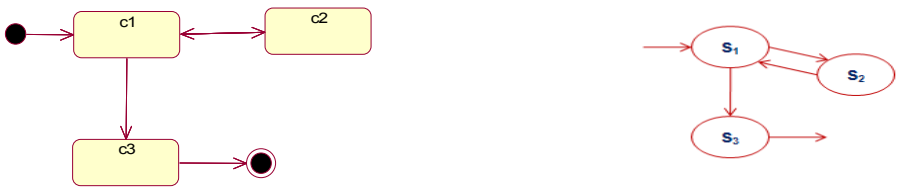
$$Service - time - Fault = V(s_{fi})[1 - \prod_{i \in m} (1 - p_i)]T(i) \quad (5)$$

3.4 Call and Return style

In this style, the execution of one component may need the services provided by other components. So a calling component can invoke a called component a number of times. For each invocation, the caller performs a context switch by granting a temporary control to the called component and then waits for its response. Once receiving the response, the caller component runs the execution from where it left. Eventually, the caller component transfers its complete control to its subsequent component. Therefore the called component can execute many times.

- **Step1: Defining the architecture with state diagram:** An example of this style is shown in figure 4.a, where component c_1 is a caller component and component c_2 is a called component. c_1 may request some services provided by c_2 , so c_1 can invoke c_2 from zero to many times. This means that c_1 component immediately gives the flow control to c_3 component or calls c_2 during its execution and before finishing its work.

- **Step 2: Mapping state diagram to Markov model:** The state model is shown in figure 4.b. Note that component c1 can invoke c_2 from zero to an indefinite number of times. The transferring of a Call and Return style is similar to the batch-sequential style. A state represents an execution of a component. The transfer among states happens when the execution of one component is completed and the flow control is given to the next component, or while a component is being executed, another component is called and the flow control is temporarily given to it.



(a) (b)

Figure 4. (a) State diagram for Call and Return style (b) Markov model

- **Step 3: Creating the transition probability matrix:** In order to calculate the transfer probability between the states, equation 1 is used. After the calculations are carried out, p matrix for fault tolerant style is constructed as follows.

$$P = \begin{matrix} 1 \\ 2 \\ 3 \end{matrix}\begin{bmatrix} 0 & P_{12} & P_{13} \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

- **Step 4: Calculating the number of visits of each state in Markov model:** Suppose the initial state is state S_1 in Markov model, we can calculate the number of visits in each state. In continuation, for instance, some V_j for states S_1, S_2, and S_3 have been calculated.

$$V_1 = q_1 + \sum_{k=1}^{3-1} P_{k,1} V_k = q_1 = 1$$

$$V_2 = q_2 + \sum_{k=1}^{3-1} P_{k,2} V_k = P_{1,2} V_1 = P_{1,2}$$

$$V_3 = q_3 + \sum_{k=1}^{3-1} P_{k,3} V_k = P_{1,3} V_1 = P_{1,3}$$

- **Step5: Finding performance parameter:** If the number of call and return components is more than two, and there is a possibility of call and return among several components in figure 4, we will consider an M_{m*m} matrix to show the number of transfers among call and return components, where m is the total number of caller and called components.
  In matrix M, M_{ij} indicates the number of component j calls by component i. So the response time for these components is calculated by the following equation:

$$Service-time-caand\,\mathrm{Re} = T(caller) + \sum_{i \in m} \sum_{j \in m} M_{ij} T(j) \qquad (6)$$

4. Case Study

An Example of line examination that contains architectural styles is used to validate the correctness of the new algorithm [9]. The data about the software architecture is summarized in table1 [9]. The expected time spent by the application in component i per visit is already known, this time can either be obtained experimentally or may be known a priori.

Table 1. Example system execution behavior

Service time spent in components(in secs)			
Service time in component$_1$	0.01	Service time in component$_6$	0.30
Service time in component$_2$	0.03	Service time in component$_7$	0.20
Service time in component$_3$	0.005	Service time in component$_8$	0.04
Service time in component$_4$	0.005	Service time in component$_9$	0.01
Service time in component$_5$	0.01		
The probability that fault tolerant components run correctly			
Component 3	0.8	Component 4	0.7

The figure 5 shows the State diagram of this system. Basically this system is working in sequential manner in addition to the following. Components DBMS1 and DBMS2 are fault components. Components Identifier and Messenger are executing in parallel. Components Account Manager and Helper are call and return components. Markov Model of architectural styles is shown in figure 6.

