Chapter 11

Performance Evaluation of Distributed System Using SPN

Razib Hayat Khan, Poul E. Heegaard and Kazi Wali Ullah

Additional information is available at the end of the chapter

http://dx.doi.org/10.5772/48813

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 endeavour. 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 plays an important role in the whole design process of the system for qualitative and quantitative analysis. However, in a distributed system, system functional 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 indispensable to capture the functional behavior of the distributed objects for appropriate analysis to evaluate the performance related factors of the overall system. We therefore adopt UML collaboration and activity oriented approach as UML is the most widely used modeling language which models both the system requirements and qualitative behavior through different notations. 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 on, this UML specification style is applied to generate the SPN model by our performance modeling framework. UML collaboration and activity provides a tremendous modeling framework containing several interesting properties. Firstly, collaborations and activity model the concept of service provided by the system very nicely. They define structure of partial object behavior, 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 and role bindings [12].

Considering system execution architecture to specify the deployment of the service components is realized by the UML deployment diagram. Abstract view of the system

execution architecture captured by the UML deployment diagram defines the execution architecture of the system by identifying the system components and the assignment of software artifacts to those identified system components. 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 identify and deploy the service components that require the highest volume of interaction onto the same resource or to choose resources that are connected by the links with sufficiently high capacity [12].

It is indispensable to extend the UML model to incorporate the performance-related quality of service (QoS) information to allow modeling and evaluating the properties of a system like throughput, utilization and mean response time. So the UML models are annotated according to the *UML profile for MARTE: Modeling & Analysis of Real-Time Embedded Systems* to include quantitative system parameters [1]. Thus, it helps to maintain consistency between system design and implementation with respect to requirement specifications.

Markov models, stochastic process algebras, SPN (Stochastic Petri Net) are popular and much studied analytical approaches to conduct performance modeling and evaluation. Among all of them, we will focus on the SPN as the performance model generated by our framework due to its increasingly popular formalisms for describing and analyzing systems, its modeling generality, its ability to capture complex system behavior concisely, its ability to preserve the original architecture of the system, to allow marking dependency firing rates & reward rates defined at the net level, to facilitate any modification according to the feedback from performance evaluation and above all, the existence of analysis tools.

Several approaches have been followed to generate the performance model from system design specification. Lopez-Grao *et al.*, described a conversion method from annotated UML activity diagram to stochastic petrinet model [2]. 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 [3]. 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 [4]. Trowitzsch and Zimmermann proposed the modeling of technical systems and their behavior by means of UML and for the resulting models, a transformation into a Stochastic Petri Net was established [13]. Abdullatif and Pooly presented a method for providing computer support for extracting Markov chains from a performance annotated UML sequence diagram [14]. However, most existing approaches do not highlight more on the

issue of how to optimally conduct the system modeling and performance evaluation. The approach presented here is the first known attempt that introduces a new specification style utilizing UML behavioral diagrams as reusable specification building block which is later on 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 approach 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 framework. The novelty of our approach also reflected in showing the optimality of our solution with respect to both deployment logic and evaluation of performance metrics.

The objective of the chapter 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 solves the model for relevant performance metrics to demonstrate performance prediction results at early stage of the system development life cycle. To incorporate the cost function to draw relation between service component and available physical resources permit us to identify an efficient deployment mapping in a fully distributed manner. The work presented here is the extension of our previous work described in [5, 6, 7, 12] where we present our framework with respect to the execution of single and multiple collaborative sessions and to consider alternatives system architecture candidates to describe system functional behavior and later on to evaluate the performance factors. The chapter is organized as follows: section 2 describes the performance evaluation of distributed system where the requirements of the successful performance evaluation are mentioned, section 3 introduces our performance modeling framework in details by considering the requirements outlined in the previous section, section 4 shows the applicability of our performance modeling framework with respect to performance modeling of a distributed system, and section 5 mentions the concluding remarks with future directions.

2. Performance evaluation of distributed software system

Performance evaluation is an integral part of any distributed software system which gives an indication of whether the system will meet non functional properties, once system built. The evaluation can be done in one of the two stages of the software development process:

1. Evaluation can be conducted at the early stage of the software development process
2. Evaluation can be done when the development process is completed.

Conducting the performance evaluation in any of the two stages has some merits and demerits. Early assessment of performance evaluation allows system designer predicting the system response in order to meet the non functional requirements before the system being built. This in turn guides the system designer about the system development process in right manner which thus increases the productivity and quality in accordance with the reduction in cost. But conducting performance evaluation in the early stage of the software development process is challenging because of the absence of the real system in hand. So predication in advance not always guides the system designer in right way. Modeling system functional behavior perfectly works as a catalyst to successfully conduct the system performance evaluation. System functional behavior is disseminated across several components that are physically distributed which increases the complexity in developing distributed software systems. Perfect modeling of distributed system functional behavior is realized by capturing the local behavior of the system components and also the interaction among them. It is very difficult to achieve these tasks in correct way when development of system is limited in the laboratory where modeling will be done by generating case study or scenario.

