International Journal on New Computer Architectures and Their Applications (IJNCAA) 1(3): 744 – 765
The Society of Digital Information and Wireless Communications, 2011 (ISSN: 2220-9085)

# Derivation of Stochastic Reward net (SRN) from UML specification considering cost efficient deployment management of collaborative service components

Razib Hayat Khan, Poul E. Heegaard
Norwegian University of Science & Technology
7491, Trondheim, Norway
{rkhan, poul.heegaard}@item.ntnu.no

## ABSTRACT

Performance evaluation of distributed system is always an intricate undertaking where system behavior is distributed among several components those are physically distributed. Bearing this concept in mind, we delineate a performance modeling framework for a distributed system that proposes a transformation process from high level UML notation to SRN model and solves the model for relevant performance metrics. To capture the system dynamics through our proposed framework we outline a specification style that focuses on UML collaboration and activity as reusable specification building blocks, while deployment diagram identify the physical components of the system and the assignment of software artifacts to identified system components. Optimal deployment mapping of software artifacts on the available physical resources of the system is investigated by deriving the cost function. The way to deal with parallel thread processing of the network nodes by defining the upper bound is precisely mentioned to generate the SRN model. The proposed performance modeling framework provides transformation rules of UML elements into corresponding SRN representations and also the prediction result of a system such as throughput. The applicability of our proposed framework is demonstrated in the context of performance modeling of a distributed system.

## Keywords

UML, SRN, Performance attributes

## 1 Introduction

Distributed system poses one of the main streams of information and communication technology arena with immense complexity. Designing and implementation of such complex systems are always an intricate endeavor. Likewise performance evaluation is also a great concern of such complex system to evaluate whether the system meets the performance related system requirements. Hence modeling phase plays an important role in the whole design process of the system for qualitative and quantitative analysis. However in a distributed system, system behavior is normally distributed among several objects. The overall behavior of the system is composed of the partial behavior of the distributed objects of the system. So it is obvious to capture the behavior of the distributed objects for appropriate analysis to evaluate the performance related factors of the overall system. We therefore adopt UML collaboration and activity oriented approach as UML is the most widely used modeling language which models both the system requirements and qualitative behavior through different notations [2]. Collaboration and activity diagram are utilized to demonstrate the

overall system behavior by defining both the structure of the partial object behavior as well as the interaction between them as reusable specification building blocks and later this UML specification style is applied to generate the SRN model by our proposed performance modeling framework. UML collaboration and activity provides a tremendous modeling framework containing several interesting properties. Firstly collaborations and activity model the concept of service provided by the system very nicely. They define structure of partial object behaviors, the collaboration roles and enable a precise definition of the overall system behavior. They also delineate the way to compose the services by means of collaboration uses and role bindings [1].

The proposed modeling framework considers system execution architecture to realize the deployment of the service components. Abstract view of the system architecture is captured by the UML deployment diagram which defines the execution architecture of the system by identifying the system components and the assignment of software artifacts to those identified system components [2]. Considering the system architecture to generate the performance model resolves the bottleneck of system performance by finding a better allocation of service components to the physical nodes. This needs for an efficient approach to deploy the service components on the available hosts of distributed environment to achieve preferably high performance and low cost levels. The most basic example in this regard is to choose better deployment architectures by considering only the latency of the service. The easiest way to satisfy the latency requirements is to indentify and deploy the service components that require the highest volume of interaction onto the same resource or to choose resources that are connected by links with sufficiently high capacity [3].

It is indispensable to extend the UML model to incorporate the performance-related quality of service (QoS) information to allow modeling and evaluating the properties of a system like throughput, utilization, and mean response time. So the UML models are annotated according to the *UML profile for MARTE: Modeling & Analysis of Real-Time Embedded Systems* to include quantitative system parameters [4]. Thus it helps to maintain consistency between system design and implementation with respect to requirement specification.

Markov models, stochastic process algebras, stochastic petri net and stochastic reward net (SRN) are probably the best studied performance modeling techniques [5]. Among all of them, we will focus on the stochastic reward net (SRN) as the performance model generated by our proposed framework due to its increasingly popular formalism for describing and analyzing systems, its modeling generality, its ability to capture complex system behavior concisely, its ability to preserve the original architecture of the system, to allow marking dependency firing rates & reward rates defined at the net level, to facilitate any modification according to the feedback from performance evaluation and the existence of analysis tools.

Several approaches have been followed to generate the performance model from system design specification. Lopez-Grao *et al.* proposed a conversion method from annotated UML activity diagram to stochastic petrinet model [6]. Distefano

*et al*. proposed a possible solution to address software performance engineering that evolves through system specification using an augmented UML notation, creation of an intermediate performance context model, generation of an equivalent stochastic petri net model whose analytical solution provides the required performance measures [7]. D'Ambrogio proposed a framework for transforming source software models into target performance models by the use of meta-modeling techniques for defining the abstract syntax of models, the interrelationships between model elements and the model transformation rules [8]. However, most existing approaches do not highlight more on the issue that how to optimally conduct the system modeling and performance evaluation. The framework presented here is the first known approach that introduces a new specification style utilizing UML behavioral diagrams as reusable specification building block which is later used for generating performance model to produce performance prediction result at early stage of the system development process. Building blocks describe the local behavior of several components and the interaction between them. This provides the advantage of reusability of building blocks, since solution that requires the cooperation of several components may be reused within one self-contained, encapsulated building block. In addition the resulting deployment mapping provided by our framework has great impact with respect to QoS provided by the system. Our aim here is to deal with vector of QoS properties rather than restricting it in one dimension. Our presented deployment logic is surely able to handle any properties of the service, as long as we can provide a cost function for the specific property. The cost function defined here is flexible enough to keep pace with the changing size of search space of available host in the execution environment to ensure an efficient deployment of service components. Furthermore we aim to be able to aid the deployment of several different services at the same time using the same proposed framework. The novelty of our approach also reflected in showing the optimality of our solution with respect to both deployment logic and evaluation of performance metrics.

The objective of the paper is to provide an extensive performance modeling framework that provides a translation process to generate SRN performance model from system design specification captured by the UML behavioral diagram and later solves the model for relevant performance metrics to demonstrate performance prediction results at early stage of the system development life cycle. To incorporate the cost function to draw relation between service component and available physical resources permit us to identify an efficient deployment mapping in a fully distributed manner. The way to deal with parallel thread processing of the network node by defining the upper bound is precisely mentioned while generating the SRN model through the proposed framework. The work presented here is the extension of our previous work described in [9] [10] [14] where we presented our proposed framework with respect to the execution of single and multiple collaborative sessions and considered alternatives system architecture candidate to describe the system behavior and evaluate the performance factors.

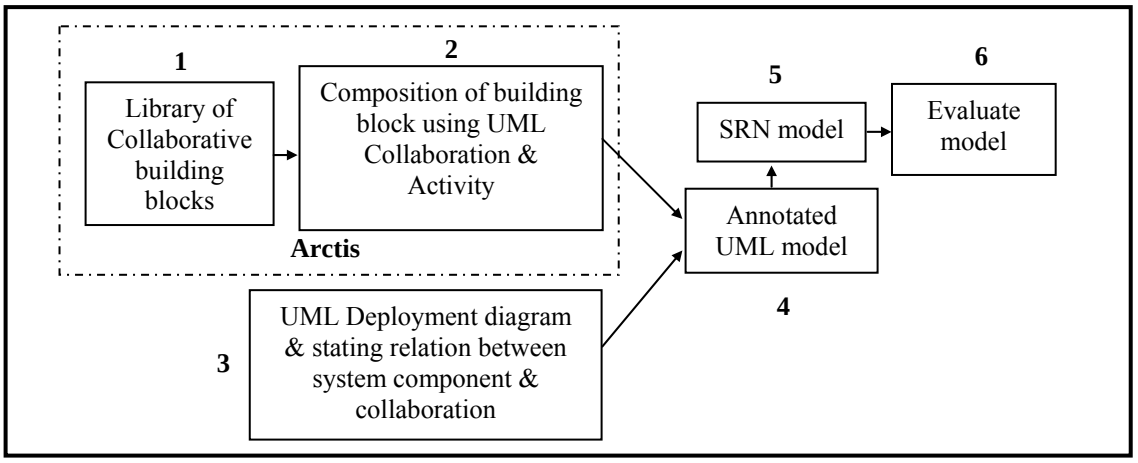


**Figure 1.** Proposed performance modeling framework

The paper is organized as follows: section 2 introduces our proposed performance modeling framework, section 3 demonstrates the application example to show the applicability of our modeling framework, section 4 delineates conclusion with future works.

## 2 Proposed Performance Modeling Framework

Our proposed performance modeling framework utilizes the tool suite Arctis which is integrated as plug-ins into the eclipse IDE [11]. The proposed framework is composed of 6 steps shown in figure 1 where steps 1 and 2 are the parts of Arctis tool suite.

Arctis focuses on the abstract, reusable service specifications that are composed form UML 2.2 collaborations and activities. It uses collaborative building blocks as reusable specification units to create comprehensive services through composition. To support the construction of building block consisting of collaborations and activities, Arctis offers special actions and wizards. In addition a number of inspections ensure the syntactic consistency of building blocks. A developer first consults a library to check if an already existing collaboration block or a collaboration of several blocks solves a certain task. Missing blocks can also be created from scratch and stored in the library for later reuse. The building blocks are expressed as UML models. The structural aspect, for example the service component and their multiplicity, is expressed by means of UML 2.2 collaborations. For the detailed internal behavior, UML 2.2 activities have been used. They express the local behavior of each of the service components as well as their necessary interactions in a compact and self-contained way using explicit control flows [11]. Moreover the building blocks are combined into more comprehensive service by composition. For this composition, Arctis uses UML 2.2 collaborations and activities as well. While collaborations provide a good overview of the structural aspect of the composition, i.e., which sub-services are reused and how their collaboration roles are bound, activities express the detailed coupling of their respective behaviors [11].

