From UML to SRN: A Performability Modeling Framework Considering Service Components Deployment

Razib Hayat Khan
Department of Telematics
NTNU, Norway
rkhan@item.ntnu.no

Fumio Machida
Service Platform Research
NEC, Japan
h-machida@ab.jp.nec.com

Poul E. Heegaard
Department of Telematics
NTNU, Norway
poul.heegaard@item.ntnu.no

Kishor S. Trivedi
Department of ECE
Duke University, NC, USA
kst@ee.duke.edu

***Abstract*-Conducting the performance modeling of distributed system separately from the dependability modeling fails to asses the anticipated system performance in the presence of system components failure and recovery. System dynamics is affected by any state changes of the system components due to failure and recovery. This introduces the concept of performability that considers the behavioral change of the system components due to failures and also reveals how this behavioral change affects the system performance. But, to design a composite model for distributed system, perfect modeling of the overall system behavior is crucial and sometimes very cumbersome. Additionally evaluation of the required measures by solving the composite model are also intricate and error prone. Bearing this concept in mind, we delineate a performability modeling framework for a distributed system that proposes an automated transformation process from high level UML notation to SRN model and solves the model to generate various numerical results. To capture 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 identifies the physical components of the system and the assignment of software artifacts to the identified system components. Optimal deployment mapping of software artifacts on the available physical resources of the system is investigated by deriving the cost function. State machine diagram is utilized to capture state changes of system components such as failure and recovery. Later on, model composition is achieved by assigning guard function.**

Keywords: UML, SRN, Performability, Deployment

I. INTRODUCTION

The analysis of the system behavior from the pure performance viewpoint tends to be optimistic since it ignores the failure and repair behavior of the system components. On the other hand, pure dependability analysis tends to be too conservative since performance considerations are not taken into account [3]. When the service is deployed it might be the case that something goes wrong in the system because of performance or dependability bottlenecks of the resources and that might adversely affects the service request completion. This bottleneck is an impediment to assure the effectiveness and efficiency requirements to achieve the purpose of system to deliver services proficiently and in timely manner [2]. Therefore, in real systems, availability, reliability and performance are important QoS indices which should be investigated in a combined manner that introduces the concept performability. Performability considers the effect of state changes because of failure and recovery of the system components and their impact on the overall performance of the system [1]. Bearing the above concept we therefore introduce a performability modeling framework for distributed system to allow modeling of the performance and dependability related behaviors in a combined way not only to model functional attributes of the service provided by the system but also to investigate dependability attributes to reflect how the changes in the dependability attributes affect the system performance. For ease of understanding the complexity behind the modeling of performability attributes the proposed modeling framework works in two different layers such as performance modeling layer and dependability modeling layer. The proposed framework achieves its objective by maintaining harmonization between performance and dependability modeling layer with the assist of model synchronization.

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 model the behavior of the distributed objects perfectly for appropriate demonstration of the system dynamics. Hence we adopt UML (Unified Modeling Language) collaboration, state machine and activity oriented approach as UML is the most widely used modeling language which models both the system requirements and qualitative behaviors through different notations [4]. Collaboration and activity diagram are utilized in the performance modeling layer to demonstrate the overall system behavior by defining both the structure of the partial object behaviors as well as the interaction between them. State machine is employed in the dependability modeling layer to capture system component behavior with respect to failure and repair events. Later the UML specification styles are applied to generate the SRN (Stochastic Reward Net) model automatically by our proposed framework. SRN models generated in both performance and dependability modeling layer are synchronized by the model synchronization role by designing guard functions (a special property of the SRN model [5]) to properly model the system performance behavior with respect to any state changes in the system due to component failure [1]. The proposed modeling framework considers system architecture to realize the deployment of the service components. Abstract view of the system architecture is captured by the UML deployment diagram,

 ISBN: 978-1-61208-186-1

which defines the execution architecture of the system by identifying the system components and the assignment of software artifacts to those identified system components [4]. Considering the system architecture to design the proposed framework 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. Moreover, UML models are annotated according to the *UML profile for MARTE* [7] and *UML profile for Modeling Quality of Service and Fault Tolerance Characteristics & Mechanisms* to include quantitative system parameters [12].

Markov model, SPN (Stochastic Petri Nets) and SRN are probably the best studied performability modeling techniques [3]. Among all of them, we will focus on the SRN model generated by our proposed framework due to its some prominent and interesting properties such as priorities assignment in transitions, presence of guard functions for enabling transitions that can use entire state of the net rather than a particular state, marking dependent arc multiplicity that can change the structure of the net, marking-dependent firing rates, and reward rates defined at the net level [5].

Several approaches have been followed to conduct the performability analysis model from system design specification [8] [9] [10] [11]. However, most existing approaches do not highlight more on the issues that how to optimally conduct the system modeling to capture system dynamics and to conduct performability 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 to characterize system dynamics. 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. This reusability provides the opportunity to design new system's behavior rapidly utilizing the existing building blocks according to the specification rather than starting the design process from the scratch. In addition the resulting deployment mapping provided by our framework has greater impact with respect to QoS provided by the system. Our aim here is to deal with vector of QoS properties rather than restricting 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 hosts 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. Moreover the introduction of model synchronization activity relinquishes the complexity and unwieldy affects in modeling and evaluation task of large and multifaceted systems. Model synchronization hides the intricacy behind demonstration of composite model behavior by designing guard functions [5]. Guard functions take charge of the proper functioning of the composite model by considering any changes either in the performance model or in the dependability model.

The paper is organized as follows: Section II introduces our proposed modeling framework, Section III depicts UML based model description, Section IV explains service component deployment issue, Section V clarifies model annotation, Section VI delineates model translation rules, Section VII introduces the model synchronization mechanism, Section VIII describes the fault tree model, Section IX demonstrates the application example to show the applicability of our modeling framework and Section X delineates the conclusion with future directions.