Conducting performance evaluation after the system development process being completed is less challenging than the former case. It is possible to retrieve the real system response in order to meet the system non functional requirements as the real system is already implemented. So the designer can get a real understanding about the correct status of the development process to know whether the system can meet the non functional requirements and end user's expectation. If the system fails to satisfy non functional requirements and can't meet the end user expectation, the only alternative is to rethink about the system design process. Any change in the system design process can cause the modification in the system development process. In worst case the development process might start from the beginning which in turn costs a lot.

In order to conduct the performance evaluation of distributed software system, the decision is not only influenced by when the evaluation should be performed but also other factors like which evaluation technique is appropriate and reasonable. There are mainly two evaluation techniques:

1. Simulation based evaluation
2. Analytic solution

Simulation based solution of the actual implementation gives a better assessment of the performance evaluation of the system. Simulation based solution gives the freedom to build the system arbitrary detailed and there is no restriction on building the simulation model of the real system [8]. Thus, it allows modeling and evaluating the system performance in a flexible way. But to develop the simulation model is not an easy task and sometimes it is error-prone. Implementing a complex system is usually a time-consuming, expensive task and needs experience [8]; mastering to handle this complexity is driven by the gaining vast

knowledge in simulation language and how to apply this language to build and present the logic behind the complex distributed system to capture system functional behavior properly for conducting performance evaluation.

Analytical solution is another way to conduct the performance evaluation of the complex distributed software system. Presence of well established mathematical formula for analytical methods makes it popular to the scientific community to obtain the performance evaluation of the systems. This method of finding solution is more acceptable than simulation based evaluation because of the direct applicability of the mathematical formula and the availability of evaluation tools. Another advantage of using analytic model is the rapid development of model for performance evaluation of large and multifaceted system using the formalisms of analytical methods. However, sometimes such analytical models can usually be constructed by placing some structural restrictions and assumptions on the original system model based on the explicit modeling formalism which has been selected; the reason is that analytical models have a limited expressiveness in some cases to capture the complex system behavior. While it is sometimes doable to simplify the model of the system in order to make it analytically tractable, there are many cases in which the significant aspects of the system can not be effectively represented into the analytical model for performance evaluation [8].

In this chapter we particularly focus on the performance evaluation of the distributed software system at the early stage of the system development process using analytical models. The requirements for performance evaluation of distributed software system are not only influenced by the question of when to conduct the evaluation and which method is appropriate for the obtaining performance results but also driven by the other requirements such as:

1. Need for an efficient approach that will help to model the system functional behavior in a way that can reflect real system behavior so that performance evaluation can be meaningful afterwards.
2. Deployment mapping is an integral part of the distributed software system development process which is defined by the assignment of software components in the physical resources that are distributed. For large and complex system it requires an efficient approach for handling the deployment mapping so that it can also ensure the efficiency with respect to performance evaluation.
3. Model that captures the system functional behavior will be used as an input model for developing the analytical model. So we need a mechanism that can also include the performance parameters to the input models for conducting the successful evaluation.
4. Need for a scalable and efficient approach to establish the correspondence between the input model that will be utilized to capture the system functional behavior and the output model that will be used to conduct the performance evaluation of the distributed software system.
5. At last, developing a tool based support for the whole process of performance evaluation considering above requirements which can ensure the rapid development, evaluation and user friendliness.

The following Figure 1 mentions the requirements or factors that we need to consider for the successful performance evaluation of the distributed software systems. In order to capture all the above mentioned factors, it needs an efficient approach or developing a framework that will allow rapid and successful performance evaluation of distributed software system which at the end reflects the aim of this chapter.