The steps are illustrated below:

**Step 1: Construction of collaborative building block:** The proposed framework utilizes collaboration as main specification units. The specifications for collaborations are given as coherent, self-contained reusable building blocks. The structure of the building block is described by UML 2.2 collaboration. The building block declares the participants (as collaboration roles) and connection between them.

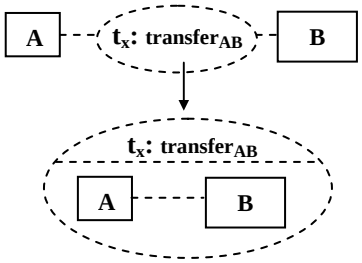


**Figure 2.** Structure of the building block using collaboration diagram

The internal behavior of building block is described by UML activity. It is declared as the classifier behavior of the collaboration and has one activity partition for each collaboration role in the structural description. For each collaboration use, the activity declares a corresponding call behavior action refereeing to the activities of the employed building blocks. For example, the general structure of the building block *t* is given in figure 2 where it only declares the participants A and B as collaboration roles and the connection between them is defined as collaboration use $t_x$ (x=1…$n_{AB}$ (number of collaborations between collaboration roles A & B)). The internal behavior of the same building block is shown in figure 3(b). The activity $transfer_{ij}$ (*where* $ij = AB$) describes the behavior of the corresponding collaboration. It has one activity partition for each collaboration role: A and B. Activities base their semantics on token flow [1]. The activity starts by placing a token when there is a response (indicated by the streaming pin *res*) to transfer by either participant A or B. After completion of the processing by the collaboration role A and B the token is transferred from the participant A to participant B and from participant B to Participant A which is represented by the call behavior action *forward.*

**Step 2: Composition of building block using UML collaboration & activity:** To generate the performance model, the structural information about how the collaborations are composed is not sufficient. It is necessary to specify the detailed behavior of how the different events of collaborations are composed so that the desired overall system behavior can be obtained. For the composition, UML collaborations and activities are used complementary to each other; UML collaborations focus on the role binding and structural aspect, while UML activities complement this by covering also the behavioral aspect for composition. For this purpose, call behavior actions are used. Each sub-service is represented by call behavior action referring the respective activity of building blocks. Each call behavior action represents an instance of a building block. For each activity parameter node of the referred activity, a call behavior action declares a corresponding pin. Pins have the same symbol as activity parameter nodes to represent them on the frame of a call behavior action. Arbitrary logic between pins may be used to synchronize the building block events and transfer data between them.

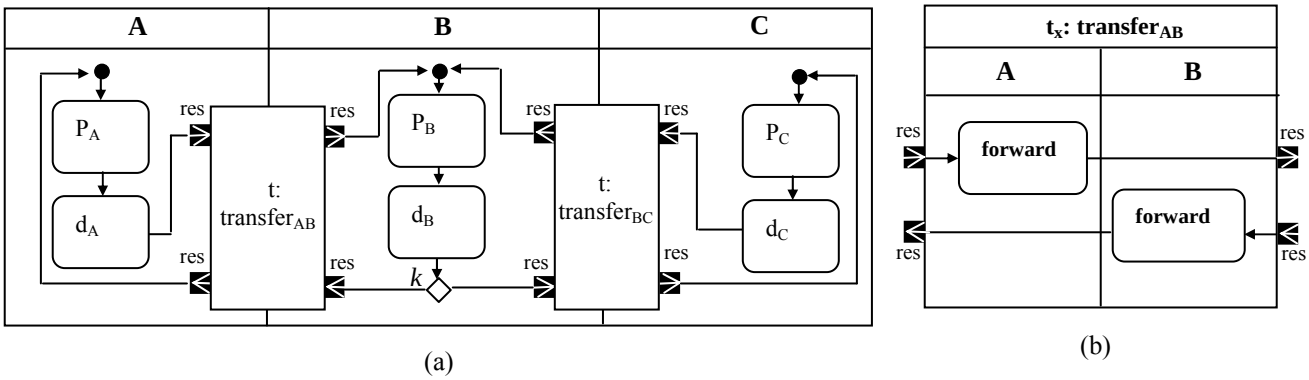


**Figure 3.** System activity to couple the collaboration

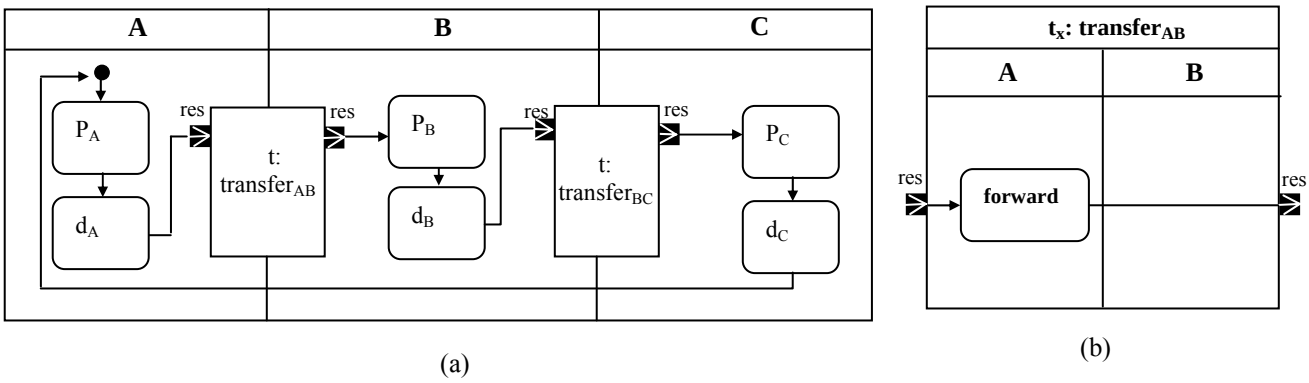


**Figure 4.** System activity to couple the collaboration when there is an order in which collaboration roles are selected for completing the processing.

By connecting the individual input and output pins of the call behavior actions, the events occurring in different collaborations can be coupled with each other. Semantics of the different kinds of pins are given in more detailed in [1].

To delineate the overall system behavior we will consider two sorts of activity diagram where activities base their semantics on token flow. In first case, each collaboration role contains one token and the processing realized by the collaboration role is independent of each other and in second case one token will be passed through the each collaboration role to realize the processing done by the collaboration role which symbolizes the dependency among the execution of collaborations roles' activity as there is an order in which collaboration roles are selected for completing the execution of their activity. For example the detailed behavior and composition of the collaboration for the first case is given in figure 3(a).The initial node (●) indicates the starting of the activity. The activity is started at the same time from each participant. After being activated, each participant starts its processing of the request which is mentioned by call behavior action $P_i$ ($Processing_i$, where i = A, B & C). Completions of the processing by the participants are mentioned by the call behavior action $d_i$ ($Processing\_done_i$, i = A, B & C). After completion of the processing, the responses are delivered to the corresponding participants indicated by the streaming pin *res*. When the

execution of the task by the participant B completes the result is passed through a decision node *k* and only one flow is activated at the certain time instance. The response of the collaboration role A and C are forwarded to B and the response of collaboration role B is forwarded to either A or C which is mentioned by collaboration *t: transfer*$_{ij}$ (where ij = AB or BC). The completion of the activity of each participant is shown by the ending node (◉). In the above way the detailed behavior and composition of the collaboration as well as the internal behavior of the collaboration for the second case can be illustrated which are portrayed in figure 4 (a) and 4 (b).

**Step 3: Designing UML deployment diagram & stating relation between system components & collaborations:** Our deployment logic is launched with the service model enriched with the requirements specifying the search criteria and with a resource profile of the hosting environment specifying the search space. In our view, however, the logic we develop is capable of catering for any other types of non-functional requirements too, as long as a suitable cost function can be provided for the specific QoS dimension at hand. In this paper, costs in the model are constant, independent of the utilization of underlying hardware [3]. Furthermore, we benefit from using collaborations as design elements as they incorporate local behavior of all participants and all interactions between them. That is, a single cost value can describe communication between component instances, without having to care about the number of messages sent, individual message sizes, etc.

We model the system as collection of N interconnected nodes shown in figure 5. Our objective is to find a deployment mapping for this execution environment for a set of service components C available for deployment that comprises service. Deployment mapping can be defined as M: $C \rightarrow N$ between a numbers of service components instances c, onto nodes n. A components $c_i \in C$ can be a client process or a service process, while a node, $n \in N$ is a physical resource. Generally, nodes can have different responsibilities, such as providing

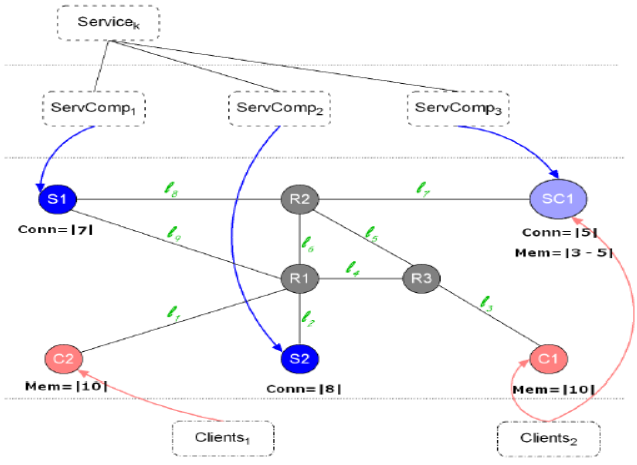


**Figure 5.** Components mapping example

services (*S1*), relaying traffic (*R1*), accommodating clients (*C1*), or a mixture of these (*SC1*). Components can communicate via a set of collaborations. We consider four types of requirements in the deployment problem. Components have execution costs, collaborations have communication costs and costs for running of background process and some of the components can be restricted in the deployment mapping to specific nodes which are called bound components. Furthermore, we consider identical nodes that are interconnected in a full-mesh and are capable of hosting components with unlimited processing demand. We observe the processing load that nodes impose while host the components and also the target balancing of load between the nodes available in the network.

