Journal of Advances in Computer Research
Quarterly pISSN: 2345-606x eISSN: 2345-6078
Sari Branch, Islamic Azad University, Sari, I.R.Iran
(Vol. 5, No. 3, August 2014), Pages: 85-100
www.jacr.iausari.ac.ir

An Analytical Algorithm of Component-Based Heterogeneous Software Architectural Styles Performance Prediction

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

(1) Department of computer, dolatabad Branch, Islamic Azad University, Isfahan, Iran
(2) Department of Computer, College of Engineering, Yazd Science and Research Branch, Islamic Azad University, Yazd, Iran

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

Received: 2013/11/19; Accepted: 2014/02/19

Abstract

With regard to the society's need for complicated software and high level of expenses on its development, it is necessary to take all stakeholders' requirements and the demands into consideration, before any investments and put on the design and utilization stages. Software architecture is a technical description of a software system that indicates components and their relationships between them. In fact architecture style is a set of principles used by a software architect to design software architecture. Nowadays, this is a common behavior among the software architects in designing any software. As "Performance" is the most important qualitative features chosen for the assessment, the main objective of this research is studying the effect of various styles on its non-functional requirements, using Markov model, so that the architect can choose a suitable style based on qualitative and precise criteria. In this paper with regards to the results obtained based on homogeneous style, an algorithm has been presented to generalize the assessment method for the heterogeneous styles. Finally, to represent the correctness of the proposed algorithm, an illustrative example has been presented.

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

1. Introduction

Software architecture consists of components, connectors and configurations, represents the structure of software system. The architecture of a software system has been identified as an important aspect in software development; as it provides a formal basis to describe and analyze the software systems. Performance is one of the most important quality attributes in software architecture. Software architects take advantage of early performance analysis and measurement approaches for a software system based on components, so that evaluate their systems on the basis of performance specifications which are created by component developers [1]. Over 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 [2], stochastic process algebras [3], and stochastic Petri nets [4], coloured petri net [5] and automata [6] can be used to

model and analyze the component-based software systems. In our previous works, we proposed a new algorithm for performance evaluation of homogeneous software architecture based on various styles [7, 8]. Ramamurthy et al. have represented an analytical model for component based heterogeneous software architecture reliability. This algorithm is based on Markov Chain properties in order to compute the reliability on heterogeneous software architecture consisting of various styles [9]. Borsch et al. have introduced a reliability modeling and prediction technique that considers the relevant architectural factors of software systems and explicitly models the component usage profile and execution environment. This work has built upon the Palladio component model [10]. 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 [11, 12]. Architects use software architectural styles in designing software architecture. Common styles are Batch-sequential, Pipe and Filters, Call and Return and also Fault tolerance [13]. 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, 13, 14]. In this paper, a new algorithm for performance evaluation of architectural styles is presented. The algorithm is called extended "PEAS" [7] (Performance Evaluation of Architectural Styles). It consists of software architecture modeling 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 allows quantitative analysis for performance attribute, so it will make algorithm suitable for comparing various software architecture and component type. This algorithm is useful for both analyses at the time of system design as well as 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 heterogeneous architectural styles. Section 3 illustrates an Example of component-based system for performance evaluation of the system. Conclusion and future works are presented in section4.

2. An algorithm for performance evaluation of architectural styles

In this section, the 'response time' parameter which is one of the most important performance parameters has been chosen. The following algorithm is offered for quantitative evaluation of this parameter in heterogeneous architectural styles. Quantitative evaluations of 'service time' parameter in homogeneous architectural styles have been done in previous work [7, 8]. The architectural styles can be used for evaluation of performance through the following steps:

Step1: Defining the architecture with state diagram.

Step 2: Identifying basic styles in heterogeneous architecture based on system design features.

Step 3: Mapping the state diagram to Markov model.

Step 4: Integrating Markov models to create an overall Markov model.

Step5: Creating the separate sets for each style and enforcing limitations.
Step 6: Creating the transition probability matrix.
Step 7: Calculating the visit number of each state in Markov model.
Step 8: Evaluating the efficiency of the model.
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, transfer of current control between different components is defined by the state diagram. The diagram for the state used for this purpose is the UML state diagram.

