2011 International Conference on Computer Communication and Management
Proc .of CSIT vol.5 (2011) © (2011) IACSIT Press, Singapore

A Performance Modeling Framework Considering Service Components Deployment Issue

Razib Hayat Khan [+], Poul E. Heegaard

Norwegian University of Science & Technology (NTNU)
7491, Trondheim, Norway

Abstract. Performance evaluation of distributed system is always a complex task where the service behavior results from a collaboration of partial components behavior those are physically distributed. We present an approach to delineate a performance modeling framework in model based service engineering context that proposes a transformation process from high level UML notation to SPN model and solves the model for relevant performance metrics. To demonstrate model based service engineering approach we outline a specification style that focuses on UML collaboration and activity as reusable specification building blocks. Deployment diagram identify the physical components of the system and the assignment of service artifacts to those identified system components. Optimal deployment mapping of the service components on the physically available system resources is investigated by deriving the cost function. The proposed performance modeling framework provides transformation rules of UML elements into corresponding SPN 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 evaluation considering the model based service engineering approach in a distributed environment.

Key words: SPN, UML, 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 of the system 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 SPN model by our proposed performance modeling framework. UML collaboration and activity provides a tremendous service 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].

In addition, 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

[+] Corresponding author. Tel.: + 4745052057; fax: +4773596973.
E-mail address: rkhan@item.ntnu.no

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 at least connected by links with sufficiently high capacity and spread the work load onto the available physical resources by maintaining equilibrium among them with respect to the execution cost [3].

The *Unified Modeling Language* (UML) is a widely accepted modeling language to model the system behavior [2]. But 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 *Profile for Schedulability, Performance, and Time* (SPT) 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 are probably the best studied performance modeling techniques [5]. Among all of them, we will focus on the stochastic petri net as the performance model generated by our proposed 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 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 incorporating service component deployment issue considering reusable components of model. 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 is 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 SPN 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 incorporating efficient deployment of the software components. Incorporating 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 work presented here is the extension of our previous work described in [9] [10] [13] where we presented our proposed framework with respect to the execution of single and multiple collaborative session and considered alternatives system architecture candidate to describe the system behavior and evaluate the performance factors. 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 Fig. 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]. To reason about the correctness of the specifications, we introduce formal reasoning on the level of collaborative service specifications using temporal logic specification style cTLA/c(c for collaborative) [11].

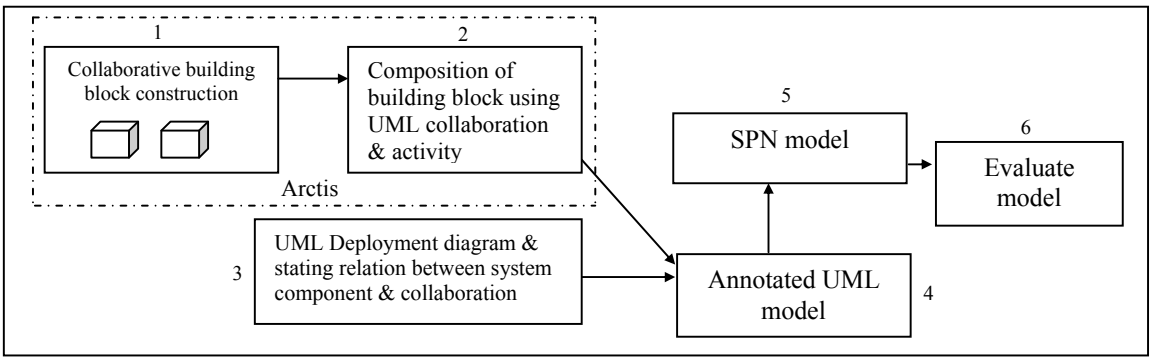


Fig. 1: Proposed performance modeling framework

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. If the building block is elementary it only declares the participants (as collaboration roles) and connection between them. If it is composite, it may additionally refer to other collaborations between the collaboration roles by means of collaboration uses. 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 Fig. 2 (a) 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 Fig.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 from the participant *A* to *B*. The token is then transferred by the participant *A* to participant *B* represented by the call behavior action *forward* after completion of the processing by the collaboration role *A*. After getting the response of the participant *A* the participant *B* starts the processing of the request (indicated by the streaming pin *req*).