By balancing the load the deviation from the global average per node execution cost will be minimized. Communication costs are considered if collaboration between two components happens remotely, i.e. it happens between two nodes [3]. In other words, if two components are placed onto the same node the communication cost between them will not be considered. The cost for executing the background process for conducting the communication between the collaboration roles is always considerable no matter whether the collaboration roles deploy on the same or different nodes. Using the above specified input, the deployment logic provides an optimal deployment architecture taking into account the QoS requirements for the components providing the specified services. We then define the objective of the deployment logic as obtaining an efficient (low-cost, if possible optimum) mapping of component onto the nodes that satisfies the requirements in reasonable time. The deployment logic providing optimal deployment architecture is guided by the cost function F (M). The evaluation of cost function F(M) is mainly influenced by our way of service definition. Service is defined in our approach as a collaboration of total E components labeled as $c_i$ (where i = 1…. E) to be deployed and total K collaboration between them labeled as $k_j$, (where j = 1 … K). The execution cost of each service component can be labeled as $fc_i$; the communication cost between the service components is labeled as $fk_j$ and the cost for executing the background process for conducting the communication between the service components is labeled as $f_{Bj}$. Accordingly we only observe the total load ($\widehat{l_n}$ , $n$ = 1…N) of a given deployment mapping at each node. We will strive for an optimal solution of equally distributed load among the processing nodes and the lowest cost possible, while taking into account the execution cost $fc_i$, i = 1….E, communication cost $fk_j$, j = 1….K and cost for executing the background process $f_{Bj}$, j = 1….k. $fc_i$, $fk_j$ and $f_{Bj}$ are derived from the service specification, thus the offered execution load can be calculated as $\sum_{i=1}^{E} fc_i$. This way, the logic can be aware of the target load [6]:

$$T = \frac{\sum_{i=1}^{|E|} fc_i}{|N|} \quad (1)$$

To cater for the communication cost $fk_j$, of the collaboration $k_j$ in the service, the function $q_0$ (M, c) is defined first [21]:

$$q_0 (M, c) = \{n \in N \mid \exists (c \rightarrow n) \in M\}$$

This means that q0 (M, c) returns the node n that host component in the list mapping M. Let collaboration $k_j = (c_1, c_2)$. The communication cost of $k_j$ is 0 if components $c_1$ and $c_2$ are collocated, i.e. $q_0 (M, c_1) = q_0 (M, c_2)$, and the cost is $fk_j$ if components are otherwise (i.e. the collaboration is remote). Using an indicator function I(x), which is 1 if x is true and 0 otherwise, this expressed as $I (q_0 (M, c_1) \neq q0 (M, c_2)) = 1$, if the collaboration is remote and 0 otherwise. To determine which collaboration $k_j$ is remote, the set of mapping M is used. Given the indicator function, the overall communication cost of service, $F_k$ (M), is the sum [21]

$$F_k (M) = \sum_{j=1}^{|k|} I (q_0 (M, K_{j,1}) \neq q0 (M, K_{j,2})). fk_j$$

Given a mapping $M = \{m_n\}$ (where $m_n$ is the set of components at node $n$ & $n \in N$) the total load can be obtained as $\widehat{l}_n = \sum_{c_i \in m_n} fc_i$. Furthermore the overall cost function F (M) becomes (where $I_j = 1$, if $k_j$ external or 0 if $k_j$ internal to a node):

$$F (M) = \sum_{n=1}^{|N|} | \widehat{l}_n - T | + F_k (M) + \sum_{j=1}^{K} f_{Bj} \quad (2)$$

**Step 4: Annotating the UML model:** Performance information is incorporated into the UML activity diagram and deployment diagram according to *UML profile for MARTE: Modeling & Analysis of Real-Time Embedded Systems* [4] for evaluating system performance by performance model solver.

**Step 5: Deriving the SRN model:** Since an SRN is based on a Petri net; the introduction of Petri net is described in brief [5]. A Petri net is represented by a bipartite directed graph with two types of nodes: places and transitions. Each place may contain zero or more tokens in a marking. Marking represents the state of the Petri net at a particular instant. A transition is enabled if all of its input places have at least as many tokens as required by the multiplicities of the input arcs. A transition may fire when it is enabled, and according to the multiplicities of the arcs, tokens in each input place are removed and new tokens are deposited in each output place. In a stochastic Petri net (SPN), each transition has firing time that represents the time to fire the transition after it is enabled.

Generalized stochastic Petri net (GSPN) extends SPN by introducing the immediate transition which has zero firing time. An immediate transition is represented by a thin black bar [5]. A marking in a GSPN is called vanishing if at least one immediate transition is enabled in the marking; otherwise the marking is called tangible. GSPN also introduces inhibitor arcs that disable the transition unless the number of tokens in input place is as many as the multiplicity of the inhibitor arc. An inhibitor arc is represented by a line terminated with a small hollow circle.

SRN is based on the Generalized Stochastic Petri net (GSPN) and extends them further by introducing prominent extensions such as guard functions, reward function and marking dependent firing rates [5]. A guard function is assigned to a transition. It specifies the condition to enable or disable the transition and can use the entire state of the net rather than just the number of tokens in places. Reward function defines the reward rate for each tangible marking of Petri Net based on which

various quantitative measures can be done in the Net level. Marking dependent firing rate allows using the number of token in a chosen place multiplying the basic rate of the transition.

By considering the internal behavior of the reusable building blocks (step1), composition of different events of the building blocks (step2), deployment mapping between system component and collaboration (step3) and annotated UML structure (step4), probable states and transition rate for triggering the change between states will be found based on which our SRN performance model will be generated. To generate the SRN model of the system, first we generate the SRN model of the individual system components and later compose them together to generate the system level SRN model. The rules are based on decomposition of UML collaboration, activity and deployment diagram into basic elements of SRN model like states as places, timed transition and immediate transition. In addition the rules are based on the rendezvous synchronization that means when communication between two processes of two interconnected nodes occur it follows the rendezvous synchronization [12]. Rendezvous provides synchronization between two threads while they communicate. In rendezvous synchronization, a synchronization and communication point called an entry is constructed as a function call. One process defines its entry and makes it public. Any process with knowledge of this entry can call it as an ordinary function call. The process that defines the entry accepts the call, executes it and returns the results to the caller. The issuer of the entry call establishes a rendezvous with the process that defined the entry [12].

SRN model of the collaboration role of a reusable building block is mentioned by the 6-tuple $\{\Phi, T, A, K, N, m_0\}$ in the following way [5]:

$\Phi$ = Finite set of the places (drawn as circles), derived from the call behavior action of the collaboration role
T = Finite set of the transition (drawn as bars), derived from the annotated UML activity diagram that denotes system's behavior
$A \subseteq \{\Phi \times T\} \cup \{T \times \Phi\}$ is a set of arcs connecting $\Phi$ and T,
K: T → {Timed (time>0, drawn as solid bar), Immediate (time = 0, drawn as thin bar)} specifies the type of the each transition, derived from the annotated UML activity diagram that denotes system's behavior
N: A→ {1, 2, 3…} is the multiplicity associated with the arcs in A,
m: $\Phi$ → {0, 1, 2...} is the marking that denotes the number of tokens for each place in $\Phi$. The initial marking is denoted as $m_0$.

The rules are following:

**Rule 1:** The SRN model of the collaboration role of a reusable building block is represented by the 6-tuple in the following way:

$\Phi_i = \{P_i, d_i\}$
T = {do, exit}
$A = \{\{(P_i \times do) \cup (do \times d_i)\}, \{(d_i \times exit) \cup (exit \times P_i)\}\}$
K = (do → Timed, exit → Immediate)
$N = \{(P_i \times do) \rightarrow 1, (do \times d_i) \rightarrow 1, (d_i \times exit) \rightarrow 1, (exit \times P_i) \rightarrow 1\}$
$m_o = \{(P_i \rightarrow 1\}, (d_i \rightarrow 0)\}$

The figure 6(a) highlights the SRN model of the collaboration role *A* where *A* has its own token to start the execution of the SRN model and the figure 6 (b) highlights the SRN model of the collaboration role *A* where the starting of the execution of the SRN model of *A* depends on the receiving of token from other element.

**Rule 2:** The SRN model of a collaboration where collaboration connects only two collaboration roles are represented by the 6-tuple in the following way (In this case, each collaboration role has its own token and the processing realized by the collaboration role is independent of each other):

$\Phi = \{\Phi_i, \Phi_j\} = \{P_i, d_i, P_j, d_j\}$
$T = \{do_i, do_j, t_{ij}\}$
$A = \{\{(P_i \times do_i) \cup (do_i \times d_i)\}, \{(d_i \times t_{ij}) \cup (t_{ij} \times P_j)\}, \{(P_j \times do_j) \cup (do_j \times d_j)\} \{(d_j \times t_{ij}) \cup (t_{ij} \times P_i)\}\}$
$K = (do_i \rightarrow \text{Timed}, do_j \rightarrow \text{Timed}, t_{ij} \rightarrow \text{Timed} \mid \text{Immediate})$
$N = \{(P_i \times do_i) \rightarrow 1, (do_i \times d_i) \rightarrow 1, (d_i \times t_{ij}) \rightarrow 1, (t_{ij} \times P_i) \rightarrow 1, \{\{(P_j \times do_j) \rightarrow 1, (do_j \times d_j) \rightarrow 1, (d_j \times t_{ij}) \rightarrow 1, (t_{ij} \times P_j) \rightarrow 1\}$
$m_o = \{(P_i \rightarrow 1, d_i \rightarrow 0, P_j \rightarrow 1, d_j \rightarrow 0\}$

$t_{ij}$ is a timed transition if the two collaboration roles deploy on the different physical node (communication time > 0) or immediate transition if the two collaboration roles deploy on the same physical node (communication time = 0). SRN model of the collaboration is graphically represented in figure 7.