Step 2: Identifying basic styles in heterogeneous architecture based on system design features: The styles existing in software architecture can be identified with regard to design features and the abstract software system, which describes the interactions and relationships between components. For instance, interactions between components can include a request for service made by one component to another (the call and return style) or the cooperation of several components to improve the system fault tolerance (fault tolerance style). In contrast, architectural styles may have commonalities in heterogeneous architecture. That is, a component or components may belong to several different styles; of course, in situations where they do not put any harm to the performance trend accuracy. For example, a component of the parallel style cannot be concurrently considered as a support component for another component because it disturbs the accuracy of the performance logic [13]. In this stage, there are separate sets, with the same number to the basic identified styles in architecture, each of which belongs to one basic style, and comprises the components of the same style. If architecture G has x components, and each architectural component is shown with Ca, the following sets are defined to separate the components of each style from another. The definitions of these sets have been summarized in figure1. Set B is created for the components of the Batch-sequential style. In this set, the components belonging to the Batch-sequential style (Ca.e Batch-sequential style) are placed. The number of members in set B is shown with No.Batch (0<a<No. Batch). Set P is created for the components of parallel style. In this set, the components belonging to the parallel style (CaE Parallel style) are placed. No.parallel variable shows the number of members in set P.

$$G = \{C_\alpha \mid \text{component } C_\alpha \in B \cup P \cup F \cup S \cup C \mid 1 \le \alpha \le x\}$$

$$B = \{C_\alpha \mid \text{component } C_\alpha \in \text{Batch sequential style}, \quad \cdot \le \alpha \le \text{No.batch}\}$$

$$P = \{C_\alpha \mid \text{component } C_\alpha \in \text{parallel style}, \quad \cdot \le \alpha \le \text{No.paralle l}\}$$

$$F = \{C_\alpha \mid \text{component } C_\alpha \in \text{fault tolerant style}, \quad \cdot \le \alpha \le \text{No.fault}\}$$

$$C = \{C_\alpha \mid \text{component } C_\alpha \text{ is caller in call and Returnstyle}, \quad \cdot \le \alpha \le \text{No.caller}\}$$

$$S = \{C_\alpha \mid \text{component } C_\alpha \text{ is caller in call and Returnstyle}, \quad \cdot \le \alpha \le \text{No.caller}\}$$

Figure1.Defination of component sets

Set F is created for the components of fault tolerance style. In this set, the components that belong to this style are placed (CaE Fault tolerance style). No.fault variable shows the components existing in this set. Set C includes the caller components, which may

call one or several other components during their performance (Ca is caller in call and return style). The number of caller components in heterogeneous architecture is shown by No.callee. Ultimately, set S has been considered for callee components in architecture (Ca is callee in call and return style). No.callee variable shows the number of these callee components in the architecture. It should be noted that if a component belongs to more than one particular style, it should be placed in the sets related to both styles, and this will bring about a commonality between the intended sets. For instance, a component may call another while sequentially performing its duty. In this case, this component should be considered separately in both B and C sets.

Step 3: 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 [14, 15 and 16]. 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 it is possible 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 4: Integrating Markov models to create an overall Markov model: The resulting Markov model in previous section shows the heterogeneous architecture of the system. Here, the central point is that the number of states of Markov model is often less than that of state diagram because if there are several components in the architecture, which are executed in a parallel form, they are mapped in one state in Markov model.

Stage 5: Creating the separate sets for each style and enforcing limitations: In heterogeneous architecture, one component or more may belong to more than a particular style, which will cause commonalities between styles; and as a result, commonalities between states of Markov model of styles. It is assumed that the overall Markov model of architecture that was obtained in stage 4 has m states, and each state in this model is shown with Si. It is obvious that each Si state in this model has been obtained from one to one, or many to one mapping of state diagram components. In order to analyze the conditions that has caused commonalities between the states of Markov model. First, sets should be defined for separating the states of Markov model for each basic style as follows:

δ_B Set consists of states of Markov model of sequential style, where Si has been created from the mapping of a sequential component existing in the state diagram. In other words, a sequential component Ca in the state diagram is mapped to a separate state Si in Markov model of sequential style (Ca maps to state Si). In addition to the definition of δ_B set, to calculate in continuation, another set called S.B has been created for Markov model of sequential style, which includes the index of states of Markov model of sequential style. δ_P set includes states of Markov model of parallel style, where Si has been created from the mapping of many-to-one of the parallel components. In other words, several parallel components in the state diagram are mapped to a separate state in Markov model of parallel style. S.P set includes the index of parallel states existing in δ_P. δ_F set includes states of Markov model of fault tolerance style, where Si has been

obtained from the many-to-one mapping of main and support components. In other words, several interdependent main and support components are mapped to a separate state in Markov model of fault tolerance style. S.F set consists of the index of states existing in δ_F.

δ_s, δ_c set include caller and callee states in Markov model of call and return style; and ultimately, S.S and S.C sets represent indices of caller and callee states. respectively Commonalities between components and consequently commonalities between Markov models call for further analysis and study; hence, limitations are taken into consideration in continuation. These limitations describe the conditions for commonalities between the states in Markov model, and have been described in figure as follows [12]. The limitations defined by Wang have been defined so that they do not put any harm to the performance trend of demand execution by the system. Dark areas in the figure 2 show the commonalities between states of Markov model.

For instance, as seen in the figure 2, a sequential component in a state of Markov model of sequential style δ_B cannot be considered as a callee component in δ_s set because it causes trouble in the logic for the sequential execution of components $\delta_B \cap \delta_s = \phi$. Whereas a sequential component can call one or more components during its sequential execution; hence, there are commonalities between the set of δ_B, δ_C states $\delta_B \cap \delta_C \neq \phi$.