II. OVERVIEW OF PROPOSED FRAMEWORK

Our proposed performability framework is composed of 2 layers: performance modeling layer and dependability modeling layer. The performance modeling layer mainly focuses on capturing the system's dynamics to deliver certain services deployed on a distributed system. The performance modeling layer is divided into 5 steps shown in Fig.1 where the first 2 steps are the parts of Arctis tool suite which is integrated as plug-ins into the eclipse IDE [14]. Arctis focuses on the abstract, reusable service specifications that are composed form UML 2.2 collaborations and activities [14]. It uses collaborative building blocks to create comprehensive services through composition. To support the construction of building block consisting of collaborations and activities, Arctis offers special actions and wizards.

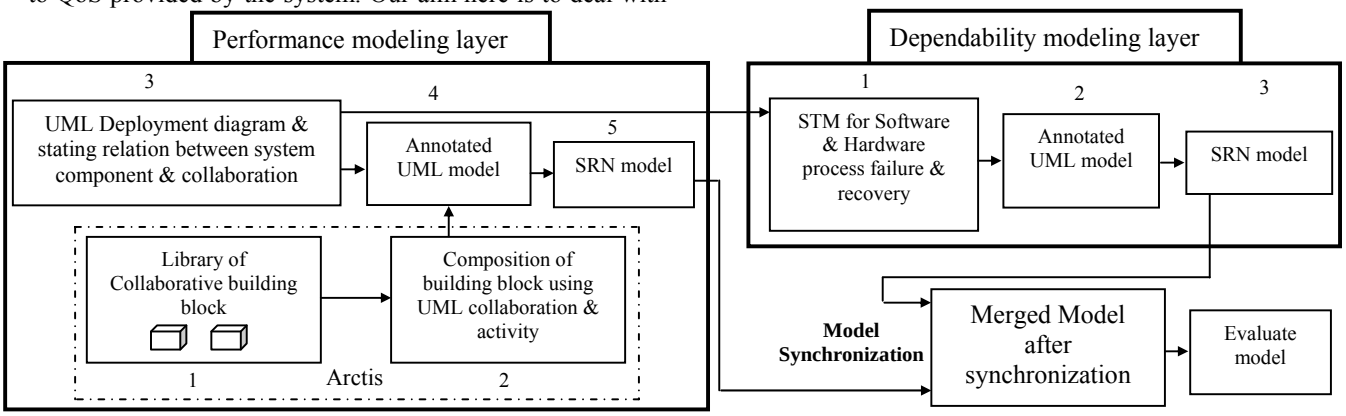


Figure 1. Proposed performability modeling framework

In the first step of performance modeling layer, a developer consults a library to check if an already existing basic collaboration role block or collaboration between several blocks solve a certain task. Missing blocks can also be created from existing building blocks 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. In the second step, the building blocks are combined into more comprehensive service by composition 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 this composition, UML collaborations and activities are used complementary to each other [14]. In the third step, the deployment diagram of our proposed system is delineated and the relationship between system component and collaboration is outlined to describe how the service is delivered by the joint behavior of the system components. In the fourth step, performance information is incorporated into the UML activity diagram and deployment diagram according to *UML profile for MARTE* [7]. The next step is devoted to automate generation of SRN model following the transformation rules. The SRN model generated in this layer is called performance SRN.

The dependability modeling layer is responsible for capturing any state changes in the system because of failure and recovery behaviors of system components. The dependability modeling layer is composed of three steps shown in Figure. 1. In the first step, UML state machine diagram (STM) is used to describe the state transitions of software and hardware components of the system to capture the failure and recovery behaviors. In the next step, dependability parameter is incorporated into the STM diagram according to *UML profile for Modeling Quality of Service and Fault Tolerance Characteristics & Mechanisms Specification* [12]. The last step reflects the automated generation of the SRN model from the STM diagram following the defined transformation rules. The SRN model generated in this layer is called dependability SRN.

The model synchronization is used as glue between performance SRN and dependability SRN. The synchronization task guides performance SRN to synchronize with the dependability SRN by identifying the transitions in the dependability SRN. The synchronization between performance and dependability SRN is achieved by defining the guard functions. Once the performance SRN model synchronized with dependability SRN model a merged SRN model will be obtained and various performability measures can be evaluated from the merged model using the software package such as SHARPE [15].

III. UML BASED SYSTEM DESCRIPTION

Construction of collaborative building blocks: The proposed framework utilizes collaboration as main entity. Collaboration is an illustration of the relationship and interaction among software objects in the UML. Objects are shown as rectangles with naming label inside. The relationships between the objects are shown as line connecting the rectangles [4]. The specifications for collaborations here 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. 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, 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 where it only declares the participants *A* and *B* as collaboration roles and the connection between them is defined as collaboration t_x $(x=1\ldots n_{AB}$ (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 forwarding 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 operation 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*).

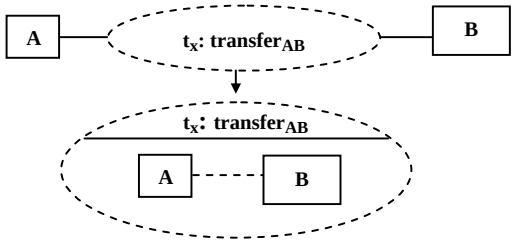


Figure 2. Structure of the Building block

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. Therefore, the activity contains a separate call behavior action for all collaboration of the system. Collaboration is represented by connecting their input and output pins. Arbitrary logic between pins may be used to synchronize the building block events and transfer data between them.