**Rule 3:** The SRN model of a collaboration where collaboration connects only two collaboration roles is represented by the 6-tuple in the following way (In this case one token will be passed through the each collaboration role to realize the processing done by the collaboration role which symbolizes the dependency among the execution of collaborations roles' activity):

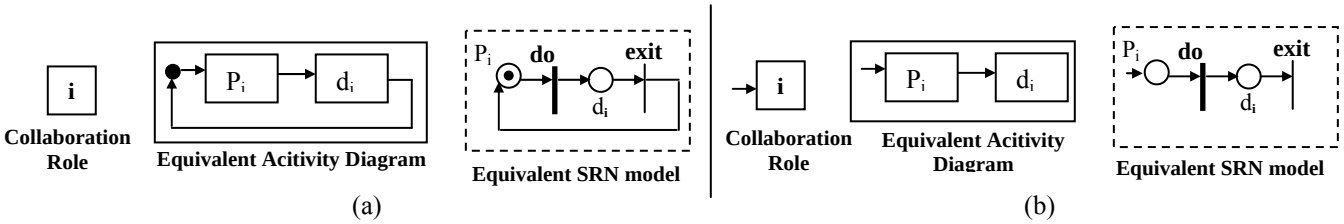


**Figure 6.** Graphical representation of Rule 1

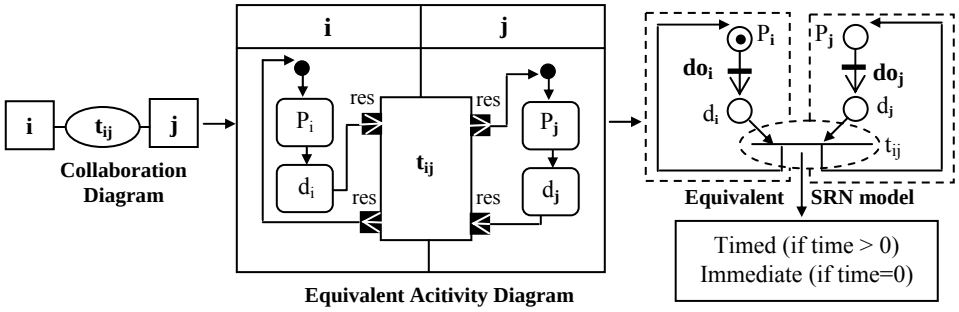


**Figure 7.** Graphical representation of Rule 2

$\Phi = \{\Phi_i, \Phi_j\} = \{P_i, d_i, P_j, d_j\}$
$T = \{do_i, do_j, t_{ij}\}$
$A = \{\{(P_i \times do_i) \cup (do_i \times d_i)\}, \{(d_i \times t_{ij}) \cup ((t_{ij} \times P_i), (t_{ij} \times P_j))\}, \{(P_j \times do_j) \cup (do_j \times d_j)\}\ \{(d_j \times exit) \cup (\emptyset)\}\}$
$K = (do_i \rightarrow Timed, do_j \rightarrow Timed, t_{ij} \rightarrow Timed \mid Immediate)$
$N = \{(P_i \times do_i) \rightarrow 1, (do_i \times d_i) \rightarrow 1, (d_i \times t_{ij}) \rightarrow 1, (t_{ij} \times P_i) \rightarrow 1, (t_{ij} \times P_j) \rightarrow 1, (P_j \times do_j) \rightarrow 1, (do_j \times d_j) \rightarrow 1, (d_j \times exit) \rightarrow 1\}$
$m_o = \{(P_i \rightarrow 1, d_i \rightarrow 0, P_j \rightarrow 1, d_j \rightarrow 0\}$

$t_{ij}$ is an immediate transition if the two collaboration roles deploy on the same physical node (communication time = 0) or timed transition if the two collaboration roles deploy on the different physical nodes (communication time > 0). SRN model of collaboration is represented graphically in figure 8.

**Rule 4:** When the collaboration role of a reusable building block deploys onto a physical node the equivalent SRN model is represented by 6-tuple in following way:

$\Phi_i = \{P_i, d_i, P_\Omega\}$
$T = \{do, exit\}$
$A = \{\{(P_i \times do) \cup (do \times d_i)\}, \{(P_\Omega \times do) \cup (do \times P_\Omega)\}, \{(d_i \times exit) \cup (exit \times P_i)\}\}$
$K = (do \rightarrow Timed, exit \rightarrow Immediate)$
$N = \{(P_i \times do) \rightarrow 1, (do \times d_i) \rightarrow 1, (P_\Omega \times do) \rightarrow 1, (do \times P_\Omega) \rightarrow 1 (d_i \times exit) \rightarrow 1, (exit \times P_i) \rightarrow 1\}$
$m_o = \{(P_i \rightarrow 1\}, (d_i \rightarrow 0), (P_\Omega \rightarrow q)\}$

Here place $P_\Omega$ contains **q** (where **q** = 1, 2, 3…..) tokens which define the upper bound of the execution of the threads in parallel by the physical node Ω and the timed transition *do* will fire only when there is a token available in both the place Pi and $P_\Omega$. The place $P_\Omega$ will again get back it's token after firing of the timed transition *do* indicating that the node is ready to execute incoming threads. SRN model of the collaboration role is graphically represented in the figure 9.

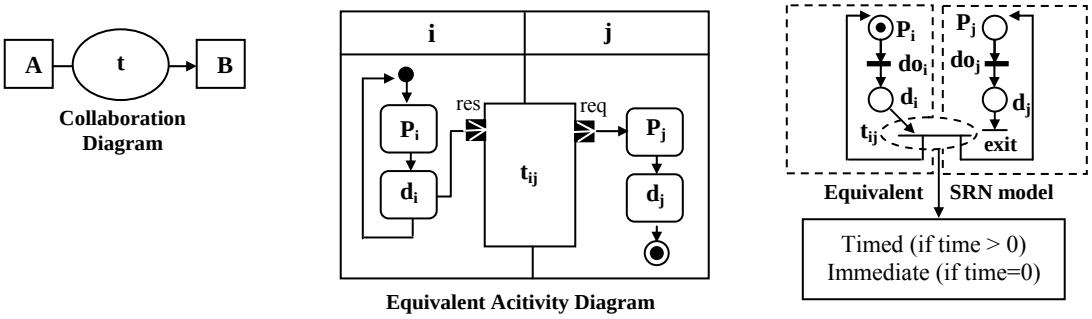


**Figure 8.** Graphical representation of Rule 3

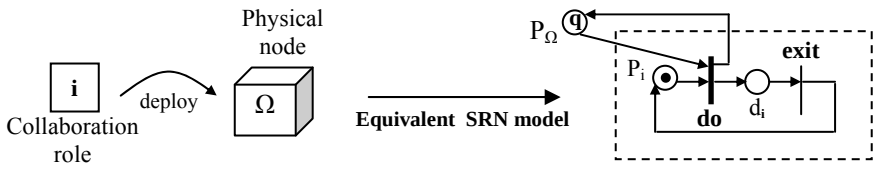


**Figure 9.** Graphical representation of Rule 4

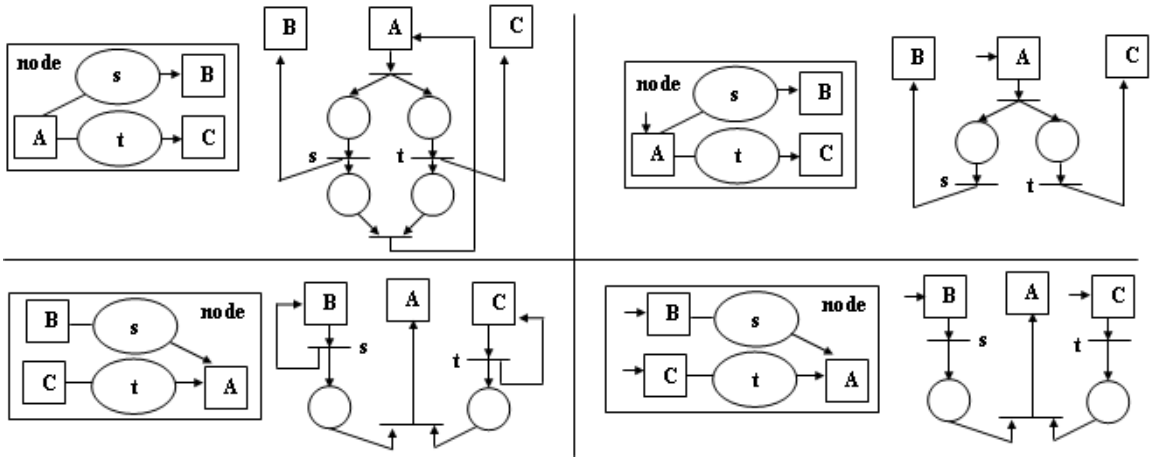


**Figure 10.** Graphical representation of Rule 5

**Rule 5:** For a composite structure, if a collaboration role *A* connects with *n* collaboration roles by *n* collaborations like a star graph (where n=2, 3, 4, …..) where each collaboration connects only two collaboration roles, then only one instance of collaboration role *A* exists during the it's basic state transition and the single instance of collaboration role *A* connects with all other collaboration roles by immediate or timed transitions based on their deployment on the same or different physical components to generate the SRN model. This rule can be demonstrated through 6-tuple in the above way. The graphical representations of the SRN model for composite structures are shown in the figure 10.