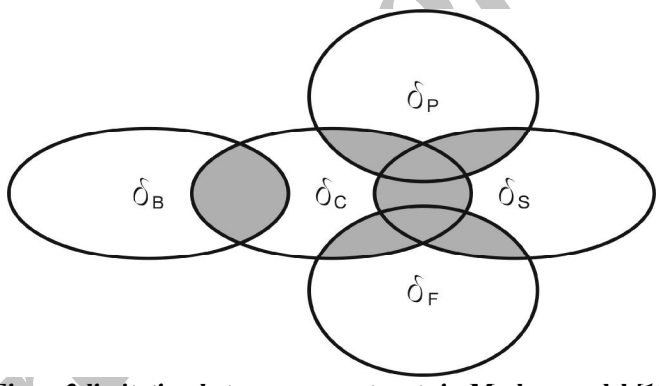

Figure2.limitation between separate sets in Morkov model [12]

Step 6: Creating the transition probability matrix: To analyze Markov model, the probability of transfer between various states should be calculated. If Markov model system has m state, a Pm*m matrix is considered in which each Pi,j shows the probability of transfer from state i to state j in Markov model and can be derived from equation1[13]. It should be noted that the probability of transfer between two states follows Markov feature; that is transfer from Si to Sj is merely dependent on the current state.

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

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 7: Calculating the visit number of each state in Markov model: In this step, it is necessary to compute how many times each state is visited 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, equation2 is used to find out the number of visits for each state Sj . In other words, this equation is calculated for each state in Markov models [17]:

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

In the above equation, qj represents probability that the beginning state in Markov model is Sj state. Pk,j shows the probability of transfer from state Sk to state Sj in Markov model. Vk shows the number of visit Sk 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. The number of times each Sj operation depends on the Sk states that will reach Sj; of course, by a exception of state in which $S_k \in \delta_s$; because returning from a callee component will not affect the operation number of caller component. For example, suppose a component is in Sj state and call another component that is in Sk state several times during its execution, it should temporarily concede the execution control to that component. After the completion of the demand, the callee component in Sk State returns the result to the caller component, and the caller component resumes its execution from where it had postponed. Therefore, it can be said that the execution control returns from a callee component to its caller, the execution of the caller component does not start from the beginning. Hence, the number of return times from callee will not affect the number of execution of caller component.

Step 8: Evaluating the efficiency of the model:

The first state: Components of the basic styles should be separable from one another.

In this stage, to calculate efficiency, it is necessary to define sets which include the response time of components of a particular style. In conditions where the components of different styles are separable from one another, to create sets, the following steps are taken: Suppose state Si in Markov model belongs to the parallel style $S_i \in \delta_p$ or fault tolerance style $S_i \in \delta_F$, then for this state Si, a separate set is defined. This set comprises the response time of components executed in that state Si; that is, in order to complete this set, the name of the ith component Ci is substituted with the response time of that component. However, in case one state or more in Markov model belong to the sequential style $S_i \in S_B$ or call and return style $S_i \in \delta_C$, there will be no need to form separate sets for each state because in each state, whether Sequential or Call and Return, only one component is executed. Hence, three separate sets; namely callee-time, caller-time and Batch-time, are defined in each of which the response time of sequential

components, caller components, and call components are sequentially entered. In continuation, there is an assessment need for the index of components existing in Batch-time, parallel-time, fault-time and other sets. Hence, 4 separate sets (B.t,P.t,F.t,C.t,S.t) have been defined which include the index for the response time of components of basic styles. Figure 3 shows the assumptions and markings used for these sets. In continuation, to know more, the response time for sequential components will be shown with T_{Bi}, the response time for parallel components with T_{Pi}, the response time for the caller components with T_{Ci}, the response time for the call components with T_{Si}, and the response time for the fault components with T_{Fi}.

The second state: The components of basic styles should not be separable from one another.

If there are commonalities between the components of basic styles, it means that a component belongs to more than one particular style. Hence, in order to map the name of i_{th} component on its response time, limitations should be considered in different sets as shown in figure 3.