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. 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]. For example the detailed behavior and composition of the collaboration is given in following Fig. 3 (a). The initial node (●) indicates the starting of the activity. The activity is started from the participant *A*. After being activated, each participant starts its processing of request which is mentioned by call behavior action Pr_i *(*$Processing_i$*, where i = A, B & C)*. Completions of the processing by the participants are mentioned by the call behavior action Prd_i *(*$Processing\_done_i$*, where i = A, B & C).* After completing the processing, the responses are delivered to the corresponding participants. When the processing of the task by the participant *A* completes the response (indicated by streaming pin *res)* is transferred to the participant *B* mentioned by collaboration *t:* $transfer_{ij}$ (*where ij = AB)*. After getting the response form the participant *A* participant *B* starts the processing of the request (indicated by the streaming pin *req*) and transfers the response to the participant *C* mentioned by collaboration *t:* $transfer_{ij}$ (*where ij = BC)*. Participant *C* starts the processing of the request after getting the response form the participant *B* and then activity is terminated which is mentioned by the terminating node (◉).

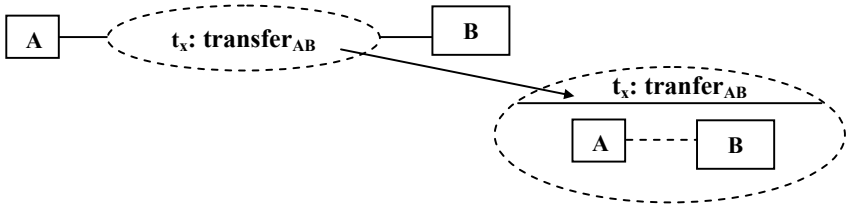


Fig. 2: Structure of the Building block

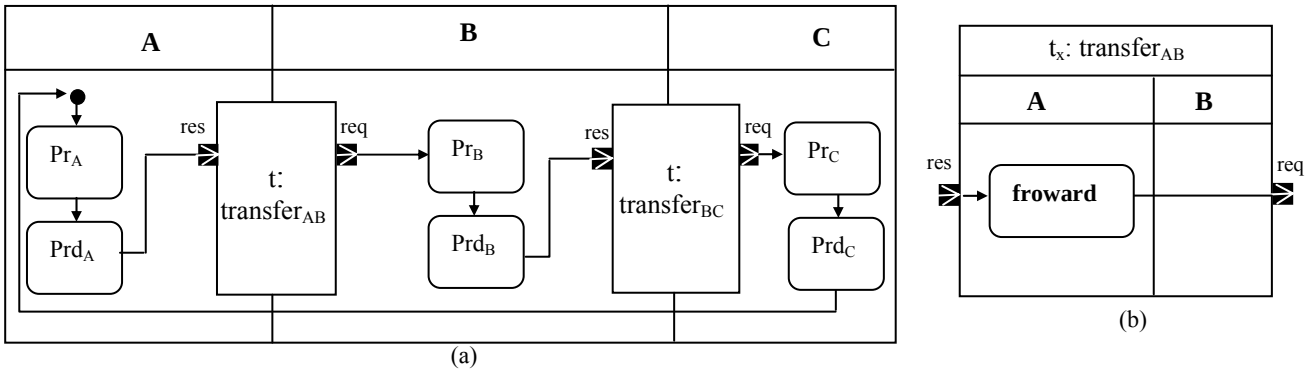


Fig. 3: (a) Detail behavior of the event of the collaboration using activity (b) internal behavior of the collaboration

Step 3: Designing UML deployment diagram & stating relation between system components & collaborations: We model the system as collection of N interconnected nodes shown in Fig. 4. 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 → 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 ∈ N is a physical resource. Generally, nodes can have different responsibilities, such as providing 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

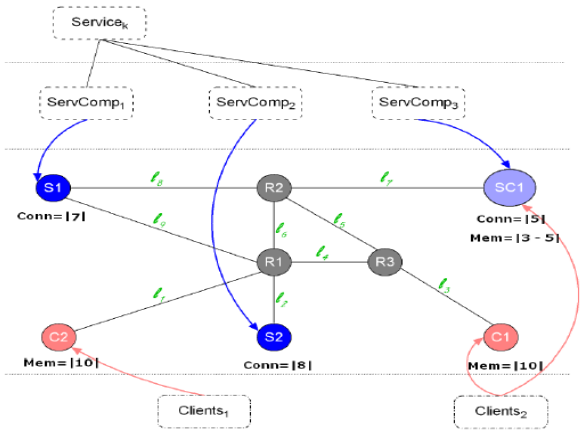


Fig. 4: Service component mapping example

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 f_{kj} 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 f_{kj}, j = 1….K and cost for executing the background process f_{Bj}, j = 1….k. fc_i, f_{kj} 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 [3]: $T = \frac{\sum_{i=1}^{E} fc_i}{N}$

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):

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

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. 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 [15].

Step 4: Annotating the UML model: Performance information is incorporated into the UML activity diagram and deployment diagram according to UML Profile for Schedulability, Performance & Time (SPT) [4] for evaluating system performance by performance model solver.