 ISBN: 978-1-61208-186-1

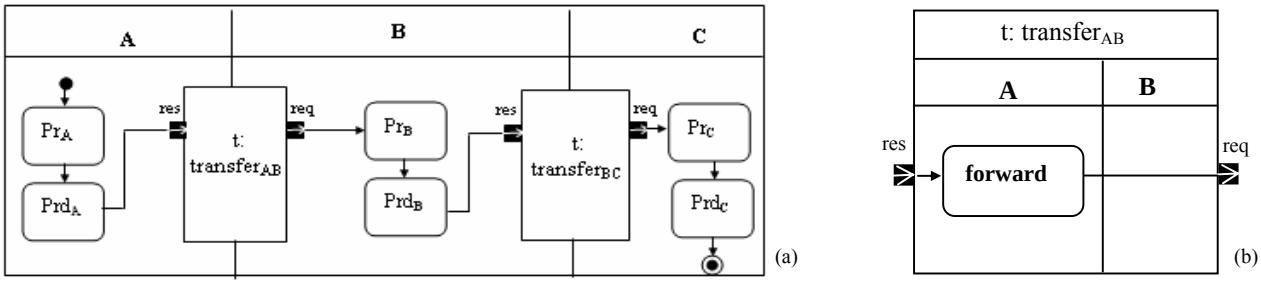


Figure 3. (a) Detail behavior of the event of the collaboration using activity (b) internal behavior of the collaboration

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 [14]. 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 operation action Pr_i *(Processing$_i$, where i = A, B & C).* Completion of the processing by the participants are mentioned by the call operation action *Prdi (Processing_done$_i$, where i = A, B & C).* After completion of the processing, the response is delivered to the corresponding participant. 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)* and participant *B* starts the processing of the request (indicated by streaming pin *req).* After completion of processing participant B transfers the response to the participant *C* mentioned by collaboration *t: transfer$_{ij}$* (*where ij = BC).* Participant *C* starts the processing after getting the response form *B* and activity is terminated after completion of the processing which is mentioned by the terminating node (◉).

Modeling failure & repair behavior of software & hardware component using STM: State transitions of a system element are described using STM diagram. In an STM, a state is depicted as a rounded rectangle and a transition from one state to another is represented by an arrow. Here STM is used to describe the failure and recovery behavior of software and hardware component. The STM of software process is shown in Fig. 4(a). The initial node (●) indicates the starting of the operation of software process. Then the process enters Running state. Running is the only available state in the STM. If the software process fails during the operation, the process enters Failed state. When the failure is detected by the external monitoring service the software process enters Recovery state and the repair operation will be started. When the failure of the process is recovered the software process returns to Running state. The STM of hardware node is shown in Fig. 4 (b). States of the hardware node start from the Stop state. The hardware node starts the operation when the on command is invoked and the node enters Running state. Running is the only available state here. If the node fails during the operation, the node enters Failed state. When the failure is detected the repair operation of the hardware node is started. When the failure of the node is repaired the node returns to Running state. The hardware node operation is terminated by the off operation and enters Stop state.

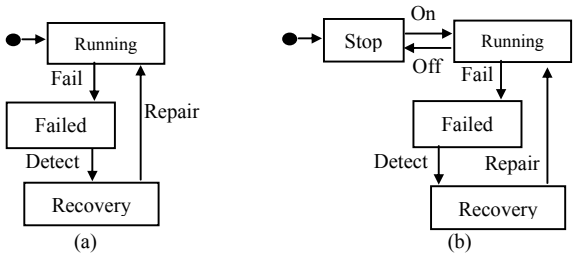


Figure 4. (a) STM of Software Process (b) STM of Hardware component

IV. DEPLOYMENT DIAGRAM & STATING RELATION BETWEEN SYSTEM & SERVICE COMPONENT

We model the system as collection of N interconnected nodes. Our objective is to find a deployment mapping for this execution environment for a set of service components C available for deployment that comprises the service. Deployment mapping can be defined as M: C $\rightarrow$ N between a numbers of service components instances C, onto nodes N. We consider four types of requirements in the deployment problem. (1) Components have execution costs, (2) collaborations have communication costs and (3) costs for running of background process known as overhead cost and (4) some of the components can be restricted in the deployment mapping to specific nodes which are called bound components. We observe the processing cost that nodes impose while host the components and also the target balancing of cost among the nodes available in the network. Communication costs are considered if collaboration between two components happens remotely, i.e., it happens between two nodes [6]. 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 components is always considerable no matter whether the components 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 service. We then define the objective of the deployment logic as obtaining an efficient (low-cost, if possible optimum) mapping of component onto the nodes

 ISBN: 978-1-61208-186-1

that satisfies the requirements in reasonable time. The deployment logic providing optimal deployment architecture is guided by the cost function F (M). The cost function is designed here to reflect the goal of balancing the execution cost and minimizing the communications cost [6]. This is in turn utilized to achieve reduced task turnaround time by maximizing the utilization of resources while minimizing any communication between processing node. That will offer a high system throughput, taking into account the expected execution and inter-node communication requirements of the service components on the given hardware architectures which is already highlighted in [13]. 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 cost ($\widehat{l}_n$, n = 1…N) of a given deployment mapping at every node. We will strive for an optimal solution of equally distributed cost among the processing nodes and the lowest cost possible, while taking into account the execution cost fc_i, i = 1….E, communication cost fkj, 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 cost can be calculated as $\sum_{i=1}^{|E|} fc_i$. This way, the logic can be aware of the target cost T [6]:

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

To cater for the communication cost f_{kj}, of the collaboration k_j in the service, the function q_0 (M, c) is defined first [16]:

$$q_0 (M, c) = \{n \in N \mid \exists (c \rightarrow n) \in M\} \quad (2)$$

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 f_{kj} 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 $I (q_0 (M, c_1) \neq q_0 (M, c_2)) = 1$, 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 [16]

$$F_k (M) = \sum_{j=1}^{|k|} I (q_0 (M, K_{j,1}) \neq q0 (M, K_{j,2})). f_{kj} \quad (3)$$

Given a mapping M = $\{m_n\}$ (where m_n is the set of components at node n & $n \in N$) the total cost can be obtained as $\widehat{l}_n = \sum_{c_i \in m_n} fc_i$. Furthermore the overall cost function F (M) becomes [16]:

$$F (M) = \sum_{n=1}^{|N|} | \widehat{l}_n - T | + F_k (M) + \sum_{j=1}^{|K|} f_{Bj} \quad (4)$$