1- If there is a sequential component in the architecture, which calls other components while executing its sequential performance as a caller, its response time is only considered in the caller set (caller-time). For instance, the component C_1 in the state diagram of figure 4 is a sequential component because after the completion of the process, it concedes the execution control to component C_2. Moreover, component C_1 also plays the role as caller because it calls components C_3 and C_4 during its execution. If the response time of component C_1 is entered as a sequential component in Batch-time set and once again as a caller component in caller-time set, its response time will be considered more than once in the next calculations. Therefore, after surveys, we concluded that the response time of components which play the sequential and caller roles are merely considered in the caller-time set.

$$\textbf{Batch.time} = \{T_i \mid C_i \in \textbf{B}, C_i \textbf{ maps to } T_{Bi}\}$$
$$\textbf{B.t} = \{i \mid T_{Bi} \in \textbf{Batch.time}\}$$
$$\textbf{Parallel.time} = \{T_i \mid C_i \in \textbf{P}, C_i \textbf{ maps to } T_{Pi}\}$$
$$\textbf{P.t} = \{i \mid T_{Pi} \in \textbf{parallel.time}\}$$
$$\textbf{Fault.time} = \{T_i \mid C_i \in \textbf{F}, C_i \textbf{ maps to } T_{Fi}\}$$
$$\textbf{F.t} = \{i \mid T_{Fi} \in \textbf{Fault.time}\}$$
$$\textbf{Caller.time} = \{T_i \mid C_i \in \textbf{C}, C_i \textbf{ maps to } T_{Ci}\}$$
$$\textbf{C.t} = \{i \mid T_{Ci} \in \textbf{Caller.time}\}$$
$$\textbf{Callee.time} = \{T_i \mid C_i \in \textbf{S}, C_i \textbf{ maps to } T_{Si}\}$$
$$\textbf{S.t} = \{i \mid T_{Si} \in \textbf{Callee.time}\}$$

Figure3.Response time sets of component

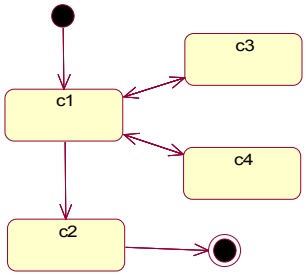

Figure4.Sequetial component has role of caller component

2-If there is a parallel component in the heterogeneous architecture that calls other components during parallel execution with other components, or is called by other components, three states should be taken into consideration as follows:

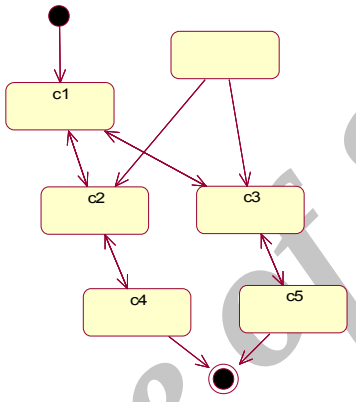

Figure 5.Architecture component has role of parallel, caller and callee component

The first state: If the parallel component in state Si of Markov model plays roles as caller and callee in the architecture, its service time is considered as the caller component in parallel-time set; that is, first the response time of that component is calculated as the caller; and then, the time obtained in the parallel-time set is taken into consideration.

For instance in figure 5, C_2 and C_3 components are executed parallel to one another, and during the parallel execution, C_2 component calls C_4 component. Moreover, C_3 component calls C_5 component. The two components C_2 and C_3 are called by C_1 component. Hence, it can be said that C_2 and C_3 components are three different roles in the architecture and that they belong to parallel, call and return styles. Therefore, the response time of C_2 and C_3 components are first calculated as callers, and then will be considered in the parallel-time set.

The second state: If the parallel component merely plays the role as caller in the architecture, its response time will be considered as the caller component in the parallel-time set. For example, if in figure 6, C_1 and C_2 components are concurrently executed together, just one state will be considered for them in Markov model, and for this state, a parallel-time set is created. In parallel-time set created for S_{pl} state, the response time of C_1 and C_2 components is not included because these components play the role as callers in addition to the parallel role. For example, C_1 calls C_2 component, and C_2 component calls C_3 component. Therefore, first their response time as the caller will be calculated. Then, this time is taken into consideration in the parallel-time set.

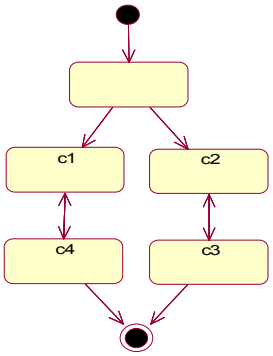

Figure 6.Architecture component has role of parallel and caller

The third state: If the parallel component in Si state merely has the callee role in the architecture, its response time of the component is considered in the parallel-time set. For instance in figure 7, C_2 and C_3 components are executed parallel to one another and may be called by C_1 component. The response time of these components is then considered as the callee in the parallel time set. It should be noted that the response time of components as callee is in fact the response time of the component itself.

3-If there is a component in fault set components, which the components call, first its response time is calculated as the caller, and then it will be entered in the fault time set.

4-If the callee component in the architecture plays another role such as caller, parallel and support in the heterogeneous architecture, the response time of this component will be entered in the caller-time set. For example, C_2 component in figure 8 has three roles: parallel, caller and callee. The response time of this component is considered in the callee-time set.