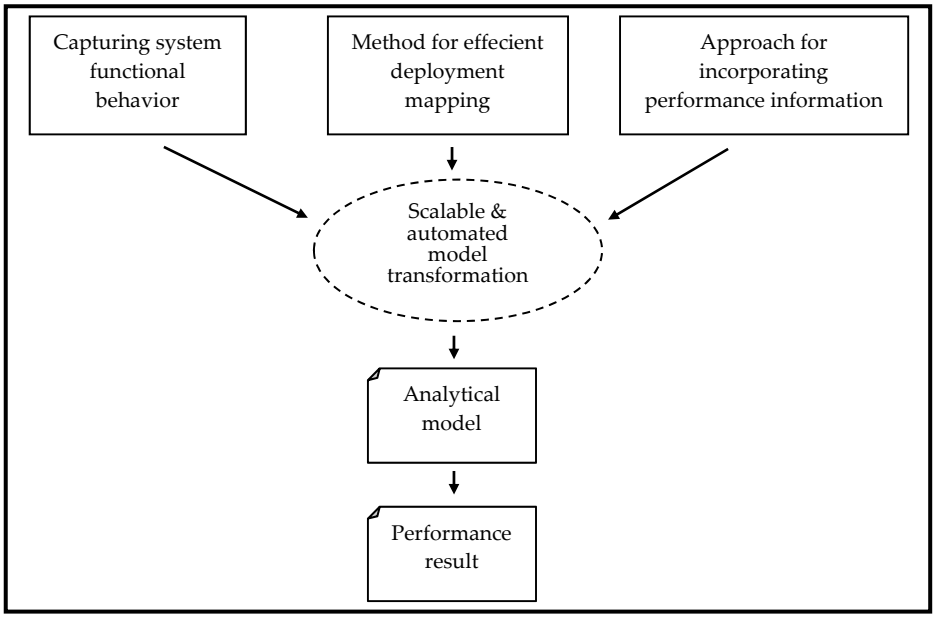

Figure 1. Performance modeling framework

3. Performance modeling framework

We already mentioned our main objective in the previous section that will be presented broadly in this section. In order to achieve the main objective, we need to follow an engineering approach that will accelerate the distributed software system development process. We also need to define the method that will be accounted for evaluating the system performance. We limit ourselves to methods targeting system development process using the standards of UML (Unified Modeling Languages) [9]. In the evaluation side we limit ourselves to methods that will analytically solve our problem using the technique SPN (Stochastic Petri Nets). This section mainly presents these two main techniques and also focuses on their properties that will be utilized to design our performance modeling framework.

3.1. Capturing system functional behavior

We use UML collaboration as main specification unit to specify system functional behavior. The UML standard focuses in particular on the structural aspects of UML collaborations. UML does not, however, elaborate detailed semantics of the behavioral implications of the

structural composition. Collaborations are intended as a context in which behaviors may be defined. Compared to the other uses of collaborations, and what we need, this is an obvious shortcoming. We will later see how a combination of collaborations with activities may solve this problem [9].

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 [11]. As a representative example, we introduce a scenario description utilizing UML collaboration 2. Several users equipped with cell phone or smart phone want to receive weather information of their current location using his/her hand held device. The user request is first transferred to authentication server through base transceiver station to ensure the authenticity of the user. Thereafter, the request of the legitimate user is transferred to the location server to retrieve the location information of the user. The location information is then transferred to weather server for retrieving the weather information according to the location of the user. Figure 2 defines this scenario as UML 2 collaboration. Participants in the system are users, mobile terminals, base transceiver stations, authentication servers, location servers, weather servers which are represented by the collaboration roles user, MT, BTS, AuS, LS, and WS. The users are the part of the environment and therefore labeled as <<external>>.The default multiplicity of the users, mobile terminals, base transceiver stations, authentication servers, location servers, weather servers are one to many, which are denoted by (1..*). The interactions between the collaboration roles are represented by the collaboration such as mobile terminal and BTS interact through *t*: *transfer*, BTS and authentication server, location server, weather server interact successively through *a*: *authenticate, l: request location info, w: request weather info,* while the user interacts with the mobile terminal by collaboration *g*: *generate request* [6].