V. ANNOTATION

To annotate the UML diagram the stereotype *saStep, computingResource, scheduler, QoSDimension* and the tag value *execTime, deadline, mean-time-to-repair, mean-time-between-failures* and *schedPolicy* are used according to the *UML profile for MARTE* and *UML Profile for Modeling Quality of Service & Fault Tolerance Characteristics* [7],[12]. *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*. The ResourceBroker is a kind of resource that is responsible for allocation and de-allocation of a set of resource instances (or their services) to clients according to a specific access control policy [7]. *QoSDimension* provides support for the quantification of QoS characteristics and attributes *mean-time-to-repair* and *mean-time-between-failures* [12]. We also introduce a new stereotype <<*transition*>> and three tag values *mean-time-to-stop, mean-time-to-start* and *mean-time-to-failure-detect*. <<*transition*>> induces a state transition of a scenario. *mean-time-to-stop* defines the mean time required to stop working of a hardware instance, *mean-time-to-start* states the time required to start working of a hardware instance, *mean- time-to-failure-detect* defines the mean time required to detect failures in the system.

VI. MODEL TRANSLATION

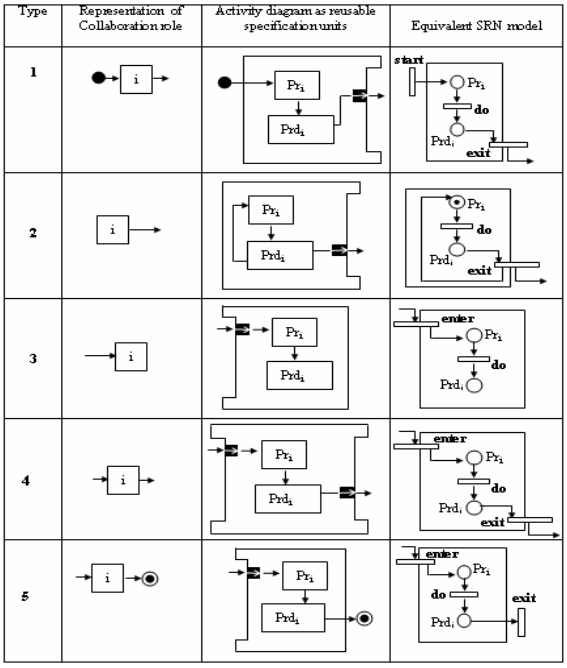
This section highlights the rules for the model translation from various UML models to SRN model. Since all the models will be translated into SRN we will give a brief introduction about SRN model. SRN is based on the Generalized Stochastic Petri net (GSPN) [3] and extends them further by introducing prominent extensions such as guard function, reward function and marking dependent firing rate [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 [5]. 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. SRN model has the following elements: Finite set

 ISBN: 978-1-61208-186-1

of the places (drawn as circles), Finite set of the transitions defined as either timed transition (drawn as thick transparent bar) or immediate transition (drawn as thick black bar), set of arcs connecting places and transition, multiplicity associated with the arcs, marking that denotes the number of token in each place.

Before introducing the translation rules different types of collaboration roles as reusable basic building block are demonstrated with the corresponding SRN model in Table I that can be utilized to form the collaborative building blocks.

TABLE I. SPECIFICATION OF REUSABLE UNITES AND EQUIVALENT SRN MODEL



The rules are the following:

Rule1: The SRN model of a collaboration (Fig. 5), where collaboration connects only two collaboration roles, is formed by combining the basic building blocks type 2 and type 3 from Table I. Transition **t** in the SRN model is only realized by the overhead cost if service components A & B deploy on the same physical node as in this case communication cost = 0, otherwise **t** is realized by both the communication & overhead cost.

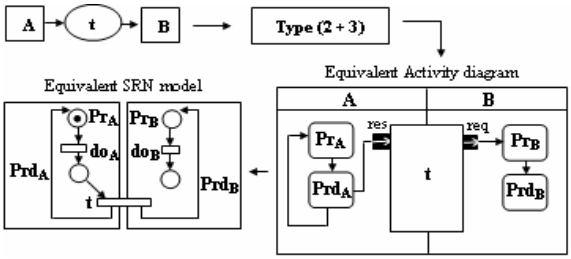


Figure 5. Graphical representation of rule 1

In the same way, SRN model of the collaboration can be demonstrated where the starting of the execution of the SRN model of collaboration role *A* depends on the receiving token from external source.

Rule 2: For a composite structure, when 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, the SRN model is formed by the utilizing the basic building block of Table I which is shown in Fig. 6.In the first diagram in Fig. 6, if component A contains its own token equivalent SRN model of the collaboration role A will be formed using basic building block type 1 from Table I. The same applies to the component B and C in the second diagram in Fig. 6.

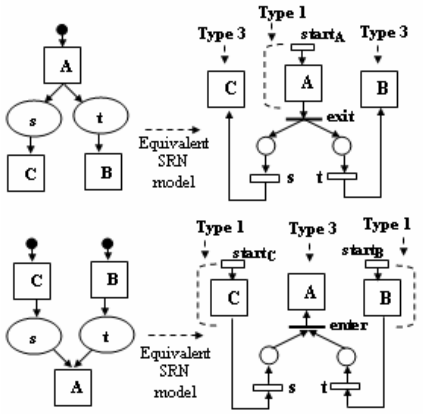


Figure 6. Graphical representation of rule 2

STM can be translated into a SRN model by converting each state into place and each transition into a timed transition with input/output arcs which is reflected in the transformation Rules 3.

Rule 3: Rule 3 demonstrates the equivalent SRN model of the STM of hardware and software components which are shown in the Fig. 7.