**Step 6: Evaluate the model:** We focus on measuring the throughput of the system from the developed SRN model. Before deriving formula for throughput estimation we consider several assumptions. Firstly if more than one service component deploy on a network node the processing power of the network node will be utilized among the multiple threads to complete the parallel processing of that node. There must be an upper bound of the execution of parallel threads by a network node. Secondly when communication between two processes of two interconnected nodes occur it follows the rendezvous synchronization. Moreover all the communications among the interconnected nodes occur in parallel. Finally the communications between interconnected nodes will be started following the completion of all the processing inside each physical node. By considering the all the assumption we define the throughput as function of total expected number of jobs, E (N) and cost of the network, C_Net. The value of E (N) is calculated by solving the SRN model using SHARPE [15]. The value of C_Net is evaluated by considering a subnet which is performance limiting factor of the whole network i.e., which posses maximum cost with respect to its own execution cost, communication cost with other subnet and cost for running background processes. Assume cost of the network, C_Net is defined as follows (where $fc_m$ = execution cost of the $m^{th}$ component of $subnet_i$; $c\_subnet_i$ = cost of the $i^{th}$ subnet where i = 1…n that comprises the whole network and $I_j = 0$ in this case as $k_j$ internal to a node):

$$c\_subnet_i = \max \{ fc_m + I_j f k_j + f B_j \};$$

$$= \max \{ fc_m + f B_j \}; \quad (3)$$

Now we evaluate the cost between each pair of subnet ($sbunet_i$ & $subnet_j$ ; where $i \neq j$) with respect to the subnet's own processing cost, cost for running background process and the cost associated with the communication with other subnet in the network. Cost between $subnet_i$ and $subnet_j$, $C\_subnet_{i,j}$ is defined as (where $fk_{i,j}$ = communication cost between $subnet_i$ & $subnet_j$ and $I_{i,j} = 1$ as $k_{i,j}$ external to a node):

$$c\_subnet_{i,j} = \max \{\max \{c\_subnet_i, c\_subnet_j\} + I_{i,j} f k_{i,j} + f B_{i,j} \}; \quad (4)$$

$$C\_Net = \max \{c\_subnet_{i,j}\}; \quad (5)$$

$$\therefore \text{Throughput} = \frac{\mathbf{E(N)}}{\mathbf{C\_Net}} \quad (6)$$

Equation 6 for conducting the throughout calculation is considered when each collaboration role has its own token and the processing realized by the collaboration role is independent of each other. The below equation 7 is considered for throughput calculation when there is an order in which collaboration roles are selected for completing the execution.

$$\therefore \text{Throughput} = \frac{\mathbf{E(N)}}{\mathbf{C\_Net'}} \quad (7)$$

Value of C_Net′ will be derived from equation (8).

$$C\_Net' = \sum_{n=1}^{|N|} \widehat{l}_n + F_k(M) + \sum_{j=1}^{K} f_{Bj} \quad (8)$$

## 3 Application Example

As a representative example, we consider the scenario originally from Efe dealing with heuristically clustering of modules and assignment of clusters to nodes [13]. This scenario is sufficiently complex to show the applicability of our proposed framework. The problem is defined in our approach as a service of collaboration of $E = 10$ components or collaboration role (labeled $C_1 \ldots C_{10}$) to be deployed and $K = 14$ collaborations between them depicted in figure 11. We consider four types of requirements in this specification. Besides the execution cost, communication costs and cost for running background process, we have a restriction on components $C_2$, $C_7$, $C_9$ regarding their location. They must be bound to nodes $\boldsymbol{n_2}$, $\boldsymbol{n_1}$, $\boldsymbol{n_3}$, respectively.

Moreover collaboration and components in the example scenario are shown in figure 12 as an order in which components are selected for completing the execution of their activity.

The internal behavior of the collaboration $K$ of our example scenario is realized by the call behavior action through UML activity like structure already mentioned in figure 3(b). The composition of the collaboration role $C$ is realized through UML activity diagram shown in figure 13. The initial node (●) indicates the starting of the activity. The activity is started at the same time from the entire participants $C_1$ to $C_{10}$. After being activated, each participant starts its processing of request which is mentioned by call behavior action $P_i$ (Processing of the $i^{th}$ service component). Completions of the processing by the participants are mentioned by the call behavior action $d_i$ (Processing done of the $i^{th}$ service component).

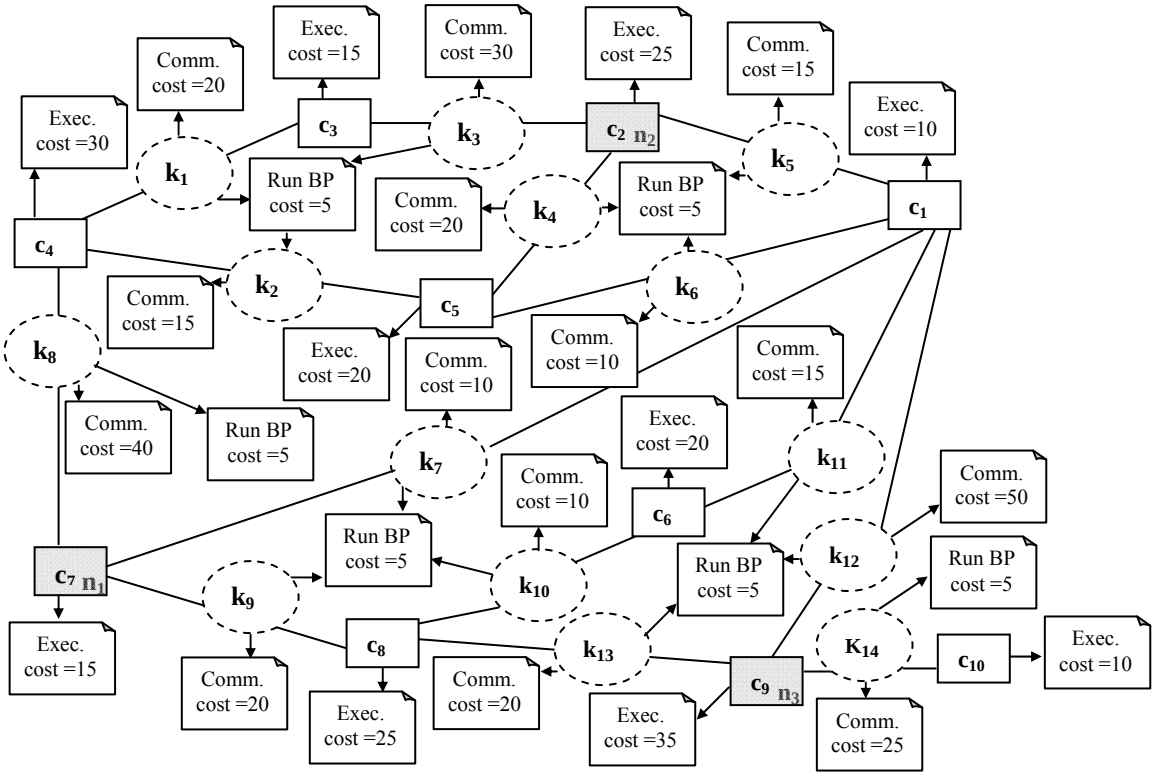


**Figure 11.** Collaborations and components in the example scenario

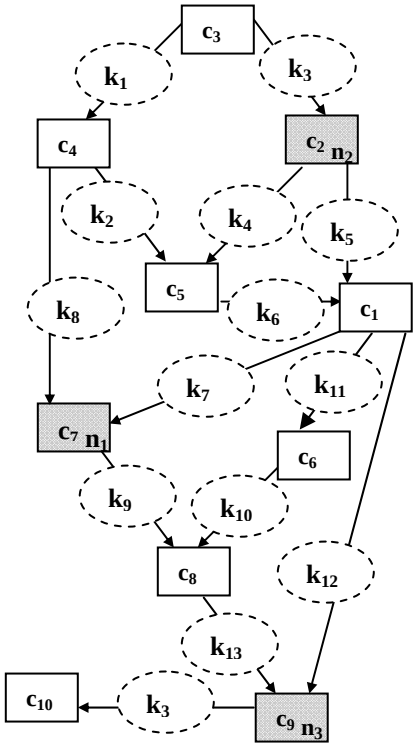


**Figure 12.** Collaborations and components in the example scenario when there is an order in which components are selected for completing the processing

After completion of the processing, the responses are delivered to the corresponding participants indicated by the streaming pin *res*. When any participant is associated with more than one participant through collaborations the result of the processing of that participant is passed through a decision node and only one flow is activated at the certain time instance. For example after completion of the processing of participant $C_2$ the response will be passed through the decision node $X_2$ and only one flow (flow towards $C_1$ or $C_3$ or $C_5$) will be activated. The completion of the processing of the each participant is shown by ending node (◉). In the same way the composition of the collaboration role $C$ is also realized through UML activity diagram (figure 14) where there is an order in which collaboration roles are selected for completing the execution of their activity.