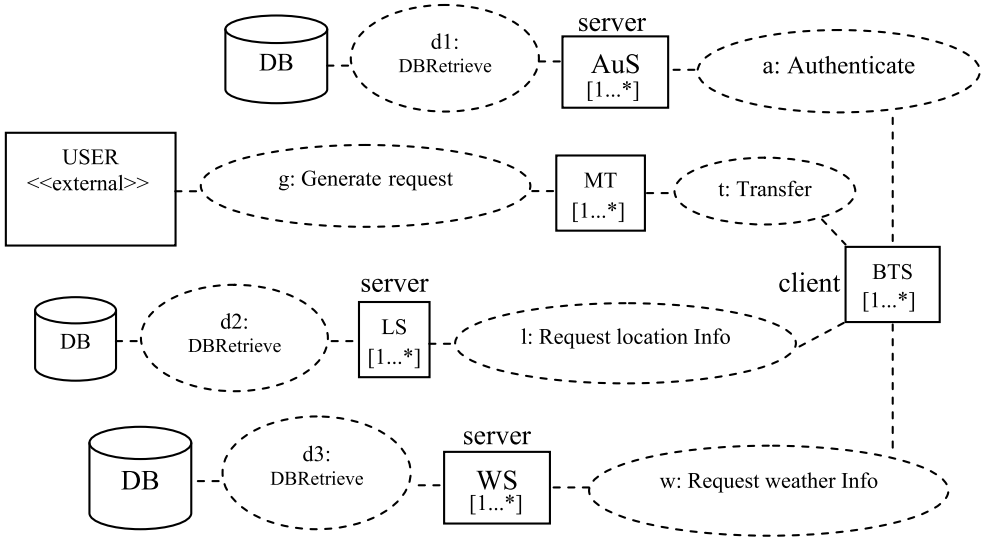

Figure 2. Collaboration diagram

The specifications for collaborations are given as coherent, self-contained reusable building blocks. The internal behavior of building block is described by the 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 [6]. For each collaboration, the activity declares a corresponding call behavior action refereeing to the activities of the employed building blocks. Depending on the number of participants, connectivity to other blocks and level of decomposition, we distinguish three different kinds of building blocks [10]:

1. The most general building block is collaboration with two participants providing functionality that is intended to be composed with other functionality. We refer to such a building block as service collaboration.
2. Building blocks that involve only local behavior of one participant are referred to as activity blocks. They are represented by activities.
3. A special building block is system collaboration, which is collaboration on the highest composition level. In contrast to a service, a system is closed and cannot be composed with other building blocks.