Step 5: Deriving the SPN model: Here we define our rules for transforming into SPN model by utilizing the specification of reusable building blocks. 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 SPN performance model will be generated. The rules are based on decomposition of UML collaboration and activity diagram into basic elements of SPN model like states as places, timed transition and immediate transition [12]. The rules are following:

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

Φ = 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 $\subseteq$ {Φ × T} $\cup$ {T × Φ} is a set of arcs connecting Φ 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: Φ → {0, 1, 2...} is the marking that denotes the number of tokens for each place in Φ. The initial marking is denoted as m_0.

Rule1: The SPN model of the collaboration role of a reusable building block is represented by the 6-tuple in the following way:

$\Phi_A = \{Pr_A, Prd_A\}$

T = {do, exit}

$A = \{\{(Pr_A \times do) \cup (do \times Prd_A)\}, \{(Prd_A \times exit) \cup (exit \times Prd_A)\}\}$

K = (do → Timed, exit → Immediate)

$N = \{(Pr_A \times do) \rightarrow 1, (do \times Prd_A) \rightarrow 1, (Prd_A \times exit) \rightarrow 1, (exit \times Pr_A) \rightarrow 1\}$

$m_o = \{(Pr_A \rightarrow 1\}, (Prd_A \rightarrow 0)\}$

Here places are represented by Pr_A and Prd_A. Transitions are represented by *do* and *exit* where *do* is a timed transition and *exit* is an immediate transition. Initially place Pr_A contains one token and place Prd_A contains no token. The first figure highlights the SPN model of the collaboration role *A* where *A* has its own token to start the execution of the SPN model and the second figure highlights the SPN model of the collaboration role *A* where the starting of the execution of the SPN model of *A* depends on the receiving of token from other element.

Rule2: The SPN model of a collaboration where collaboration connects only two collaboration roles is represented by the 6-tuple in the following way:

$\Phi = \{\Phi_A, \Phi_B\} = \{Pr_A, Prd_A, Pr_B, Prd_B\}$

$T = \{do_A, do_B, t\}$

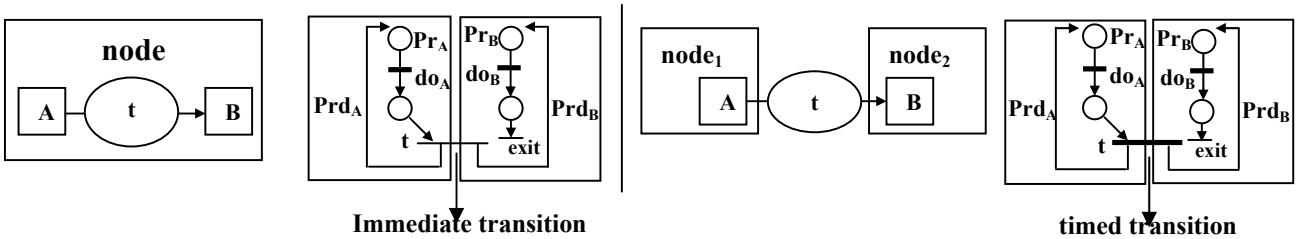
$A = \{\{(Pr_A \times do_A) \cup (do_A \times Prd_A)\}, \{(Prd_A \times t) \cup ((t \times Pr_A), (t \times Pr_B))\}, \{(Pr_B \times do_B) \cup (do_B \times Prd_B)\}\ \{(Prd_B \times exit) \cup (\emptyset)\}\}$

$K = (do_A \rightarrow Timed, do_B \rightarrow Timed, t \rightarrow Timed \mid Immediate)$

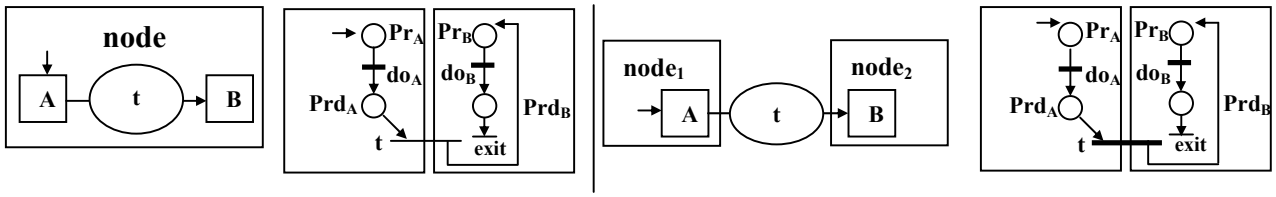
$N = \{(Pr_A \times do_A) \rightarrow 1, (do_A \times Prd_A) \rightarrow 1, (Prd_A \times t) \rightarrow 1, (t \times Pr_A) \rightarrow 1, (t \times Pr_B) \rightarrow 1, (Pr_B \times do_B) \rightarrow 1, (do_B \times Prd_B) \rightarrow 1, (Prd_B \times exit) \rightarrow 1\}$

$m_o = \{(Pr_A \rightarrow 1, Prd_A \rightarrow 0, Pr_B \rightarrow 1, Prd_B \rightarrow 0\}$.