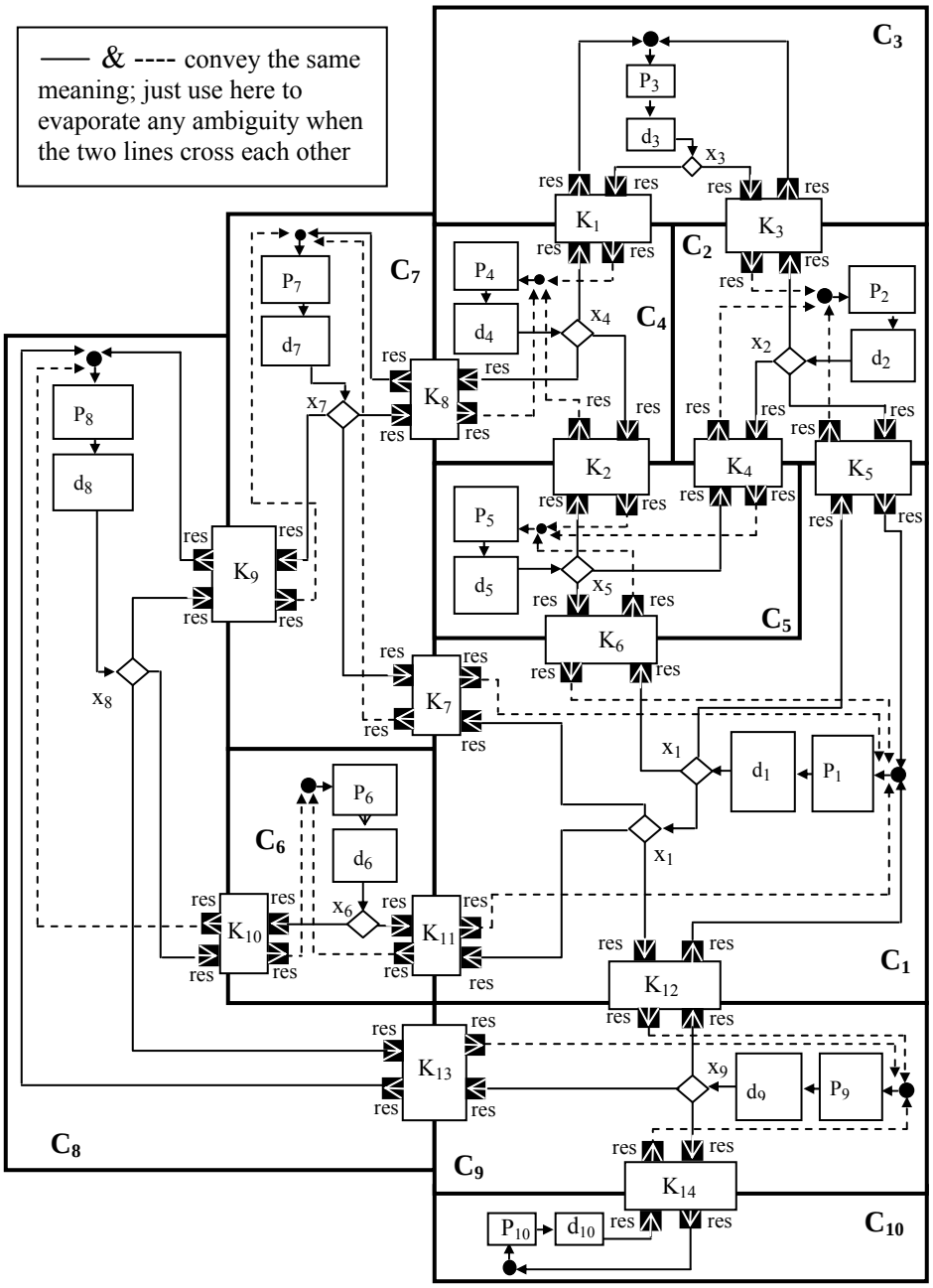


**Figure 13.** Detail behavior of the event of the collaboration using activity for our example scenario

In this example, the target environment consists only of $N = 3$ identical, interconnected nodes with a single provided property, namely processing power and with infinite communication capacities depicted in figure 15(a). The optimal deployment mapping can be observed in table 1. The lowest possible deployment cost, according to (2) is 17 + (100 + 70) = 187.

To annotate the UML diagram in figure 13, 14 & 15(a) we use the stereotype *saStep computingResource, scheduler* and the tag value *execTime, deadline* and *schedPolicy* [4].

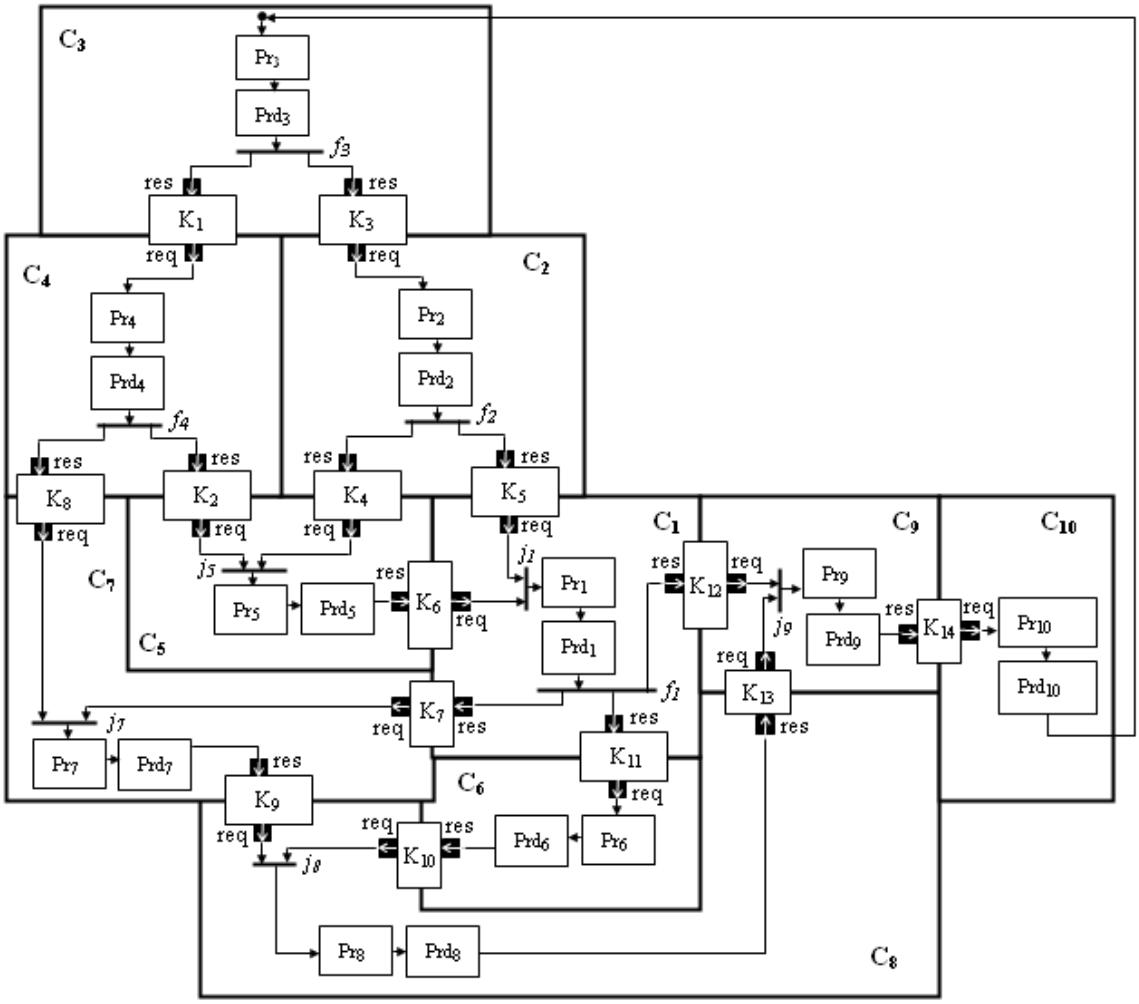


**Figure 14.** Detail behavior of the event of the collaboration using activity for our example scenario where there is an order in which collaboration roles are selected for completing the processing

*saStep* is a kind of step that begins and ends when decisions about the allocation of system resources are made. The duration of the execution time is mentioned by the tag value *execTime* which is the average time in our case. *deadline* defines the maximum time bound on the completion of the particular execution segment that must be met. A *ComputingResource* represents either virtual or physical processing devices capable of storing and executing program code. Hence its fundamental service is to compute. A *Scheduler* is defined as a kind of ResourceBroker that brings access to its brokered ProcessingResource or resources following a certain scheduling policy tagged by *schedPolicy*. Collaboration $K_i$ is associated with two instances of *deadline* (figure 15(b)) as collaborations in example scenario are associated with two kinds of cost: communication cost & cost for running background process.

By considering the above deployment mapping and the transformation rule the analogous SRN model of our example scenario is depicted in figure 16 where

each collaboration role has its own token and the processing realized by the collaboration role is independent of each other. The states of the SRN model are derived from the call behavior action of the corresponding collaboration role and collaboration among them. While generating the SRN model of the system if more than one service component deploy on a network node the processing power of the network node will be utilized among the multiple threads to complete the parallel processing of that node. This can be achieved through marking dependency firing rate defined as the following way in SRN model:

$$\lambda_i \; / \; \sum_{i=1}^{n} (\#(P_i)) \qquad (8)$$

Where $\lambda_i$ = processing rate of the $i^{th}$ service component deploys in a network node and i=1…n defines the number of service components deploy on a network node. (# ($P_i$)) returns the number of tokens in the place $P_i$.