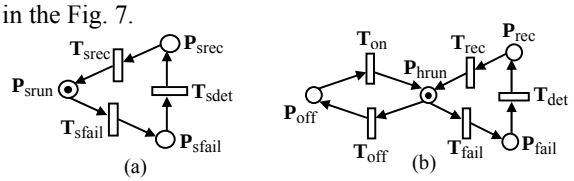


Figure 7. (a) SRN of Software process (b) SRN of hardware component

VII. MODEL SYNCHRONIZATION

The model synchronization is achieved hierarchically. Performance SRN is dependent on the Dependability SRN. Transitions in dependability SRN may change the behavior of the performance SRN. Moreover transitions in the SRN model for the software process also depend on the transitions in the SRN model of the hardware component. These dependencies in the SRN models are handled by the model synchronization by incorporating the guard functions [5].

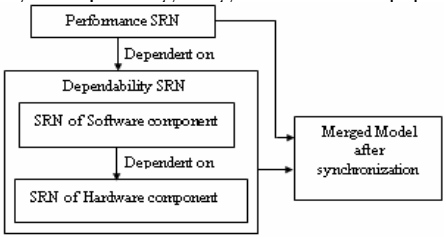


Figure 8: Model synchronization hierarchy

The model synchronization is focused in details here:

Synchronization between the Dependability SRN models in the dependability modeling layer: SRN model for the software process (Fig. 7(a)) is expanded by incorporating one additional place P_{hf}, three immediate transitions t_{hf}, t_{hfl}, t_{hfr} and one timed transition T_{recv} to synchronize the transitions in the SRN model for the software process with the SRN model for the hardware component. The expanded SRN model (Fig. 9(a)) is

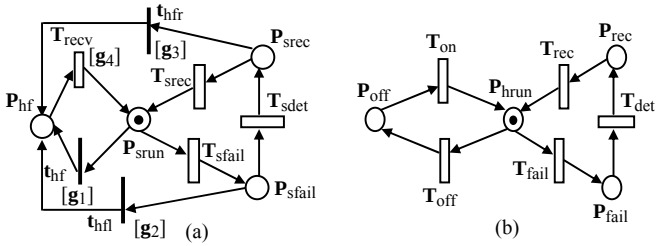


Figure 9. (a) Synchronized transition in the SRN model of the software process with the (b) SRN model of the hardware component

associated with four additional arcs such as $(P_{sfail} \times t_{hfl}) \cup (t_{hfl} \times P_{hf})$, $(P_{srec} \times t_{hfr}) \cup (t_{hfr} \times P_{hf})$, $(P_{srun} \times t_{hf}) \cup (t_{hf} \times P_{hf})$ and $(P_{hf} \times T_{recv}) \cup (T_{recv} \times P_{srun})$. The immediate transitions t_{hf}, t_{hfl}, t_{hfr} will be enabled only when the hardware node (in Fig. 9 (b)) fails as failure of hardware node will stop the operation of software process. The timed transition T_{recv} will be enabled only when the hardware node will again start working after being recovered from failure. Four guard functions g_1, g_2, g_3, g_4 allow the four additional transitions t_{hf}, t_{hfl}, t_{hfr} and T_{recv} of software process to work consistently with the change of states of the hardware node. The guard functions definitions are given in the Table III.

Synchronization between the dependability SRN & performance SRN: To synchronize the collaboration role activity, performance SRN model is expanded by incorporating one additional place P_{fl} and one immediate transition f_A shown in Fig. 10. After being deployed when collaboration role "A" starts execution a checking will be performed to examine whether both software and hardware components are running or not. If both the components work the timed transition do_A will fire which represents the continuation of the execution of the collaboration role "A". But if software resp. hardware components fail the immediate transition f_A will be fired which represents the quitting of the operation of collaboration role "A". Guard function gr_A allows the immediate transition f_A to work consistently with the change of states of the software and hardware components.

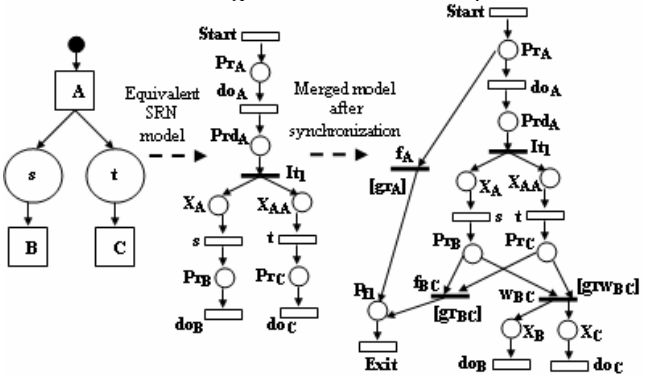


Figure 10. Synchronize the performance SRN model with dependability SRN

Performance SRN model of parallel execution of collaboration roles are expanded by incorporating one additional place P_{fl} and immediate transitions f_{BC}, W_{BC} shown in Fig. 10. In our discussion, during the synchronization of the parallel processes it needs to ensure that failure of one process eventually stop providing service to the users. This could be achieved by immediate transition f_{BC}. If software resp. hardware components (Fig. 9) fail immediate transition f_A will be fired which symbolizes the quitting of the operation of both parallel processes "B" and "C" rather than stopping either process "B" or "C", thus postponing the execution of the service. Stopping only either the process "B" or "C" will result inconsistent execution of the whole SRN and produce erroneous result. If both the software and hardware components work fine the timed transition W_{BC} will fire to continue the execution of parallel processes "B" and "C". Guard functions gr_{BC}, grw_{BC} allow the immediate transition f_{BC}, W_{BC} to work consistently with the change of the states of the software and hardware components. The guard function definitions are shown in the Table III.

VIII. HIERARCHICAL MODEL FOR MTTF CALCULATION

It is very demanding and not efficient with respect to execution time to consider behavior of all the hardware components during the SRN model generation. SRN model becomes very cumbersome and inefficient to execute. To solve the problem, we evaluate the MTTF (Mean time to failure) of system using the hierarchical model in which a fault tree is used to represent the MTTF of the system by considering MTTF of every hardware component in the system. Later we consider this MTTF of the system in our dependability SRN model for hardware components (Fig. 7(b)) rather than considering failure behavior of all the hardware components individually. The below Fig. 11 introduces one example scenario of capturing failure behavior of the hardware components using fault tree where system is composed of different hardware devices such as one CPU, two memory interfaces, one storage device and one cooler. The system will work when CPU, one of the memory interfaces, storage device and cooler will run. Failure of both memory interfaces or failure of either CPU or storage device or cooler will result the system unavailability.