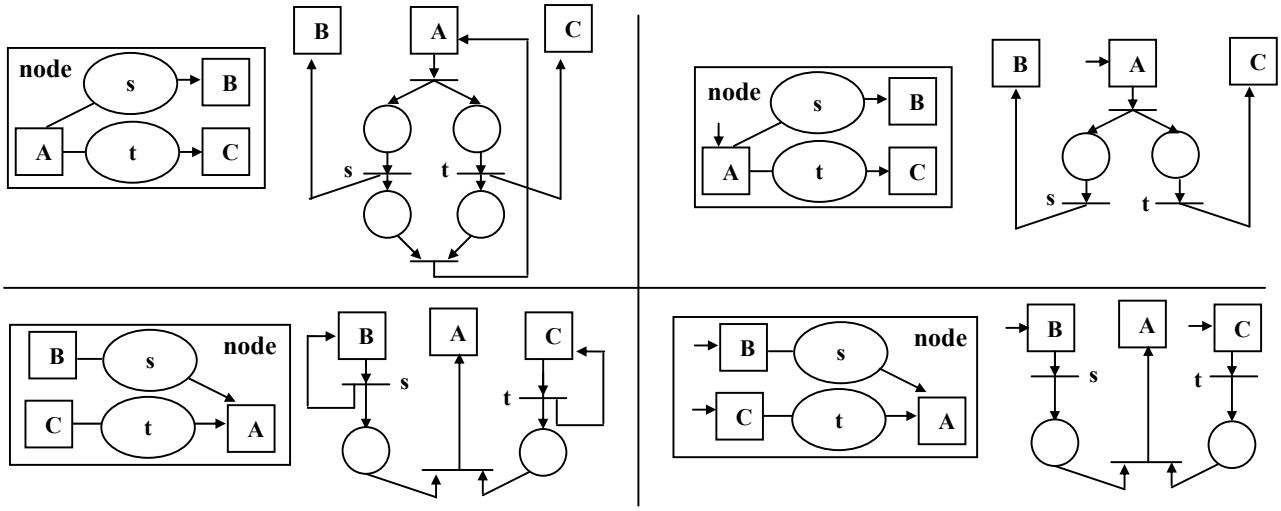
Here places are represented by Pr_A, Prd_A, Pr_B, Prd_B, transitions are represented by do_A, do_B and t where do_A and do_B are timed transition and t is an immediate transition if the two collaboration roles deploy on the same physical node (communication time = 0) or timed transition if the two collaboration roles deploy on the different physical nodes (communication time > 0). Initially place Pr_A and Pr_B contains one token and place Prd_A and Prd_B contains no token. SPN model of collaboration is represented graphically by the following way:



The below figures highlight the SPN model of the collaboration where the starting of the execution of the SPN model of collaboration role *A* depends on the receiving of token from other element.



Rule3: 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 SPN model. This rule can be demonstrated through 6-tuple in the above way. The graphical representations of the SPN model for composite structures are shown in the below figures:



Step 6: Evaluate the model: We focus on measuring the throughput of the system from the developed SPN model. 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 SPN model using SHARPE [14]. The value of *C_Net* is evaluated from the equation (1) where *C_Net* = F (M).

$$\therefore \text{Throughput} = \frac{\mathbf{E(N)}}{\mathbf{C\_Net}} \qquad \ldots\ldots\ldots\ldots \qquad (2)$$

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 [12]. This scenario, even though artificial and may not be tangible from a designer's point of view, 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 Fig. 5. We consider four types of requirements in this specification. Besides the execution costs (Exec. cost) and communication costs (Comm. cost) and cost for running background process (Run BP cost), 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.