According to the transformation rules 1, each collaboration role is defined by the two states $p_i$ and $d_i$ and the passing of token from state $p_i$ to $d_i$ is realized by the timed transition $t_i$ which is derived from the annotated UML model. Initially there will be a token from place $p_1$ to $p_{10}$. For generating the SRN model (figure 16) firstly we will consider the collaboration roles deploy on the processor node $\boldsymbol{n_1}$ which are $C_4$, $C_7$ & $C_8$. Here components $C_7$ are connected with $C_4$ and $C_8$. The communication cost between the components is zero but there is still some cost for execution of the background process. So according to rule 2, after the completion of the state transition from $p_7$ to $d_7$ (states of component $C_7$), from $p_4$ to $d_4$ (states of component $C_4$) and from $p_8$ to $d_8$ (states of component $C_8$) the states $d_7$, $d_4$ and $d_7$, $d_8$ are connected by the timed transition $k_8$ and $k_9$ to generate the SRN model. Collaboration roles $C_2$, $C_3$ & $C_5$ deploy on the processor node $\boldsymbol{n_2}$. Likewise after the completion of the state transition from $p_2$ to $d_2$ (states of component $C_2$), from $p_3$ to $d_3$ (states of component $C_3$) and from $p_5$ to $d_5$ (states of component $C_5$) the states $d_2$, $d_3$ and $d_2$, $d_5$ are connected by the timed transition $k_3$ and $k_4$ to generate the SRN model according to rule 2. Collaboration roles $C_6$, $C_1$, $C_9$ & $C_{10}$ deploy on the processor node $\boldsymbol{n_3}$. In the same way after the completion of the state transition from $p_1$ to $d_1$ (states of component $C_1$), from $p_6$ to $d_6$ (states of component $C_6$), $p_9$ to $d_9$ (states of component $C_9$) and from $p_{10}$ to $d_{10}$ (states of component $C_{10}$) the states $d_1$, $d_6$; $d_1$, $d_9$ and $d_9$, $d_{10}$ are connected by the timed transition $k_{11}$, $k_{12}$ and $K_{14}$ to generate the SRN model following rule 2. To generate the system level SRN model we need to combine the entire three SRN model generated for three processor nodes by considering the interconnection among them.

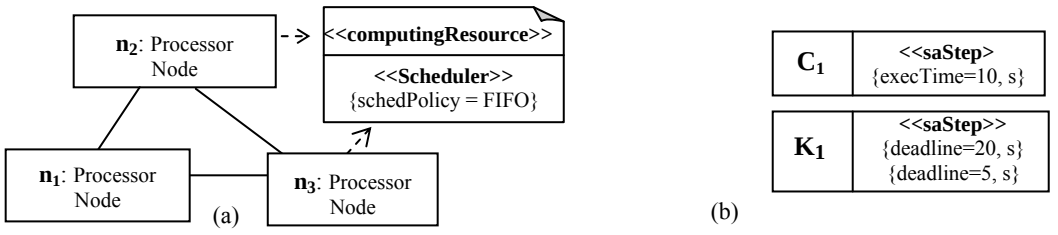


**Figure 15.** (a)The target network of hosts (b) annotated UML model using MARTE profile

**Table. 1.** Optimal deployment mapping in the example scenario

| Node | Components | $\widehat{l}_n$ | $\lvert \widehat{l}_n - T \rvert$ | Internal collaborations |
|---|---|---|---|---|
| $n_1$ | $c_4$, $c_7$, $c_8$ | 70 | 2 | $k_8$, $k_9$ |
| $n_2$ | $c_2$, $c_3$, $c_5$ | 60 | 8 | $k_3$, $k_4$ |
| $n_3$ | $c_1$, $c_6$, $c_9$, $c_{10}$ | 75 | 7 | $k_{11}$, $k_{12}$, $k_{14}$ |
| $\sum$ cost | | | 17 | 100 |

To compose the SRN models of processor node $\boldsymbol{n_1}$ and $\boldsymbol{n_2}$, states $d_4$ and $d_3$ connect by the timed transition $k_1$ and states $d_4$ and $d_5$ connect by the timed transition $k_2$ according to rule 2. Likewise to compose the SRN models of processor node $\boldsymbol{n_2}$ and $\boldsymbol{n_3}$, states $d_2$ and $d_1$ connect by the timed transition $k_5$ and states $d_5$ and $d_1$ connect by the timed transition $k_6$ according to rule 2. To compose the SRN models of processor node $\boldsymbol{n_1}$ and $\boldsymbol{n_3}$, states $d_7$ and $d_1$ connect by the timed transition $k_7$, states $d_8$ and $d_6$ connect by the timed transition $k_{10}$ and states $d_8$ and $d_9$ connect by the timed transition $k_{13}$ according to rule 2. By the above way the system level SRN model is derived. According to rule 4, to define the upper bound of the execution of parallel threads by a network node we introduce three places $PP_1$, $PP_2$ and $PP_3$ in the SRN model for the three network nodes and initially these three places will contain $\boldsymbol{q}$ ($\boldsymbol{q}$ = 1, 2, 3,…….) tokens where $\boldsymbol{q}$ will define the maximum number of the threads that will be handled by a network node at the same time. To ensure the upper bound of the parallel processing of a network node $\boldsymbol{n_1}$ we introduce arcs from place $PP_1$ to transition $t_4$, $t_7$ and $t_8$. That means components $C_4$, $C_7$ and $C_8$ can start their processing if there is token available in place $PP_1$ as the firing of transitions $t_4$, $t_7$ and $t_8$ not only depend on the availability of the token in the place $p_4$, $p_7$ and $p_8$ but also depend on the availability of the token in the place $PP_1$.

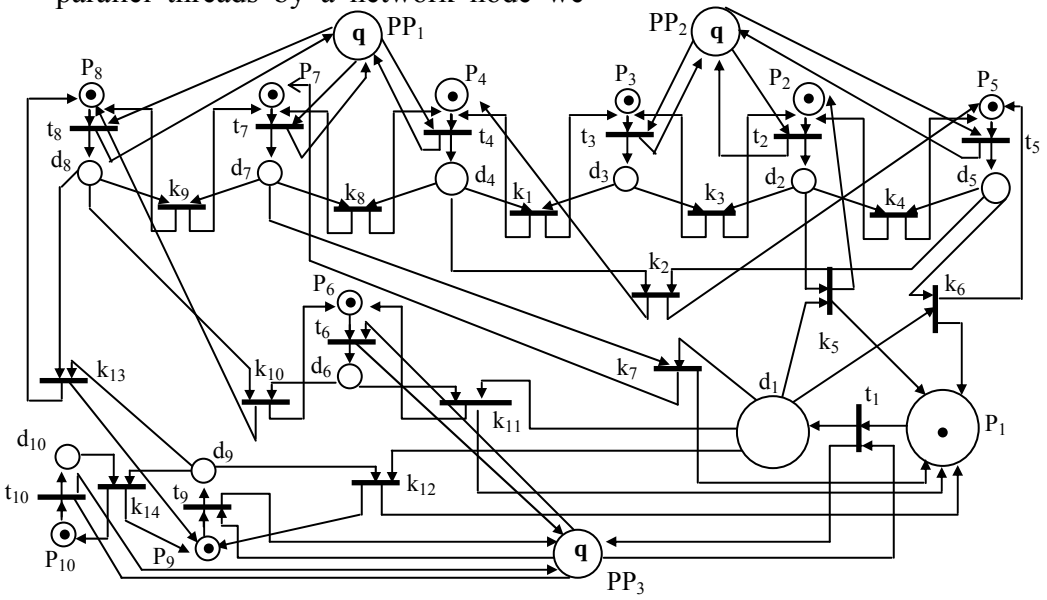


**Figure 16.** SRN model of our example scenario

Likewise to ensure the upper bound of the parallel processing of a network node $n_2$ and $n_3$ we introduce arcs from place $PP_2$ to transition $t_2$, $t_3$ and $t_5$ and from place $PP_3$ to transition $t_1$, $t_6$, $t_9$, $t_{10}$.

In the same way by considering the above same deployment mapping and the transformation rule 1, 3 and 5 the analogous SRN model of our example scenario is depicted in Figure 17 where there is an order in which collaboration roles are selected for completing the execution of their activity which symbolizes the dependency among the execution of collaborations roles' activity.

The throughput calculation according to (6) for the different deployment mapping including the optimal deployment mapping is shown in Table. 2. The throughput is $0.107s^{-1}$ while considers the optimal deployment mapping where E (N) = 6.96 (calculated using SHARPE [15]) and optimal cost = 187s.

The throughput calculation according to (7) for the different deployment mapping including the optimal deployment mapping is shown in Table. 3. The throughput is $2.33\times10^{-4}$ $s^{-1}$ while considers the optimal deployment mapping where E (N) = 0.0435 (calculated using SHARPE [15]) and optimal cost = 187s.

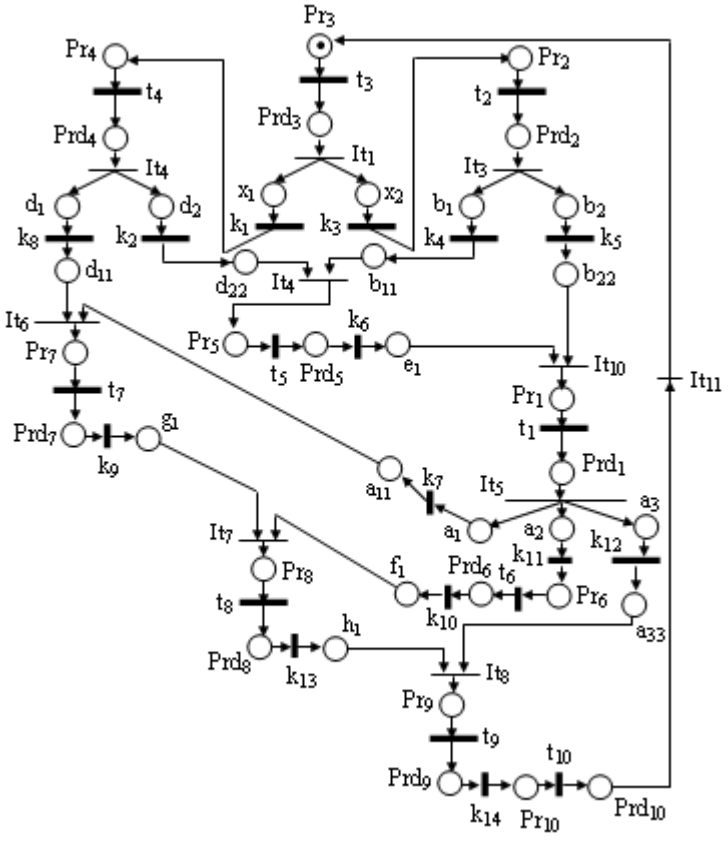


**Figure 17.** SRN model of our example scenario where there is an order in which components are selected for completing the processing

The optimal deployment mapping presented in Table 1 also ensures the optimality in case of throughput calculation for both the SRN performance model shown in Figure 16 and 17. We present here the throughput calculation of some of the deployment mappings of the software artifacts but obviously the approach presented here confirms the efficiency in both deployment mapping and throughput calculation for all the possible cases.