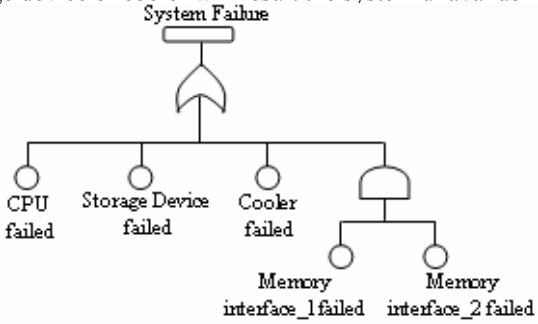


Figure 11. Fault tree model of System Failure

IX. CASE STUDY

As a representative example, we consider the scenario dealing with heuristically clustering of modules and assignment of clusters to nodes [16]. This scenario is sufficiently complex to show the applicability of our proposed framework. The problem is defined in our approach as collaboration of $E = 10$ service components or collaboration role (labeled $C_1 \ldots C_{10}$) to be deployed and $K = 14$ collaborations between them depicted in Fig. 12. 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. In this scenario, new service is generated by integrating and combining the existing service components that will be delivered conveniently by the system. For example, one new service is composed by combining the service components C_1, C_7, C_6, C_8, C_9 shown in Fig. 12 as thick dashed line. The internal behavior of the collaboration K_i is realized by the call behavior actions through UML activity like structure already demonstrated in Fig. 3(b). The composition of the collaboration role C_i of the delivered service by the system is demonstrated in Fig. 14. 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_1 where the semantics of the activity is realized by the token flow. After completion of the processing of the component C_1 the response is divided into two flows which are shown by the fork node f_1. The flows are activated towards component C_7 and C_6. After getting the response from the component C_1, processing of the components C_7 and C_6 will be started. The response and request are mentioned by the streaming pin *res* and *req*. The processing of the Component C_8 will be started after getting the responses from both component C_7 and C_6 which is realized by the join node j_8. After completion of the processing of component C_8 component C_9 starts its processing and later on activity is terminated which is mentioned by the end node (◉). In this example, the target environment consists of $N = 3$ identical, interconnected nodes with no failure of network link, with a single provided property, namely processing power, and with infinite communication capacities depicted in Fig. 13. The optimal deployment mapping can be observed in Table II. The lowest possible deployment cost, according to equation (4) is: 17 + 100 + 70 = 187.

To annotate the UML diagrams in Fig. 13 & 15 we use the stereotypes *<<saStep>>* *<<computingResource>>*, *<<scheduler>>* and the tag values *execTime*, *deadline* and *schedPolicy* which are already explained in section 5. Collaboration K_i (Fig. 15) is associated with two instances of *deadline* as collaborations in example scenario are associated with two kinds of cost: communication cost & cost for running background process (BP). To annotate the STM UML diagram of software process (shown in Fig. 14) we use the stereotype *<<QoSDimension>>*, *<<transition>>* and attributes *mean-time-between-failures*, *mean-time-to-failure detect* and *mean-time-to-repair* already mentioned in section 5. Annotation of the STM of hardware component can be demonstrated in the same way as STM of software process.

By considering the deployment mapping and the transformation rules the analogous SRN model of our example service (in Fig. 15) is depicted in Fig. 16. In our discussion, we consider M/M/1/n queuing system so that at most n jobs can be in the system at a time [3]. For generating the SRN model, firstly we will consider the starting node (●). According to rule 1, it is represented by timed transition