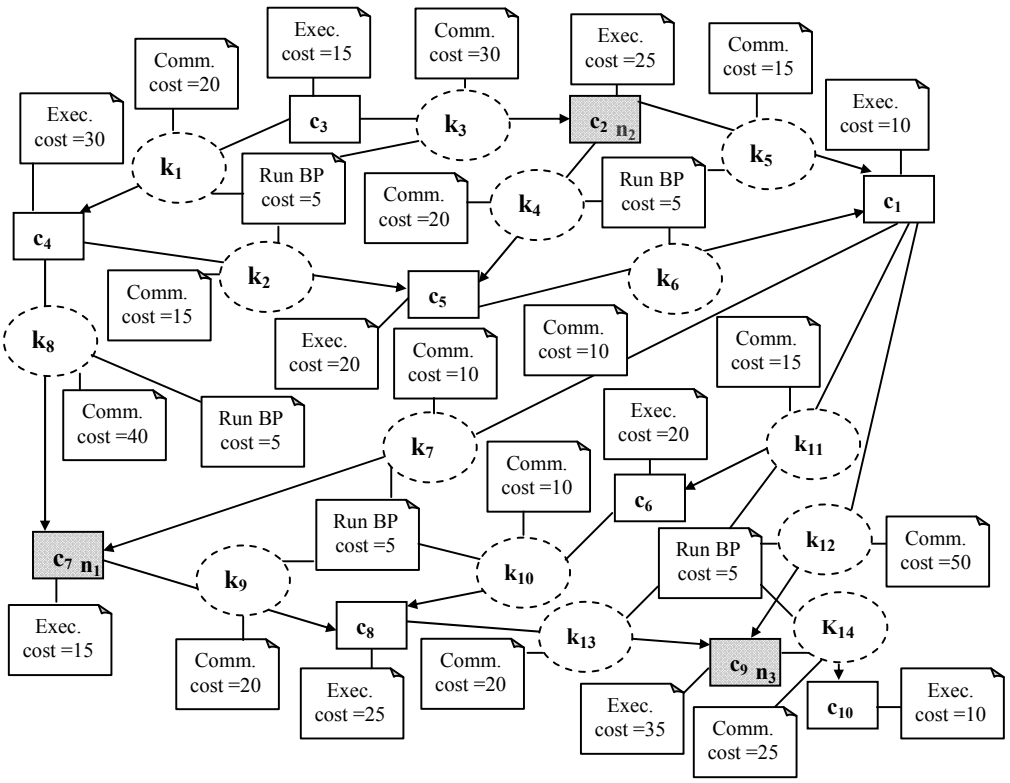


Fig. 5: Service components parameters in the example scenario

The internal behavior of the collaboration *K* is realized by the call behavior action through UML activity like structure already demonstrated in Fig.3 (b). The composition of the collaboration role *C* of our example scenario as reusable building block is demonstrated in Fig. 6. The initial node (●) indicates the starting of the activity. After being activated, each participant starts its processing of request which is mentioned by call behavior action Pr_i (Processing of the ith service component). Completions of the processing by the participants are mentioned by the call behavior action Prd_i *(*Processing done of the ith service component). The activity is started from the component C_3 where the semantics of the activity is realized by the token flow. After completion of the processing of the component C_3 the response is divided into two flows which are shown by the fork node f_3. After getting the response from the component C_3 the components C_4 and C_2 starts their processing of the request. The response and request is mentioned by the streaming pin *res* and *req*. In both the component C_4 and C_2 the response is divided into two flows which are highlighted by the fork node f_4 and f_2. Component C_5 starts its processing after getting the response both from the component C_4 and C_2 which is realized by the join node j_5. Processing of the component C_1 starts after arrival of the response both from the component C_2 and C_5 and the response of the component C_1 is divided into three flows which transfer to component C_6, C_7 *and* C_9. The processing of the component C_7 starts when the response arrives both from component C_4 and C_1 and pass through the join node j_7. After getting the response from component C_7 and C_6 component C_8 starts it's processing. The synchronization between the arrivals of the responses in component C_8 is maintained by the join node j_8. Likewise the processing of the component C_9 starts when it gets the responses from both the component C_1 and C_8 and responses pass through the join node j_5. After completion of the processing of component C_9 component C_{10} starts its processing and later on activity is terminated which is mentioned by the end node (◉).

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 Fig. 7. The optimal deployment mapping can be observed in Table 1. The lowest possible deployment cost, according to (1) is $17 + (270 - 100) = 187$.