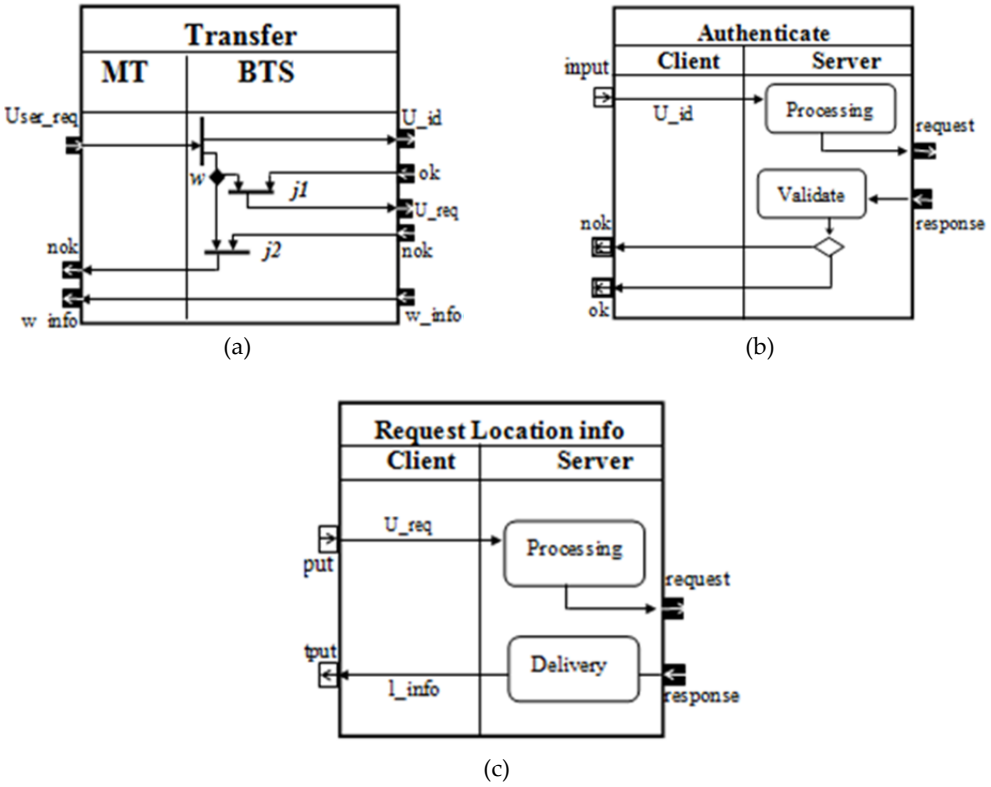

Figure 3. Activity diagram for expressing the internal behavior of collaboration

Hereby collaborations of Figure 2 are modeled by a call behavior action referring to the activity describing the behavior of the corresponding collaboration [10]. Activity diagram presents complete behavior in a quite compact form and may define connections to other behaviors via input and output pins [6]. Here we specify the behavior of one user request to show how the request is generated from his/her mobile terminal and served by the BTS, authentication server, location server and weather server and later on, compose this behavior to show how the requests will be processed by the BTS, authentication servers, location servers and weather servers so that the overall system behavior can be delineated. The activity *transfer* describes the behavior of the corresponding collaboration shown in Figure 3 (a). It has one partition for each collaboration role: mobile terminal (MT) and base transceiver station (BTS). Activities base their semantics on token flow [1]. The system starts by placing a token in the initial node of the mobile terminal when one request is generated by the user through his/her mobile terminal. The token is then transferred to the BTS where it moves through the fork node generating two flows. One flow places a token in the waiting decision node *w* which is the extension of a decision node with the difference that it may hold a token similar to an initial node, as defined in [1]. *w* is used in combination with join nodes *j1* and *j2* to explicitly model the acceptance or rejection of the user request based on the user authenticity. The other flow is forwarded as input to the authentication server to check whether the user is legitimate to generate service request. If the user is legitimate to generate the request a token is offered to the join node *j1*. If *w* still has its token *j1* can fire which emits a token which then forwarded to the location server for further processing. If the user is not legitimate to generate the request, a token is offered to the join node *j2*. If *w* still has its token *j2* can fire notifying the user upon the cancellation of request and then terminates the activity.

In order to validate the user identity (mobile number in this case) provided by a user who requests for service, BTS participates in the collaboration *authenticate* together with the authentication server. This is specified by collaboration *a: authenticate* where BTS plays the role of client and the authentication server plays the role of server. The behavior of the collaboration defined by the UML activity which is divided into two partitions, one for each collaboration role: client & server shown in Figure 3(b). The activity is started on the client side, when user id is provided as parameter *u_id* at the input pin. The input is then directly sent to the server, where it is converted into a database request in the call behavior action *processing*. Thereafter, it is the task of the collaboration between the server and the database to provide the stored user information. In order to get the information, the request leaves the activity *authenticate* and the server waits for the reception of the response. This is modeled with the input and output pins *request* and *response*. Depending on the validity of the user id, the server may decide to report *ok* or *nok* (not ok) to the client by the call behavior action *validate*. The result is then forwarded to the corresponding output pin in the client side and the activity is terminated. The semantics of all the pins are given in [12]. Likewise, we can describe the behavior of collaboration *l: Request Location info* (shown in Figure 3(c)) and w: *Request Weather info* through activity partition of client and server where BTS plays the role of client and location server and weather server play the role of server to deliver the requested information to the user through his/her mobile terminal.

The collaborative building blocks with help of activities specify overall system functional behavior which is introduced in Figure 4 for our scenario. For specifying detail behavior, 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 a call behavior action referring to the respective activity of the 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. There are different kinds of pins described as follows [10]:

1. Starting pins activate the building block, which is the precondition of any internal behavior.
2. Streaming pin may pass tokens throughout the active phase of the building block.
3. Terminating pins mark the end of the block's behavior. If collaboration is started and terminated via several alternative pins, they must belong to different parameter sets. This is visualized in UML diagram by an additional box around the corresponding node.

Figure 4 shows the activity diagram for our system to highlight the overall behavior of the system by composing all the building blocks. The initial node (·) marks the starting of the activity. The activity is started on the client side. When a user service request is generated via mobile terminal, *g: Generate request* will transfer the user service request as parameter *u_req* to the BTS via collaboration *t: Transfer*. Once the request arrived at the BTS the user id as parameter *u_id* is transferred to the authentication server to check whether the user is authentic to accept the service and the activity is represented by *a: authenticate*. The activity *authenticate* initiates a database request, modeled by collaboration *d1: DBRetrieve* and terminates with one of the alternative results *ok* or *nok*. After arriving the positive response at the BTS, request for location information is forwarded to the location server represented by activity *Request location info*. Location server makes a database request which is modeled by *d1: DBRetrieve* and terminates with result *l_info* (Location information). After getting the location information, request for weather information according to user current location is forwarded by the BTS to the weather server represented by activity *Request weather info*. Weather server makes a database request which is modeled by *d2: DBRetrieve* and terminates with result *w_info* (Weather information). After that, the final result is transferred to the user hand held device by BTS via activity *t: Transfer*. But if the user is failed to prove his/her identity then immediately a *nok* is sent to the user's hand held device.