## 4 Conclusion

We present a novel approach for model based performance evaluation of distributed system which spans from capturing the system dynamics through UML diagram as reusable building block to efficient deployment of service components in a distributed manner by capturing the QoS requirements. System dynamics is captured through UML collaboration and activity oriented approach. The behavior of the collaboration and the composition of collaboration to highlight the overall system behavior are demonstrated by utilizing UML activity. Furthermore, quantitative analysis of the system is achieved by generating SRN performance model from the UML specification style. The transformation from UML diagram to corresponding SRN elements like states, different pseudostates and transitions is proposed. Performance related QoS information is taken into account and included in the SRN model with equivalent timing and probabilistic assumption for enabling the evaluation of performance prediction result of the system at the early stage of the system development process. In addition, the logic, as it is presented here, is applied to provide the optimal, initial mapping of components to hosts, i.e. the network is considered rather static. However, our eventual goal is to develop support for run-time redeployment of components, this way keeping the service within an allowed region of parameters defined by the requirements.

**Table. 2.** Optimal deployment mapping in the example scenario

| Node | Components | Possible cost (s) | Throughput ($s^{-1}$) |
|---|---|---|---|
| $\{n_1, n_2, n_3\}$ | $\{\{c_4, c_7, c_8\}, \{c_2, c_3, c_5\}, \{c_1, c_6, c_9, c_{10}\}\}$ | 187 | 0.107 |
| $\{n_1, n_2, n_3\}$ | $\{\{c_4, c_6, c_7, c_8\}, \{c_2, c_3, c_5\}, \{c_1, c_9, c_{10}\}\}$ | 218 | 0.106 |
| $\{n_1, n_2, n_3\}$ | $\{\{ c_4, c_7\}, \{c_2, c_3, c_5, c_6,\}, \{c_1, c_8, c_9, c_{10}\}\}$ | 232 | 0.102 |
| $\{n_1, n_2, n_3\}$ | $\{\{c_5, c_7, c_8\}, \{c_2, c_3, c_4\}, \{ c_1, c_6, c_9, c_{10}\}\}$ | 227 | 0.086 |
| $\{n_1, n_2, n_3\}$ | $\{\{ c_3, c_7, c_8\}, \{c_2, c_4, c_5\}, \{c_1, c_6, c_9, c_{10}\}\}$ | 252 | 0.084 |
| $\{n_1, n_2, n_3\}$ | $\{\{ c_1, c_6, c_7, c_8\}, \{c_2, c_3, c_5\}, \{ c_4, c_9, c_{10}\}\}$ | 257 | 0.083 |
| $\{n_1, n_2, n_3\}$ | $\{\{c_1, c_6, c_7, c_8\}, \{c_2, c_3, c_4\}, \{c_5, c_9, c_{10}\}\}$ | 247 | 0.075 |
| $\{n_1, n_2, n_3\}$ | $\{\{c_4, c_7, c_8\}, \{ c_1, c_2, c_3, c_5\}, \{ c_6, c_9, c_{10}\}\}$ | 217 | 0.073 |
| $\{n_1, n_2, n_3\}$ | $\{\{c_3, c_6, c_7, c_8\}, \{c_1, c_2, c_4, c_5\}, \{c_9, c_{10}\}\}$ | 302 | 0.072 |
| $\{n_1, n_2, n_3\}$ | $\{\{c_6, c_7, c_8\}, \{ c_1, c_2, c_4, c_5\}, \{c_3, c_9, c_{10}\}\}$ | 288 | 0.071 |

**Table. 3.** Optimal deployment mapping in the example scenario when there is an order in which components are selected for completing their activity

| Node | Components | Possible cost (sec) | Throughput ($s^{-1}$) |
|---|---|---|---|
| $\{n_1, n_2, n_3\}$ | $\{\{c_4, c_7, c_8\}, \{c_2, c_3, c_5\}, \{c_1, c_6, c_9, c_{10}\}\}$ | 187 | $2.33\times10^{-4}$ |
| $\{n_1, n_2, n_3\}$ | $\{\{c_4, c_7, c_8\}, \{c_1, c_2, c_3, c_5\}, \{c_6, c_9, c_{10}\}\}$ | 217 | $2.00\times10^{-4}$ |
| $\{n_1, n_2, n_3\}$ | $\{\{c_4, c_6, c_7, c_8\}, \{c_2, c_3, c_5\}, \{c_1, c_9, c_{10}\}\}$ | 218 | $1.99\times10^{-4}$ |
| $\{n_1, n_2, n_3\}$ | $\{\{c_5, c_7, c_8\}, \{c_2, c_3, c_4\}, \{c_1, c_6, c_9, c_{10}\}\}$ | 227 | $1.92\times10^{-4}$ |
| $\{n_1, n_2, n_3\}$ | $\{\{c_4, c_7\}, \{c_2, c_3, c_5, c_6,\}, \{c_1, c_8, c_9, c_{10}\}\}$ | 232 | $1.87\times10^{-4}$ |
| $\{n_1, n_2, n_3\}$ | $\{\{c_4, c_5, c_7, c_8\}, \{c_2, c_3\}, \{c_1, c_6, c_9, c_{10}\}\}$ | 232 | $1.87\times10^{-4}$ |
| $\{n_1, n_2, n_3\}$ | $\{\{c_1, c_6, c_7, c_8\}, \{c_2, c_3, c_4\}, \{c_5, c_9, c_{10}\}\}$ | 247 | $1.76\times10^{-4}$ |
| $\{n_1, n_2, n_3\}$ | $\{\{c_1, c_6, c_7, c_8\}, \{c_2, c_3, c_5\}, \{c_4, c_9, c_{10}\}\}$ | 257 | $1.69\times10^{-4}$ |
| $\{n_1, n_2, n_3\}$ | $\{\{c_6, c_7, c_8\}, \{c_1, c_2, c_4, c_5\}, \{c_3, c_9, c_{10}\}\}$ | 288 | $1.51\times10^{-4}$ |
| $\{n_1, n_2, n_3\}$ | $\{\{c_3, c_6, c_7, c_8\}, \{c_1, c_2, c_4, c_5\}, \{c_9, c_{10}\}\}$ | 302 | $1.44\times10^{-4}$ |

As the results with our proposed framework show our logic will be a prominent candidate for a robust and adaptive service execution platform. However the size of the underlying reachability set to generate SRN model is major limitation for large and complex system. Further work includes automating the whole translation process, the way to solve the performance model and to tackle state explosion problems of reachability marking.

## References

1. F. A. Kramer, R. Bræk, P. Herrmann, "Synthesizes components with sessions from collaboration-oriented service specifications", SDL 2007, V-4745, LNCS, 2007.
2. OMG UML Superstructure, Version-2.2
3. M. Csorba, P. Heegaard, P. Herrmann, "Cost-Efficient Deployment of Collaborating Components", DAIS 2008, LNCS, pp. 253–268.
4. OMG 2009, "UML Profile for MARTE: Modeling & Analysis of Real-Time Embedded Systems", V – 1.0
5. K. S. Trivedi, "Probability and Statistics with Reliability, Queuing and Computer Science application", Wiley- Interscience publication, ISBN 0-471-33341-7
6. J. P. Lopez, J. Merseguer, J. Campos, "From UML activity diagrams to SPN: application to software performance engineering", ACM SIGSOFT software engineering notes, NY, 2004
7. S. Distefano,M. Scarpa, A. Puliafito, "Software Performance Analysis in UML Models", FIRB-PERF, 2005
8. A. D'Ambrogio, "A Model Transformation Framework for the Automated Building of Performance Models from UML Models", WOSP, 2005
9. R. H. Khan, P. E. Heegaard, "Translation from UML to SPN model: A performance modeling framework", EUNICE, 2010
10. R H Khan, P Heegaard, "Translation from UML to SPN model: Performance modeling framework for managing behavior of multiple session & instance" ICCDA 2010
11. F. A. Kramer, "ARCTIS", Department of Telematics, NTNU, http://arctis.item.ntnu.no
12. Rendezvous synchronization, http://book.opensourceproject.org.cn/embedded/cmprealtime/opensource/5107final/lib0091.html, retrieved June, 2010
13. Efe, K., "Heuristic models of task assignment scheduling in distributed systems", Computer (June 1982)
14. R H Khan, P Heegaard, " A Performance modeling framework incorporating cost efficient deployment of collaborating components" ICSTE, 2010
15. K. S. Trivedi, R Sahner, "Symbolic Hierarchical Automated Reliability / Performance Evaluator (SHARPE)", Duke University, Durham, NC.