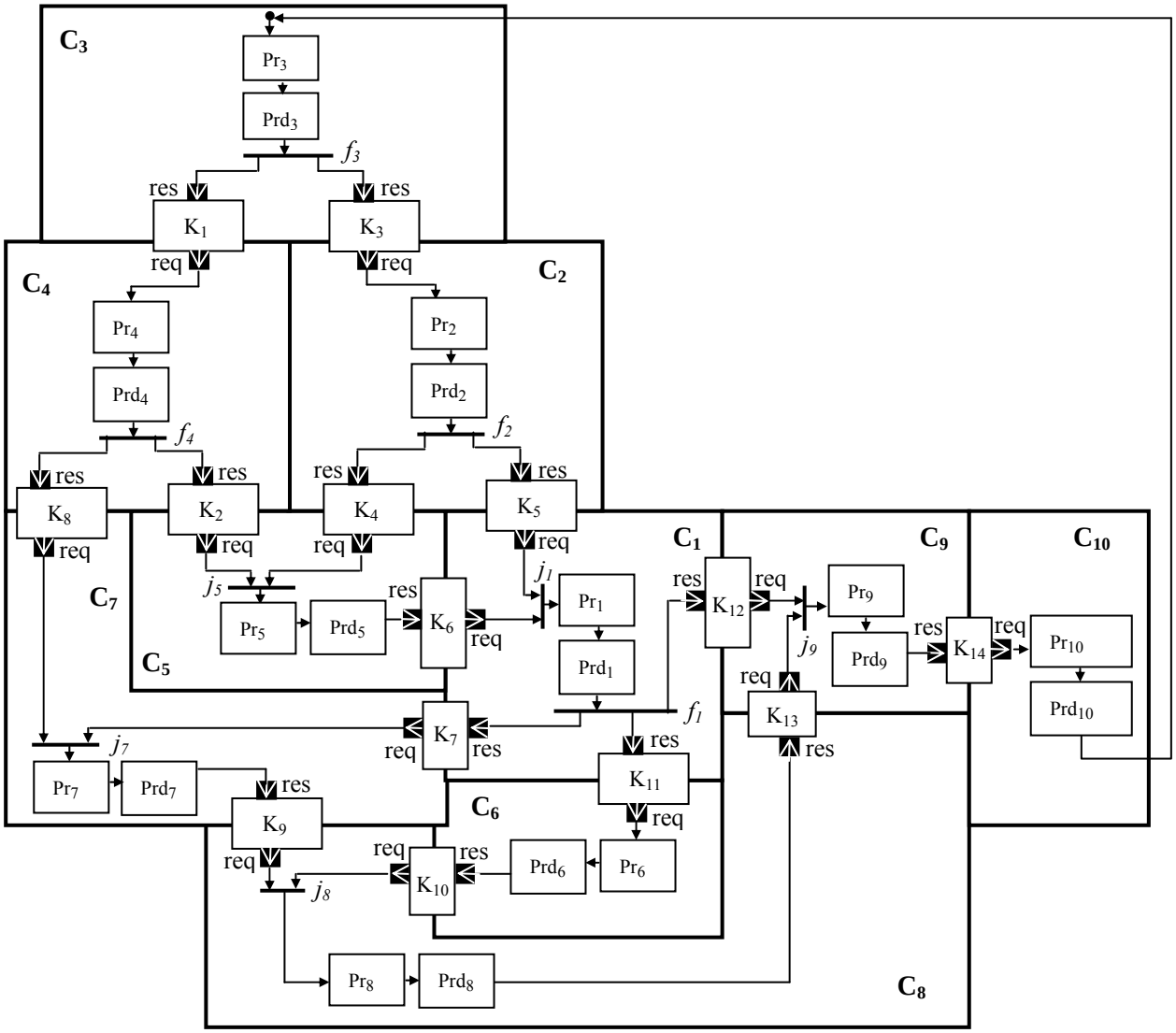


Fig. 6: Detail behavior of the event of the collaboration using activity for our example scenario

To annotate the UML diagram in Fig. 6 & 7 we use the stereotype *RTaction, PAhost* and the tag value *RTduration, PAschdPolicy*. *RTaction* models any action that takes time. The duration of the time is mentioned by the tag value *RTduration*. A *PAhost* models a processing resource with tags *PAschdPolicy* defining the policy by which access to the resource is controlled. The annotation of service component and collaboration of Fig. 6 is shown in Table. 2. Collaboration K_i is associated with two instances of stereotype *RTaction* as all the collaborations in our example scenario are associated with two costs: communication cost and cost for running backgrounds process.

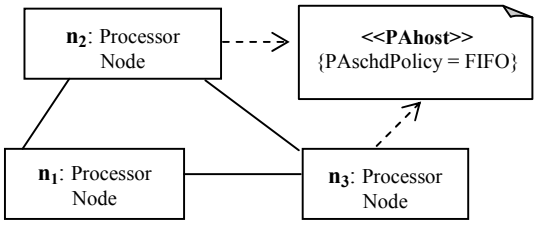


Fig. 7: The target network of hosts

Table 1: Optimal deployment mapping in the example scenario

Node	Components	$\hat{l}$ n	$\lvert \hat{l} n - T \rvert$	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