So far, we introduced the system functional behavior with respect to specific example. Now we would like to introduce the specification in more generalized way. For example, the

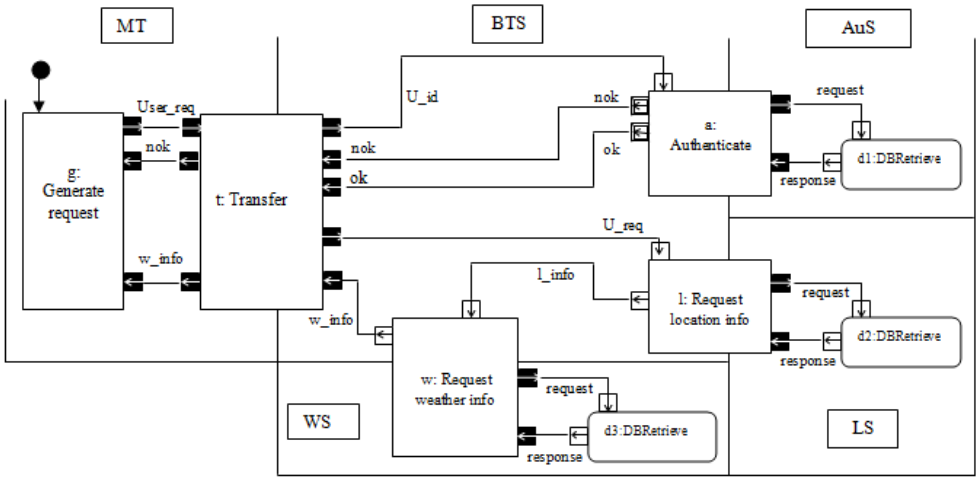

Figure 4. Activity diagram for detail system behavior

general structure of the building block *t* is given in Figure 5 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 and B)). The internal behavior of the same building block is shown in Figure 6(b). The activity $transfer_{ij}$ (*where ij = AB*) describes the behavior of the corresponding collaboration. It has one activity partition for each collaboration role: A and B. Activities base their semantics on token flow [1]. The activity starts by placing a token when there is a response (indicated by the streaming pin *res*) to transfer by either participant A or B. After completion of the processing by the collaboration role A and B the token is transferred from the participant A to participant B and from participant B to Participant A which is represented by the call behavior action *forward*.

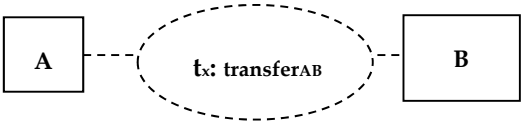

Figure 5. Collaboration diagram in generalized way

The detailed behavior of collaboration is given in following Figure 6(a). The initial node (·) indicates the starting of the activity. The activity is started at the same time from each participant A and B. After being activated, each participant starts its processing of the request which is mentioned by call behavior action P_i (Processing$_i$, where i = A, B). Completions of the processing by the participants are mentioned by the call behavior action d_i (Processing_done$_i$, i = A, B). After completion of the processing, the responses are delivered to the corresponding participants indicated by the streaming pin *res*. The response of the collaboration role A will be forwarded to B and vice versa which is mentioned by collaboration *t: transfer*$_{ij}$ (where ij = AB).

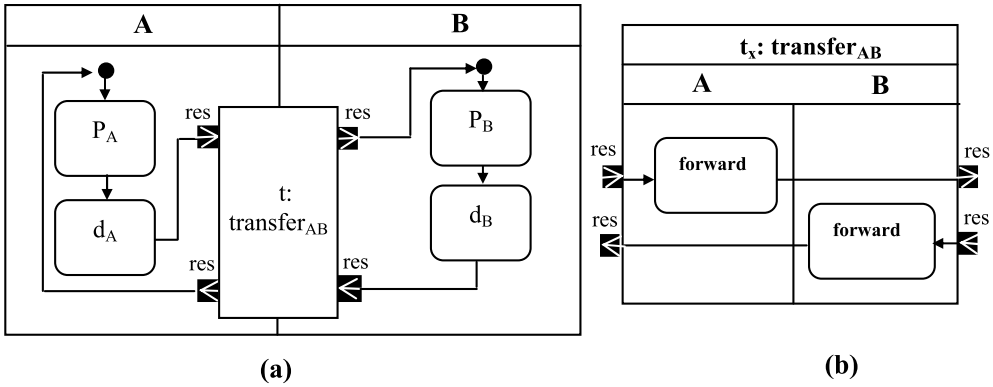

Figure 6. (a) Detail behavior of collaborative building block (b). Internal behavior of collaboration

3.2. Method for efficient Deployment mapping

Deployment diagram can be used to define the execution architecture of the system by identifying the system physical components and the assignment of software artifacts to those identified physical components [11]. After designing the deployment diagram the relation between system component and collaboration will be delineated to describe the service delivered by the system. The service is delivered by the joint behavior of the system components which may be physically distributed. The necessary partial behavior of the component used to realize the collaboration is represented by the collaboration role. In this way, it is possible to expose direct mapping between the collaboration roles to the system components to show the probable deployment of service components to the physical nodes of the system [6].