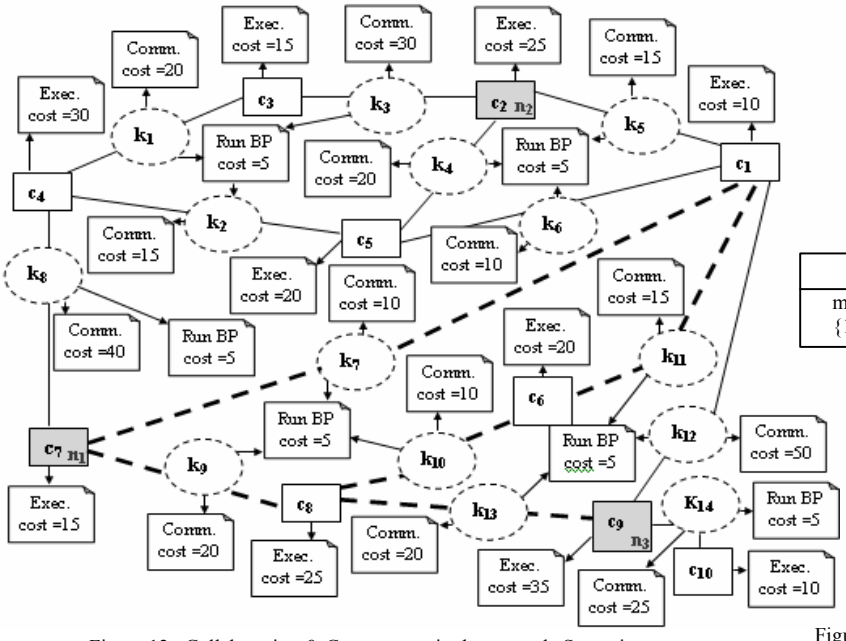


Figure 12. Collaboration & Components in the example Scenario

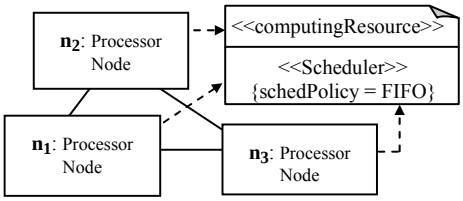


Figure 13. The target network of hosts

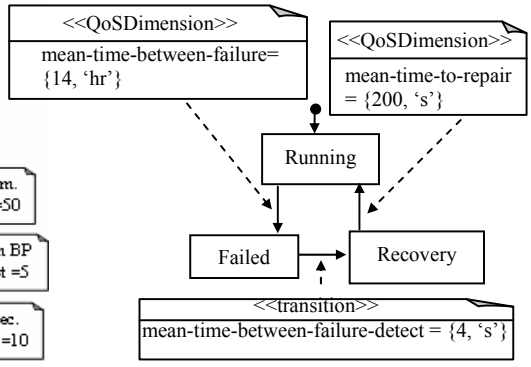


Figure 14. Annotated STM diagram of software component

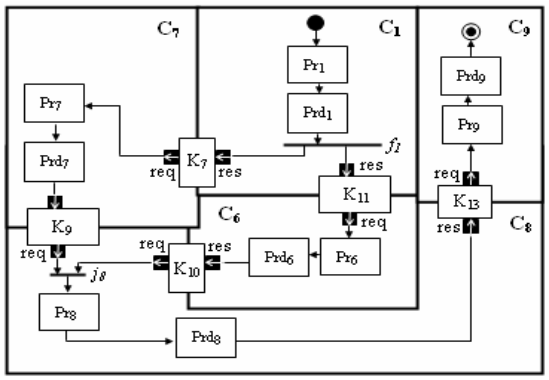


Figure 15. Service composition & Detail behavior of the event of the Collaboration using activity

(denoted as start) and the arc connected to place Pr_1 (states of component C_1). When a token is deposited in place Pr_1, immediately a checking is done about the availability of both software and hardware components by inspecting the corresponding SRN models (Fig. 9). The availability of software and hardware component allow the firing of timed transition t_1 mentioning the continuation of the further execution. Otherwise immediate transition f_1 will be fired mentioning the ending of the further execution because of software resp. hardware component failure. The enabling of immediate transition f_1 is realized by the guard function $\boldsymbol{gr_1}$. After the completion of the state transition from Pr_1 to Prd_1 (states of component C_1) the flow is divided into two branches (denoted by the immediate transition It_1) according to rule 2. The token will be deposited to place Pr_7 (states of component C_7) and Pr_6 (states of component C_6) after the firing of transitions K_7 and K_{11}. The collaboration 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 $\boldsymbol{n_1}$. According to rule 1, collaboration K_{11} is realized only by the cost for running background process as C_1 and C_6 deploy on the same processor node $\boldsymbol{n_3}$. When a token is deposited into place Pr_7 and Pr_6, immediately a checking is done about the availability of both software and hardware components by inspecting the corresponding dependability SRN model (Fig. 9). The availability of software and hardware components allow the firing of immediate transition w_{76} which eventually enables the firing of timed transition t_7 and t_6 mentioning the continuation of

TABLE II. OPTIMAL DEPLOYMENT MAPPING

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

the further execution. The enabling of immediate transition w_{76} is realized by the guard function $\boldsymbol{grw_{76}}$. Otherwise immediate transition f_{76} will be fired mentioning the ending of the further execution because of failure of software resp. hardware component. The enabling of immediate transition f_{76} is realized by the guard function $\boldsymbol{gr_{76}}$. After the completion of the state transition from Pr_7 to Prd_7 (states of component C_7) and from Pr_6 to Prd_6 (states of component C_6) component C_8 starts processing. The merging of result is realized by the immediate transition It_2 after the firing of transitions K_9 and K_{10}. Collaboration K_9 is realized only by the cost for running background process as C_7 and C_8 deploy on the same processor node $\boldsymbol{n_1}$. K_{10} is translated by the timed transition which 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 $\boldsymbol{n_1}$. When a token is deposited in place Pr_8, immediately a checking is done about the availability of both software and hardware components by inspecting the corresponding SRN model (Fig. 9). The availability of software and hardware components allow the firing of timed transition t_8 mentioning the continuation of the further execution. Otherwise immediate transition f_8 will be fired mentioning the ending of the further execution because of software resp. hardware component failure. The enabling of immediate transition f_8 is realized by the guard function $\boldsymbol{gr_8}$. After the completion of the state transition from Pr_8 to Prd_8 (states of component C_8) the token is passed to place Pr_9 by firing of timed transition K_{13}. K_{13} is realized by both communication cost and cost for running background process as C_8 and C_9 deploy on the two different nodes $\boldsymbol{n_1}$ and $\boldsymbol{n_3}$. When a token is deposited in place Pr_9, immediately a checking is done about the availability of both software and hardware components by inspecting the corresponding SRN model (Fig. 9). The availability of software and hardware component allow the firing of timed transition t_9 mentioning the continuation of the further execution. Otherwise immediate transition f_9 will be fired mentioning the ending of the further execution because of software resp. hardware component failure and the ending of the execution of the SRN model is realized by the timed transition $Exit_2$. The enabling of immediate transition f_9 is realized by the guard function $\boldsymbol{gr_9}$. After the completion of the state transition from Pr_9 to Prd_9 (states of component C_9) the ending of the execution of the SRN model is realized by the timed transition $Exit_1$. The definition of guard functions are shown in Table III (Phrun & Psrun are shown in Fig. 9).

TABLE III. GUARD FUNTIONS DEFINITION

Function	Definition
g_1, g_2, g_3	if (# Phrun = = 0) 1 else 0
g_4	if (# Phrun = = 1) 1 else 0
gr_A, gr_{BC}, gr_1, gr_{76}, gr_8, gr_9	if (# Psrun = = 0) 1 else 0
grw_{BC},grw_{76}	if (# Psrun = = 1) 1 else 0

We use SHARPE [15] to execute the obtained model and calculate the system's throughput. The throughput of successful jobs can be computed by checking the throughput of the transition $Exit_1$ by SHARPE [15]. The throughput result is summarized in Tab. IV and graph in Fig. 17 shows throughput variation of the system against the change of failure rate of both hardware and software components.