Based on the new algorithm that presented in previous section, and the information of this system, the performance parameter of this system is presented in the following:

Service time for components c_1 and c_2 and c_7 are:

$$tm - seq = \sum_{i \in 1,2,7} V(i)T(i) = 1(0.01) + 1(0.03) + 1(0.01) = 0.05$$

Service time for Fault components c3 and c4 are:

$$tm - fault = V(S_{fi})[1 - (\prod_{i \in 3,4} (1 - p_i))]T(i) = 1(0.94)0.005 = 0.0047$$

Service time for Parallel components c5 and c6 are:

$$tm - parallel = V(S_{pl})[\underset{i \in 5,6}{Max}\, T(i)] = 1\,[0.30] = 0.3$$

Service time for Call and Return components c5 and c6 are:

$$tm - caand\,\mathrm{Re} = T(caller) + \sum_{i \in 7,8} \sum_{j \in 7,8} V_{ij} T(j) = 0.02 + 2(0.04) = 0.1$$

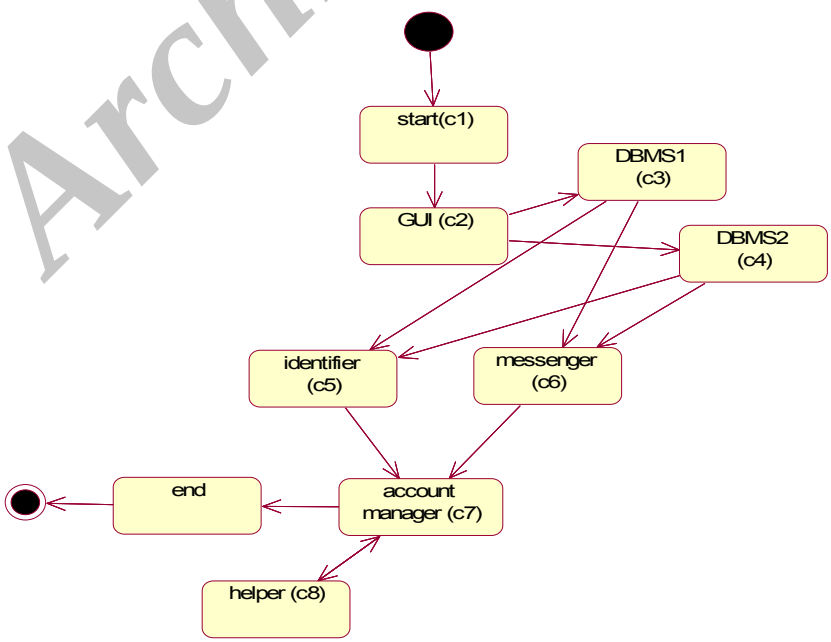


Figure 5. State diagram of the system

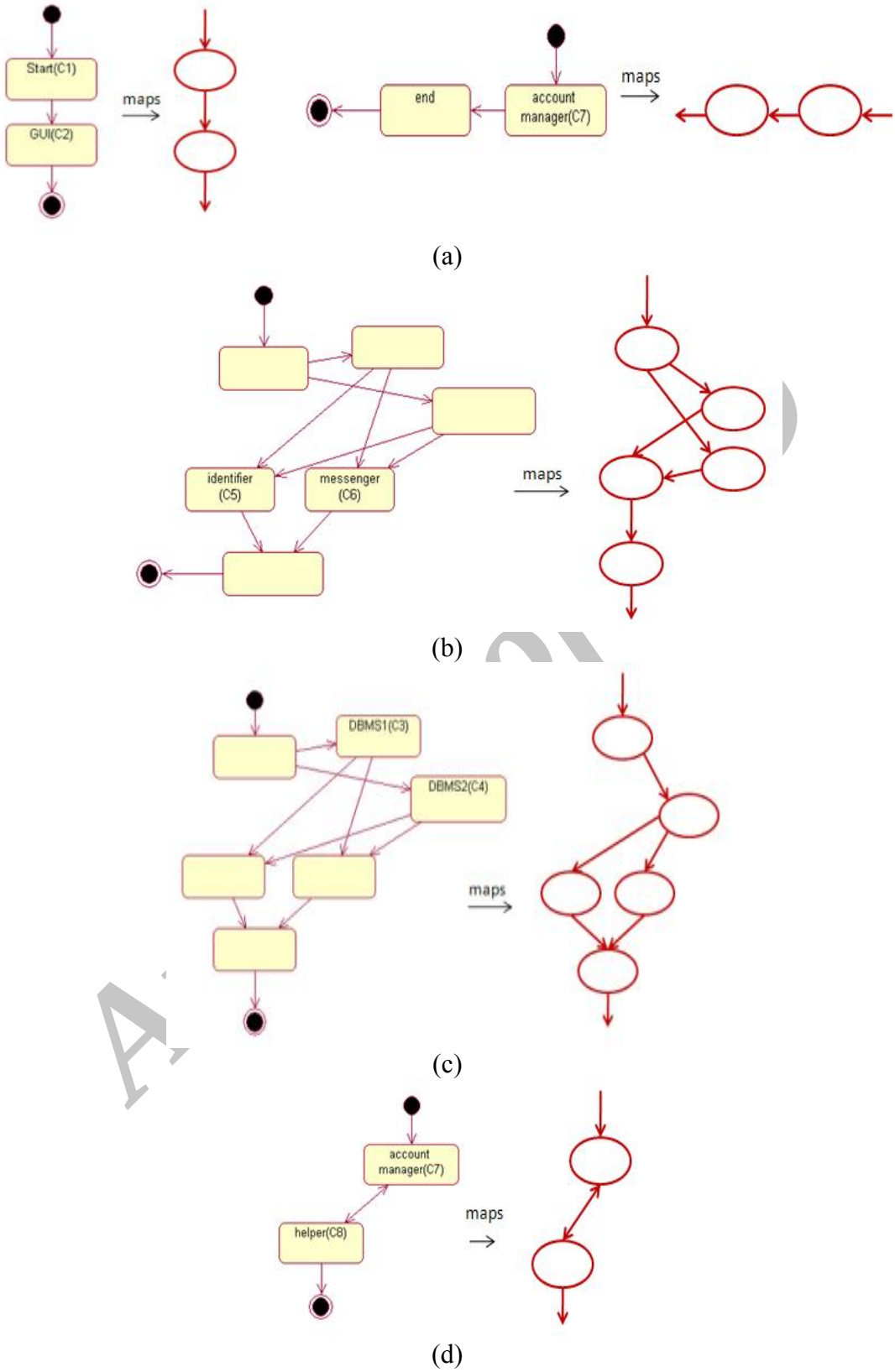


Figure 6. (a) Batch-sequential style (b) parallel style(c) fault tolerance style (d) call and return style

5. Conclusion and Future work

In this paper, a new algorithm was presented for performance evaluation of homogenous architectural styles such as batch-sequential, parallel, Fault tolerant, Call and return style. Our algorithm initially models the system as a Discrete Time Markov. This algorithm was named PEAS. This model provides the visit counts to different states and can calculate the total service time (response time) of the software system.

In this paper, Performance evaluation of homogeneous architecture styles was emphasized. In our future research works, performance evaluation of heterogeneous architecture styles is to be taken into consideration. Moreover, other non functional requirements on the homogeneous and heterogeneous architecture styles will be evaluated.

Also, automatic Tools are not developed for PEAS algorithm, so far. In developed Tools, some other input performance parameters will be considered for evaluating other non-functional requirement such as throughput, latency, data transmission time, Workload, Resource utilization, channel capacity, completion time, bandwidth, relative efficiency, scalability, compression ratio, instruction path length and speed up.

6. References

[1] K. Kant, "Introduction to computer system performance evaluation", 1993.

[2] L. Bass, P. Clements, R. kazman, "Software Architecture in Practice", 3rd edition, SEI Series in Software Engineering, Addison-Wesley, 2012.