Table 2: Annotating UML model according to SPT

C_1	<<RTaction>> {RTduration=10, s}

K_1	<<RTaction>> {RTduration=20, s} <<RTaction>> {RTduration=5, s}

By considering the above composition of the collaborations, deployment mapping and the transformation rule the analogous SPN model of our example scenario is depicted in Fig. 8. The states of the SPN model are derived from the call behavior action of the corresponding collaboration role and collaboration among them. According to the transformation rules 1, each collaboration role is defined by the two states Pr_i and Prd_i (stands for *Processingi* and *Processing_done*$_i$ for the ith service component) and the passing of token from state Pr_i to Prd_i is realized by the timed transition t_i which is derived from the annotated UML model. For generating the SPN model firstly we will consider the collaboration role C_3 deployed on the processor node $\boldsymbol{n_2}$ as the execution will be started from the component C_3 for our example scenario. After the completion of the state transition from Pr_3 to Prd_3 (states of component C_3) the flow is divided into two branches according to rule 4 and the token will be passed to place Pr_4 (states of component C_4) and Pr_2 (states of component C_2). The timed transition K_1 is realized both by the communication cost and cost for running background process as C_3 and C_4 deploy on the two different nodes $\boldsymbol{n_2}$ and $\boldsymbol{n_1}$. Though according to rule 2 collaboration K_3 is an immediate transition as C_3 and C_2 deploy on the same processor node $\boldsymbol{n_1}$ but it is marked by a timed transition as it is still realized by the cost for running background process. Likewise after the completion of the state transition from Pr_4 to Prd_4 (states of component C_4) and the state transition from Pr_2 to Prd_2 (states of component C_2) both the flows are divided into two branches according to rule 4. Place Pr_5 will get the token just after the firing of the immediate transition It_4 whose firing depends on getting the token from both the component C_4 and C_2 and completes the state transition from Pr_5 to Prd_5 (states of component C_5). The timed transition K_2 is realized both by the communication cost and cost for running background process as C_4 and C_5 deploy on the two different nodes $\boldsymbol{n_1}$ and $\boldsymbol{n_2}$. But timed transition K_4 is realized only by the cost for running background process as C_2 and C_5 deploy on the same processor node $\boldsymbol{n_2}$. Place Pr_1 will get the token just after the firing of the immediate transition It_{10} whose firing depends on getting the token from both the component C_5 and C_2. According to rule 4 the flow will be divided into three branches after completion of the state transition from Pr_1 to Prd_1 (states of component C_1). The timed transition K_6 and K_5 is realized both by the communication cost and cost for running background process as C_1 and C_5 deploy on the two different processor nodes $\boldsymbol{n_3}$ and $\boldsymbol{n_2}$ and C_1 and C_2 deploy on the two different processor nodes $\boldsymbol{n_3}$ and $\boldsymbol{n_2}$. Place Pr_7 will get the token just after the firing of the immediate transition It_6 whose firing depends on getting the token from both the component C_1 and C_4 and completes the state transition from Pr_7 to Prd_7 (states of component C_7). The timed transition K_7 is realized both by the communication cost and cost for running background process as C_1 and C_7 deploy on the two different nodes $\boldsymbol{n_3}$ and $\boldsymbol{n_1}$. But timed transition K_8 is realized only by the cost for running background process as C_4 and C_7 deploy on the same processor node $\boldsymbol{n_1}$. Component C_6 completes its state transition from place Pr_6 to Prd_6 just after the firing of the timed transition K_{11} which is realized only by the cost for running background process as C_1 and C_6 deploy on the same processor node $\boldsymbol{n_3}$. Place Pr_8 will get the token just after the firing of the immediate transition It_7 whose firing depends on getting the token from both the component C_6 and C_7 and completes the state transition from Pr_8 to Prd_8 (states of component C_8). The timed transition K_{10} is realized both by the communication cost and cost for running background process as C_6 and C_8 deploy on the two different nodes $\boldsymbol{n_3}$ and $\boldsymbol{n_1}$. But timed transition K_9 is realized only by the cost for running background process as C_7 and C_8 deploy on the same processor node $\boldsymbol{n_1}$. Place Pr_9 will get the token just after the firing of the immediate transition It_8 whose firing depends on getting the token from both the component C_8 and C_1 and completes the state transition from Pr_9 to Prd_9 (states of component C_9). The timed transition K_{13} is realized both by the communication cost and cost for running background process as C_8 and C_9 deploy on the two different nodes $\boldsymbol{n_1}$ and $\boldsymbol{n_3}$. But timed transition K_{12} is realized only by the cost for running background process as C_1 and C_9 deploy on the same processor node $\boldsymbol{n_3}$. Component C_{10} completes its basic transition from place Pr_{10} to Prd_{10} just after the firing of the timed transition

K_{14} which is realized only by the cost for running background process as C_9 and C_{10} deploy on the same processor node n_3.