 ISBN: 978-1-61208-186-1

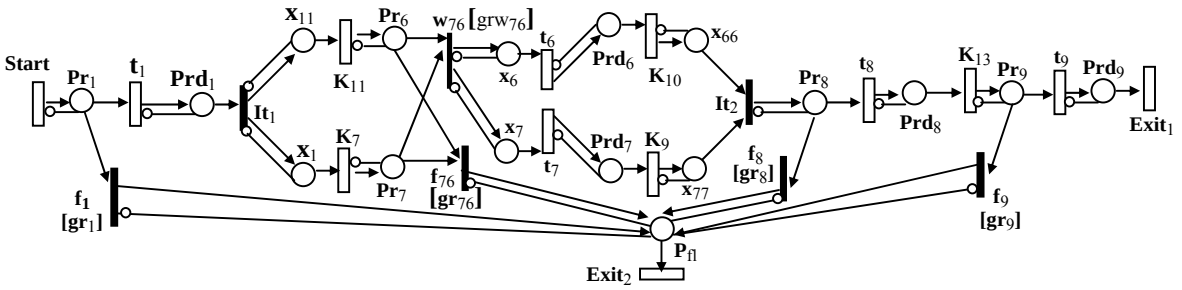


Figure 16. Equivalent SRN model of the example service

X. CONCLUSION AND FUTURE WORK

We presented a novel approach for model based performability evaluation of a distributed system which spans from system's dynamics demonstration and capturing behavior of system components through UML diagram as reusable building blocks to efficient deployment of service components in a distributed manner by focusing the QoS requirements. We put emphasis to establish some important concerns relating specification and solution of performability models emphasizing the analysis of the system's dynamics. We design the framework in a hierarchical and modular way which has the advantage to introduce any modification or adjustment at a specific layer in a particular submodel rather than in the combined model according to any change in the specification. Among the important issues that come up in our development is flexibility of capturing the system's dynamics using our new reusable specification of building blocks and ease of understanding the intricacy of combined model generation and evaluation from that specification by proposing transformation from UML diagram to corresponding SRN elements like states, different pseudostates and transitions. However, our eventual goal is to develop support for runtime redeployment of components, this way keeping the service within an allowed region of parameters defined by the requirements. As a result, with our proposed framework we can show that 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 systems. Further work includes tackling the state explosion problems of reachability marking of large distributed systems.

TABLE IV. THROUGHPUT CALCULATION

	Throughput
Performability model	0.0095
Pure performance model	0.01385

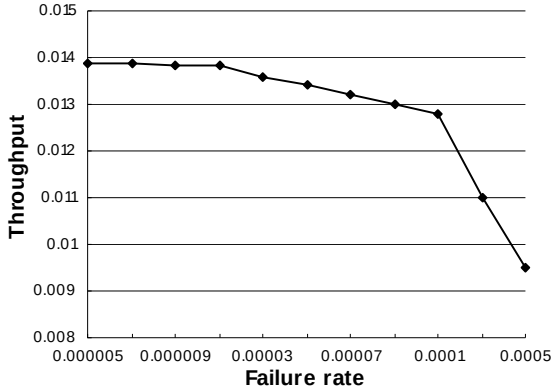


Figure 17. Numerical result of our example scenario

REFERENCES

[1] F. A. Jawad and E. Johnsen, "Performability: the vital evaluation method for degradable systems and its most commonly used modeling method, Markov reward modeling", http://www.doc.ic.ac.uk/~nd/surprise_95/journal/vol4/eaj2/report.html, <retrieved May 2011>

[2] E. de Souza e. Silva and H. R. Gali,"Performability analysis of computer systems: from model specification to solution", Performance evaluation 14, pp. 157-196, 1992

[3] K. S. Trivedi, "Probability and Statistics with Reliability, Queuing and Computer Science application", Wiley-Interscience publication, ISBN 0-471-33341-7, 2001

[4] OMG UML Superstructure, Version-2.2

[5] G. Ciardo, J. Muppala, and K. S. Trivedi, "Analyzing concurrent and fault-tolerant software using stochastic reward nets", Journal of Parallel and Distributed Computing, Vol. 15, 1992

[6] M. Csorba, P. Heegaard, and P. Herrmann, "Cost-Efficient Deployment of Collaborating Components", DAIS, pp. 253–268, Springer, 2008

[7] OMG 2009, "UML Profile for MARTE: Modeling & Analysis of Real-Time Embedded Systems", V – 1.0

[8] N. Sato and Trivedi, "Stochastic Modeling of Composite Web Services for Closed-Form Analysis of Their Performance and Reliability Bottlenecks", ICSOC, pp. 107-118, Springer, 2007

[9] P. Bracchi, B. Cukic, and Cortellesa, "Performability modeling of mobile software systems", ISSRE, pp. 77-84, 2004

[10] N. D. Wet and P. Kritzinger, "Towards Model-Based Communication Protocol Performability Analysis with UML 2.0", http://pubs.cs.uct.ac.za/archive/00000150/01/No_10, <retrieved May 2011>

[11] Gonczy, Deri, and Varro, "Model Driven Performability Analysis of Service Configurations with Reliable Messaging", MDWE, 2008

[12] OMG 2009, "UML Profile for Modeling Quality of Service & Fault Tolerance Characteristics Specification", V-1.1

[13] R. H. Khan and P. Heegaard, "A Performance modeling framework incorporating cost efficient deployment of multiple collaborating components" ICSECS, pp. 31-45, Springer, 2011

[14] F. A. Kramer, "ARCTIS", Department of Telematics, NTNU, http://arctis.item.ntnu.no, <retrieved May 2011>

[15] K. S. Trivedi and R. Sahner, "Symbolic Hierarchical Automated Reliability / Performance Evaluator (SHARPE)", Duke University, NC, 2002

[16] Mate J. Csorba, "Cost efficient deployment of distributed software services", PhD Thesis, NTNU, Norway, 2011

 ISBN: 978-1-61208-186-1