We consider two design alternatives of system architecture captured by UML deployment diagram to demonstrate the relationship between collaboration and system component for the scenario mentioned in Figure 2. For our defined scenario the identified system components by the 1st variation of deployment diagram are Mobile terminal, Base transceiver station, Authentication server, Location server and Weather server. After designing the deployment diagram the relationship between system component and collaboration will be delineated to describe the service delivered by the system. The service is delivered by the joint behavior of system components which may be physically distributed. The necessary partial behavior of the components used to realize the collaboration is represented by the collaboration role. Behavior of the components Mobile terminal, Base transceiver station, Authentication server, Location server, Weather server are represented by the collaboration roles MT, BTS, AuS, LS & WS to utilize the collaboration *t: transfer, a: authenticate, l: request location info, w: request weather info*. Here it is one to one mapping between system component & collaboration role shown in Figure 7(a).

We consider other variation of deployment diagram for mentioned scenario. In this variation of deployment diagram the identified system components are mobile terminal,

Base transceiver station, application server. In this case, the behavior of the components Mobile terminal and Base transceiver station is represented by the collaboration roles MT and BTS to utilize the collaboration *t: transfer* and the behavior of the component application behavior is represented jointly by the collaboration role AuS, LS and WS to utilize the collaboration *a: authenticate, l: request location info, w: request weather info*. In second case, the mapping between system component & collaboration role is generalized into one to many relations mentioned in Figure 7(b).

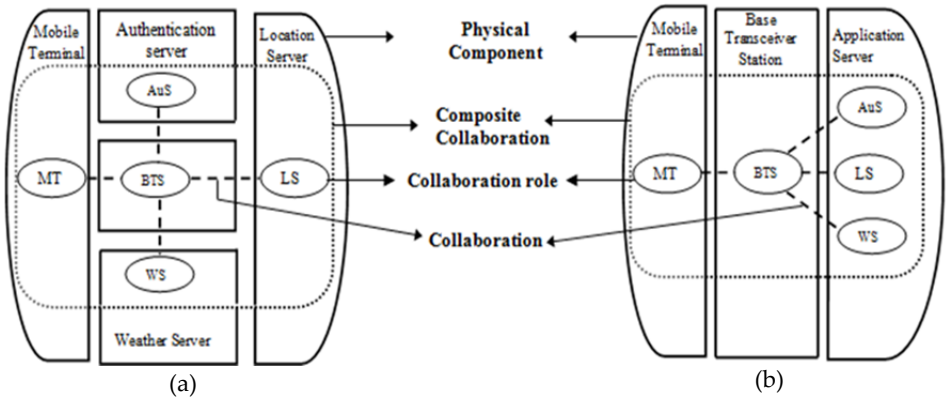

Figure 7. UML deployment diagram with service components deployment mapping

For large and complex system, conducting the deployment mapping is not straight forward like the previous cases. The deployment mapping has implication with respect to satisfying the non functional properties of the system. So we need for an approach that will be apposite for conducting the deployment mapping for complex system considering constraints and capabilities of the system components. We introduce our approach by considering 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 M can be defined as $(M : C \to N)$ between a numbers of service components instances c, onto nodes n mentioned in Figure 8. A components $c_i \in C$ can be a client process or a service process, while a node, $n \in N$ is a physical resource. Generally, nodes can have different responsibilities, such as providing services (*S1*), relaying traffic (*R1*), accommodating clients (*C1*), or a mixture of these (*SC1*). Components can communicate via a set of collaborations. We consider 3 types of requirements in the deployment problem where the term cost is introduced to capture several non-functional requirements those are later on utilized to conduct performance evaluation of the systems:

1. Components have execution costs
2. Collaborations have communication costs and costs for running of background process known as overhead cost
3. Some of the components can be restricted in the deployment mapping to specific nodes which are called bound components.