5- If a caller component in Si state of Markov model has the roles of sequential, parallel, fault and callee, too, the response time of this component will be entered in the caller-time only if this component does not play the role of parallel and fault. That is, if two components also have the role of callee along with their parallel role, their response time will be calculated separately, and the result of this calculation will be entered in the parallel-time set, and ultimately their maximum will be taken into consideration.

Now, with regard to the sets defined in figure 9, two states are studied to calculate the response time: In conditions where the architecture components are distributed and installed on various machines, the delay time between the components should be taken into account in the response time.

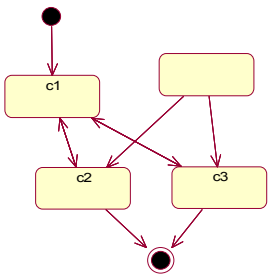

Figure 7.Architecture component has role of parallel and callee

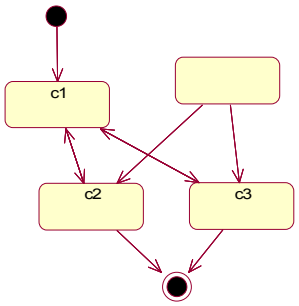

Figure 8.Architecture component has role of fault,caller and callee component

$$Batch.time = \{T_i \mid C_i \in B - C, C_i \quad maps \quad to \quad T_{Bi}\}$$

$$parallel.time = \begin{cases} T_i \mid C_i \in P \cap C \cap S, C_i \quad maps \quad to \quad T_{Ci} \\ T_i \mid C_i \in P \cap C, C_i \quad maps \quad to \quad T_{Ci} \\ T_i \mid C_i \in P \cap S, C_i \quad maps \quad to \quad T_{Pi} \end{cases}$$

$$caller.time = \{T_i \mid C_i \in (C \cup S) - (P \cap F)C_i \quad maps \quad to \quad T_{Ci}\}$$

$$callee.time = \{T_i \mid C_i \in S \cap (F \cup C \cup P), C_i \quad maps \quad to \quad T_{Si}\}$$

$$Fault.time = \begin{cases} T_i \mid C_i \in (F \cup S) - C, C_i \quad maps \quad to \quad T_{Fi} \\ T_i \mid C_i \in (F \cup S) \cap C, C_i \quad maps \quad to \quad T_{Ci} \end{cases}$$

$$B.t = \{i \mid T_{Bi} \in Batch.time\} \qquad p.t = \{i \mid T_{Pi} \in parallel.time\}$$

$$c.t = \{i \mid T_{Ci} \in caller.time\} \qquad s.t = \{i \mid T_{Si} \in callee.time\} \qquad f.t = \{i \mid T_{Fi} \in Fault.time\}$$

Figure 9.Response time sets of component

After the analysis of the distribution conditions of components, the architect can use the relationships presented in previous parts in order to calculate the delay time. After separate calculation of the response time of each style, the obtained times will be added together in order to find the response time for the whole system. In continuation of the last stage, after the calculation of the response time for each style separately, the total times should be calculated. This time will be the response time of the system with heterogeneous architecture.

$$response.time.Batch.sequential = \sum_{\substack{i \in S.B \\ j \in B.t}} V_i.T_j$$

$$response.time.parallel = \sum_{S_{si} \in S.P} V_{S_{si}}.[MAX_{i \in p.t}(T_i)]$$

$$response.time.Fault = \sum_{S_{s_i} \in S.t} V_{S_{si}}.[1 - (\prod_{i \in F.t}(1 - P_i)).T_i]$$

$$response.time.\text{call and Return} = \sum_{S_{si} \in s.c} V_{S_{si}}[(T_{i \in c.t}) + \sum_{j \in s.s} V_j.T_j]$$

Figure10.Response time of styles if architecture components are in the same machine

4- Illustrative example

An Example of a software system containing architectural styles is used to validate the correctness of the new algorithm. The architecture is composed of several components that each has a specific response time. These components are implemented in two separate Machines MC1 and MC2.Components C_1… C_{14} are on MC1 and other components are running on MC2. The system information and the performance parameter of components is presented in the table1. 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.

Table1. An Example of response time of the component based software system

Response time per visit:(in Secs)			
Response time of component C_1	0.01	Response time of component C_2	0.02
Response time of component C_3	0.02	Response time of component C_4	0.1
Response time of component C_5	0.04	Response time of component C_6	0.01
Response time of component C_7	0.01	Response time of component C_8	0.01
Response time of component C_9	0.1	Response time of component C_{10}	0.03
Response time of component C_{11}	0.02	Response time of component C_{12}	0.03
Response time of component C_{13}	0.03	Response time of component C_{14}	0.2
Response time of component C_{15}	0.1	Response time of component C_{16}	0.01
Response time of component C_{17}	0.2		
The probability that fault tolerant components run correctly:			
Run correctly C_6	0.7	Run correctly C_7	0.9
Run correctly C_{12}	0.3	Run correctly C_{13}	0.6