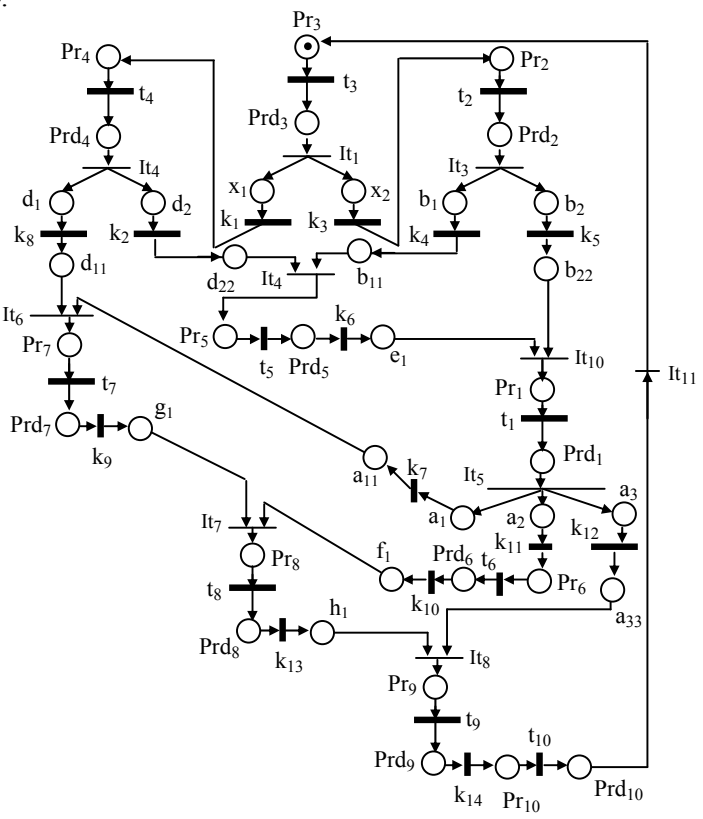


Fig. 8: SPN model of the example scenario

The throughput calculation according to (2) for the different deployment mapping including the optimal deployment mapping is shown in Table. 3. The throughput is 2.33×10^{-4} s^{-1} while considers the optimal deployment mapping where E (N) = 0.0435 (calculated using SHARPE [14]) and C_Net = 187s. The optimal deployment mapping presented in Table 1 also ensures the optimality in case of throughput calculation. We present here the throughput calculation of some of the deployment mapping of the software artifacts but obviously the approach presented here confirms the efficiency in both deployment mapping and throughput calculation.

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 SPN performance model from the UML specification style. The transformation from UML diagram to corresponding SPN elements like states, different pseudostates and transitions is proposed. Performance related QoS information is taken into account and included in the SPN 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. As the results with our proposed framework show our logic will be a prominent candidate for

Table 3. Possible cost and throughput for different deployment mapping

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	2.33×10^{-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	2.00×10^{-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	1.99×10^{-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	1.92×10^{-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	1.87×10^{-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	1.87×10^{-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	1.76×10^{-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	1.69×10^{-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	1.51×10^{-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	1.44×10^{-4}

a robust and adaptive service execution platform. However the size of the underlying reachability set to generate SPN model is major limitation for large and complex system. Further work includes automating the whole translation process from our UML specification style to generate a SPN performance model and the way to solve the model through our proposed framework & to tackle state explosion problems of reachability marking of large system.

5. References

[1] F. A. Kramer, R. Bræk, P. Herrmann. Synthesizes components with sessions from collaboration-oriented service specification. *Proceedings of SDL 2007*. V-4745, Lecture notes of Computer Science, 2007.

[2] OMG UML Superstructure, Version-2.2

[3] M. Csorba, P. Heegaard, P. Herrmann. Cost-Efficient Deployment of Collaborating Components. *DAIS 2008*. LNCS

[4] OMG 2005. UML Profile for Schedulability, Performance, & Time Specification. V – 1.1

[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. 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] Efe, K. Heuristic models of task assignment scheduling in distributed systems. *Computer* (June 1982)

[13] R H Khan, P Heegaard. A Performance modeling framework incorporating cost efficient deployment of collaborating components. *ICSTE* 2010

[14] K. S. Trivedi, R. Sahner, "Symbolic Hierarchical Automated Reliability / Performance evaluator (SHARPE)", Duke University, Durham, NC.

[15] Mate J Csorba, P. Heegaard. Swarm Intelligence Heuristics for component deployment, *EUNICE* 2010.