[3] S. Gokhalea, , W. Eric Wong, J.R. Horganc and S. Trivedi, "An analytical approach to architecture-based software performance and reliability prediction", An International Journal of performance evaluation, ELSEVIER, vol 58, Dec.2004, pp. 391-412, doi:10.1016/ j.peva. 2004.04.003.

[4] S. Balsamo, V.D.N. Persone and P. Inverardi, "A review on queuing network models with finite capacity queues for software architectures performance prediction", An International Journal performance evaluation, ELSEVIER, vol. 51, Feb. 2003, pp.269-288, doi:10.1016/S0166-5316(02)00099-8.

[5] H. Hermanns, U. Herzog and J.P. Katoen, "Process algebra for performance evaluation", An International Journal Theoretical Computer Science, elsevier vol. 274, Mar. 2002, pp.43-87, doi:10.1016/j.peva.2009.07.007.

[6] S. Emadi and F. Shams, "From UML component diagram to an executable model based on Petri Nets", Proc. the Third International Symposium on transformation Technology, Aug.2008, pp.2780-2787, doi:10.1109/ITSIM.2008.4631945.

[7] M. sharafi, "Extending Team Automata to Evaluate Software Architectural Design", Proc. 32nd Annual IEEE International Computer Software and Applications Conference,2008 , pp. 393-400

[8] K. Khodamoradi, J. Habibi and A. Kamandi, "Architectural Styles as a Guide for Software Architecture Reconstruction",13th International CSI computer Science, Kish Island, Persian Gulf, Iran, 2008.

[9] W. Wang, D. Pan and M. Chen, "Architecture-based software reliability modeling", Journal of System and Software, vol. 79, Jan. 2006, pp. 132-146, doi: 10.1016/j.jss.2005.09.004.

[10] S. Ramamoorthy and S. P. Rajagopalan, "Component-Based Heterogeneous Software Architecture Reliability(COHAR) Modeling", International Journal on Computer Science and Engineering, vol. 02, 2010, pp. 1280-1285.

[11] Franz Brosch, Heiko Koziolek, Barbora Buhnova, Ralf Reussner, "Architecture-based Reliability Prediction with the Palladio omponent Model", accepted in IEEE Transactions On Software Engineering, SEPTEMBER 2011.

[12] Alistair Sinclair, "Markov Chain Monte Carlo: Foundations & Applications", lecture note, 2009.

[13] S. M. Sharafi, G. Aghaee Ghazvini, "An Analytical Model for Performance Evaluation Of Software Architectural Styles", International Conference on Software Technology and Engineering(ICSTE), Vol 01 , pp 394-398, 2010.