Step1: Defining the architecture with state diagram: The state diagram of system architecture is shown in Figure 11. Components C_3, C_4 and C_{10}, C_{11} are categorized into parallel style and components C_6, C_7 and C_{12},C_{13} are categorized into fault tolerance style. Components C_1, C_2…, C_8 have caller/callee relationships. C1 is the first Caller component that may call Callee components C_2, C_3, C_4 from zero to an indefinite number of times. Also components C3 and C4 may call components C_5, C_6, C_7. Finally components C_6, C_7 may call C_8 .Other components are run in sequential manner. In order to verify the algorithm presented in this paper and to obtain analytical results, the number of calls is assumed as follows: Component C_1 calls C_2, C_3, C_4 only once during the execution time. Also Parallel components C_3, C_4 call components C_5, C_6, C_7 and finally C_6 calls C_8 only once. Component C_7 is a backup for C6 and has a similar behavior.

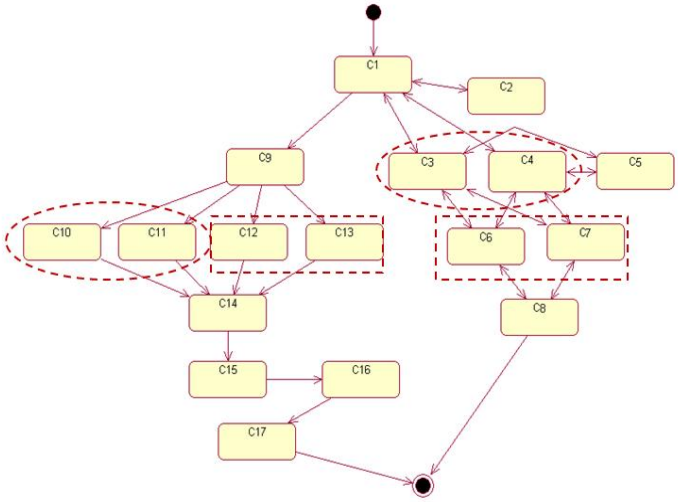

Figure11. State diagram of system architecture

Step 2: Identification of basic styles in heterogeneous architecture based on system design features: Considering the state diagram four basic styles can be identified as follow:

$$G = \{C_\alpha \mid \quad C_\alpha \in B \cup P \cup F \cup C \cup S, 1 \le \alpha \le 17\}$$

$$B = \{C_1, C_9, C_{14}, C_{15}, C_{16}, C_{17}\}$$

$$P = \{C_3, C_4, C_{10}, C_{11}\}$$

$$F = \{C_6, C_7, C_{12}, C_{13}\}$$

$$S = \{C_2, C_3, C_4, C_5, C_6, C_7, C_8\}$$

Step 3: Mapping the state diagram to Markov model: In this step mapping with regard to Markov model can be a one to one or many to one. For example in the figure 12 components c3 and c4 are operated in parallel way, so they are mapped to one state.

Step 4: Integrating Markov models in order to create an overall Markov model: with regard to the separate Markov models in previous step, overall model can be computed after integration of these models, as shown in figure 13.

Stage 5: Creating the separate sets for each style and enforcing limitations: Separate sets for different styles in the Morkov model can be computed as follows:

$$\delta_B = \{S_1, S_V\} \cup \{S_{10}, S_{11}, S_{12}, S_{13}\} = \{S_1, S_V, S_{10}, S_{11}, S_{12}, S_{13}\} \quad \delta_P = \{S_3\} \cup \{S_8\} = \{S_3, S_8\}$$

$$\delta_F = \{S_6\} \cup \{S_9\} = \{S_6, S_9\}$$

$$\delta_C = \{S_1\} \cup \{S_2\} \cup \{S_5\} = \{S_1, S_2, S_5\}$$

$$\delta_S = \{S_2\} \cup \{S_3\} \cup \{S_4\} \cup \{S_5\} \cup \{S_7\} = \{S_2, S_3, S_4, S_5, S_7\}$$

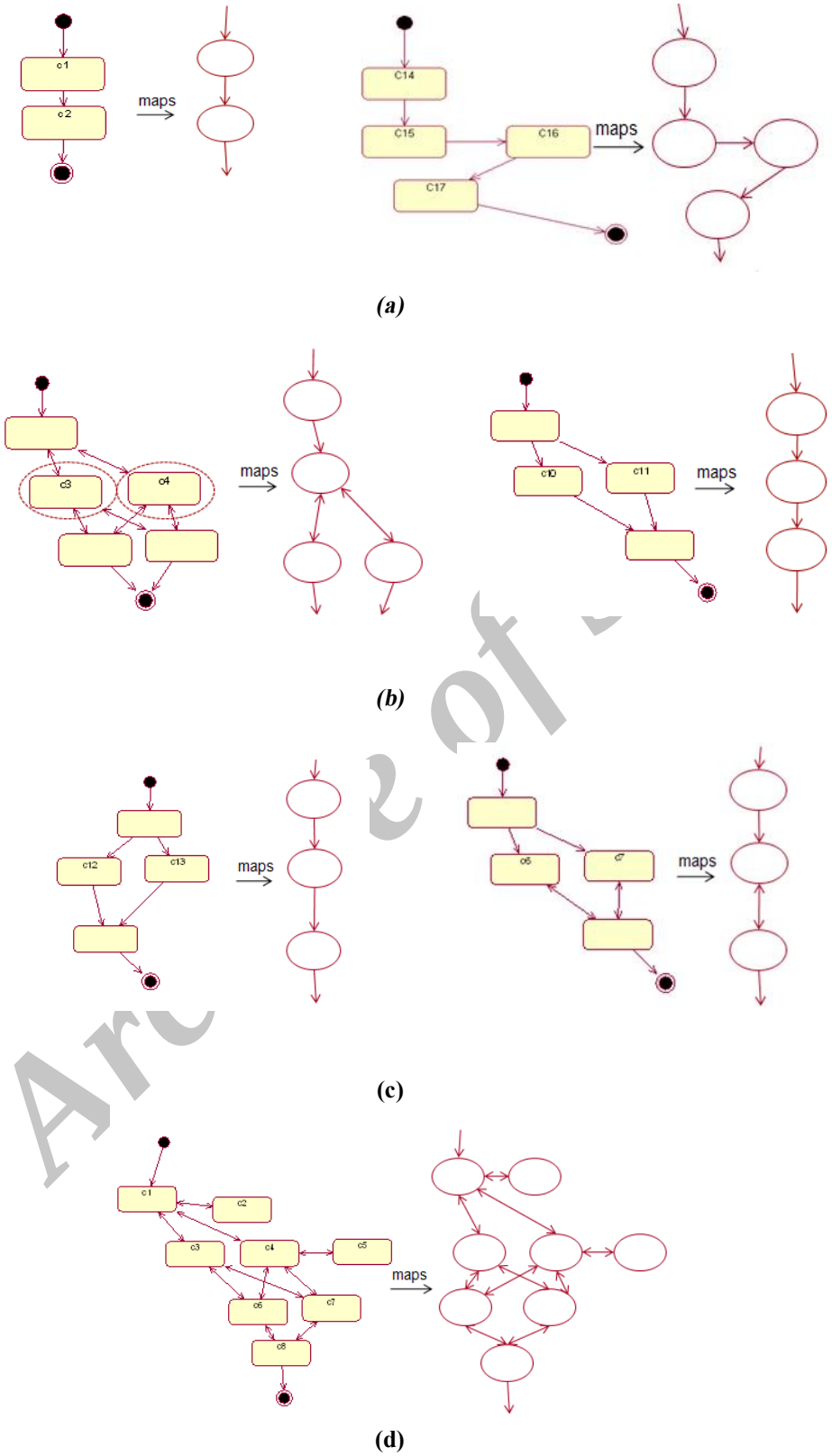

Figure12. Mapping state diagram to Markov model

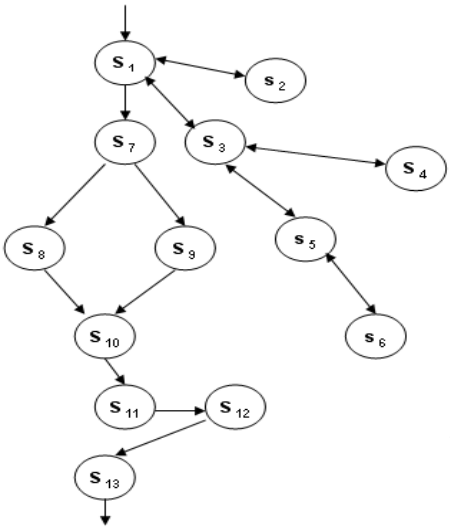

Figure13. Markov model of Heterogeneous software architecture

Step 6: Creating the transition probability matrix: In this step the transition probability matrix can be computed from equation 1:

$$\begin{bmatrix}
0 & 0/3 & 0/3 & 0 & 0 & 0 & 0/3 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0/3 & 0 & 0 & 0/3 & 0/3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0/5 & 0 & 0 & 0/5 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0/5 & 0/5 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}$$

Step 7: Calculating the visits number of each state in Markov model: with regard to equation 2 the number of state visits can be computed as follows:

$$V_j = q_j + \sum_{k=1}^{m-n} P_{kj} V_k V_1 = q_1 + \sum_{k=1}^{12} P_{k,1}.V_k = q_1 = 1$$

$$V_2 = \sum_{k=1}^{12} t(k,2) = t(1,2) = 1$$

$$V_3 = \sum_{k=1}^{12} t(k,3) = t(1,3) = 1$$

$$V_4 = \sum_{k=1}^{12} t(k,4) = t(3,4) = 1$$

$$V_5 = \sum_{k=1}^{12} t(k,5) = t(3,5) = 1$$

$$V_6 = \sum_{k=1}^{12} t(k,6) = t(5,6) = 1$$

$$V_7 = q_7 + \sum_{k=1}^{12} P_{k,7}.V_k = P_{1,7}.V_1 = 0/3$$

$$V_8 = q_8 + \sum_{k=1}^{12} P_{k,8}.V_k = P_{7,8}.V_7 = 0/15$$

$$V_9 = q_9 + \sum_{k=1}^{12} P_{k,9}.V_k = P_{7,9}.V_7 = 0/15$$

$$V_{10} = q_{10} + \sum_{k=1}^{12} P_{k,10}.V_k = P_{8,10}.V_8 + P_{9,10}.V_9 = 0/3$$

$$V_{11} = q_{11} + \sum_{k=1}^{12} P_{k,11}.V_k = P_{10,11} V_{10} = 0/3$$

$$V_{12} = q_{11} + \sum_{k=1}^{12} P_{k,12}.V_k = P_{11,12} V_{11} = 0/3$$

$$V_{13} = q_{13} + \sum_{k=1}^{12} P_{k,13}.V_k = P_{12,13} V_{12} = 1 \times 0/3 = 0/3$$

Step 8: Evaluating the efficiency of the model: In the following with respect of the results in step 8, service time is calculated for separate styles. Finally the response time of overall system can be computed by rolling up the following calculation for each style.

$$response.time.Batch = \sum_{\substack{i \in S.B \\ j \in B.t}} V_i.T_j = 0/03 + 0/06 + 0/03 + 0/003 + 0/06 = 0/183$$

$$response.time.parallel = \sum_{S_{PL} \in s.p} V_{S_{PL}}.[MAX(T_i)] = MAX[0/08, 0/14] + 0/15[MAX(0/03, 0/02)] = 0/144$$

$$response.time.fault = \sum_{S_{Ft} \in s.t} V_{S_{Ft}}.[1 - (\prod_{i \in Ft} (1 - P_i)).T_i] = 0/15[0/021] + [0/015] = 0/018$$

$$response.time.call\ and\ Return = \sum_{S_{Ca} \in S.C} V_{S_{Ca}}.[(T_{i \in C.t}) + \sum_{J \in S.S} V_j.T_j] = T_1 + V_2.T_2 = 0/01 + 0/02 = 0/03$$

5. Discussions

Since architectural styles have special effects on qualitative features, software architects use them as catalogs in various architectural designs with regard to the characteristics of the system. Therefore, the advantage of using architectural styles in software architectural design is obvious. Since most of the architectures designed for large and complicated systems, are combinations of several varied styles, if the effect of architectural styles on performance quality attribute is quantitatively measurable, the architect will be able to make his/her decisions with more facility and care with a view to the system's requirement as well as access to different styles.

6. Conclusion and future work

In this paper a new algorithm was presented for performance evaluation of heterogeneous architectural styles. The algorithm consists of modeling the software architecture as a Discrete Time Markov Chain (DTMC) and DTMC model then

analyzed to get performance feature. This paper focused on service time parameter for evaluating software architecture; other parameters such as throughput, latency, data transmission and bandwidth for evaluating heterogeneous software architectural styles could be discussed in future works. Instead of using the styles mentioned in this paper, one can use patterns in performance evaluation. Other formal models such as petri net, colored petri net could improve this research for future works.

Reference

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

[2] 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.

[3] 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.

[4] 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.

[5] V.abroshan,A.haroonabadi,S.J.mirabedini, "Evaluation of software architecture using fuzzy colored petrinets",management science letters 2013,pp.665-682.

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

[7] S. Emadi, G. Aghaee Ghazvini, "A New Algorithm for Performance Evaluation of Homogeneous Architectural Styles", Journal of Advances in Computer Research, Sari Branch, Islamic Azad University, Sari, Iran, Vol. 3, No. 2, May 2012, pp. 53-64.

[8] 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.

[9] 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.

[10] F. Brosch, H. Koziolek, B. Buhnova and R. Reussner, "Architecture-based Reliability Prediction with the Palladio component Model", IEEE Transactions On Software Engineering, 2011.

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

[12] N.Esfahani, S.Malek, "Utilizing architectural styles to enhance the adaptation support of middleware platforms", Journal of information and software technology, 2012, pp. 786-801.

[13] W. Wang, D. Pan and M. Chen, "Architecture-based software reliability modeling", Journal of System and Software, vol. 79, Jan. 2006, pp. 132-146.

[14] 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.

[15] V.S.Sharma,P.Jalote,K.S.Trivedi,"Evaluating performance Attributes of layered software architecture",Springer Journal,Component-Based Software engineering,Lecture Notes in Computer Science, 2005,Vol.3489, pp.66-81.

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

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