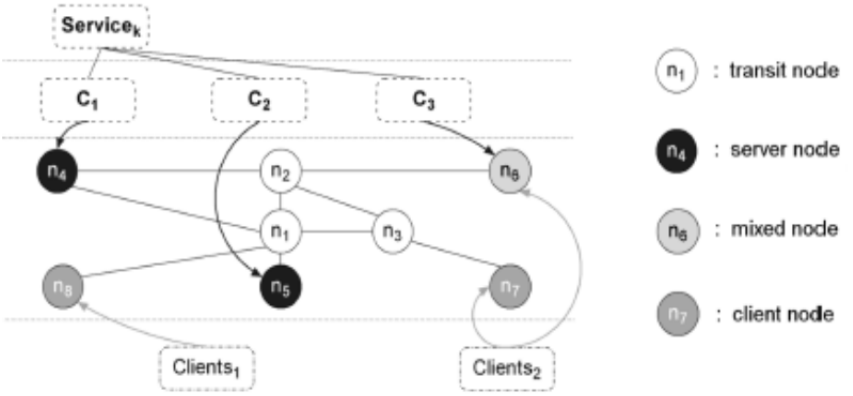

Figure 8. Service component mapping example

Furthermore, we consider identical nodes that are interconnected each other and are capable of hosting components with unlimited processing demand. 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 [15]. 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 components 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 cost function is designed here to reflect the goal of balancing the execution cost and minimizing the communications cost. This is in turn utilized to achieve reduced task turnaround time by maximizing the utilization of resources while minimizing any communication between processing nodes. 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 architecture. 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 collaborations between them labeled as k_j, (where j = 1 … K). The execution cost of each service component can be labeled as f_{Ci}; 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 cost ($\hat{l}_n$, n = 1… X) ofa

given deployment mapping at each node where X defines the total number of physical nodes available in the execution environment. 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 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 cost can be calculated as $\sum_{i=1}^{|E|} fc_i$. This way, the logic can be aware of the target cost T [15]:

$$T = \frac{1}{|X|}\sum_{i=1}^{|E|} fc_i \tag{1}$$

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 [15]:

$$q_0\left(M,c\right) = \{n \in N \mid \exists\left(c \to n\right) \in M\} \tag{2}$$

This means that $q_0(M,c)$ returns the node n that hosts components in the list mapping M. Let collaboration $k_j = \left(c_1, c_2\right)$. 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. In order 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 [15]:

$$F_k\left(M\right) = \sum_{j=1}^{|k|} I\left(q_0\left(M,K_{j,1}\right) \neq q_0\left(M,K_{j,2}\right)\right)\cdot fk_j \tag{3}$$

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 $\hat{l}_n = \sum_{c_i \in m_n} fc_i$. 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) [15]:

$$F(M) = \sum_{n=1}^{|N|} |\hat{l}_n - T| + F_k\left(M\right) + \sum_{j=1}^{|k|} f_{B_j} \tag{4}$$

3.3. Approach for incorporating performance information

UML is no doubt a well established language for modeling system functional behavior. But UML has lacking of incorporating non functional parameters in the model while specifying the functional behavior of any system. This needs for an approach or specification to incorporate the performance parameters in the UML for quantitative analysis. That's why we use a specification called the *UML profile for MARTE* for Modeling and Analysis of Real-Time and Embedded systems, provides support for specification, design, and verification/validation stages [1]. This new profile is intended to replace the existing UML Profile for Schedulability, Performance and Time. This specification of a UML profile adds

capabilities to UML for model-driven development of Real Time and Embedded Systems (RTES) [1].

MARTE defines foundations for model-based descriptions of real time and embedded systems. These core concepts are then refined for both modeling and analyzing concerns. Modeling parts provides support required from specification to detailed design of real-time and embedded characteristics of systems. MARTE concerns also model-based analysis. In this sense, the intent is not to define new techniques for analyzing real-time and embedded systems, but to support them. Hence, it provides facilities to annotate models with information required to perform specific analysis. Especially, MARTE focuses on performance and schedulability analysis. But it defines also a general analysis framework that intends to refine/specialize any other kind of analysis. Among others, the benefits of using this profile are thus [1]:

1. Providing a common way of modeling both hardware and software aspects of an RTES in order to improve communication between developers.
2. Enabling interoperability between development tools used for specification, design, verification, code generation, etc.
3. Fostering the construction of models that may be used to make quantitative predictions regarding real-time and embedded features of systems taking into account both hardware and software characteristics.

We apply several stereotypes of MARTE that permit us to map model elements into the semantics of an analysis domain such as schedulability, and give values for properties that are needed in order to carry out the analysis [1]. Specific tagged values are also applied. Tagged values are a kind of value slots associated with attributes of specific UML stereotypes [1]. In order to annotate the UML diagram we use several stereotypes and tag values according to the UML profile for MARTE. The stereotypes are the following [1]:

1. *saStep* is a kind of step that begins and ends when decisions about the allocation of system resources are made.
2. *ComputingResource* represents either virtual or physical processing devices capable of storing and executing program code. Hence its fundamental service is to compute.
3. *Scheduler* is defined a kind of ResourceBroker that brings access to its brokered ProcessingResource or resources following a certain scheduling policy mentioned by tag value *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 [1].

The tagged values are the following [1]:

1. *execTime*: The duration of the execution time is mentioned by the tagged value *execTime* which is the average time in our case. The execution cost of service component is expressed by this tagged value in the annotated UML model that is later on used by the performance model to conduct the performance evaluation.

2. *deadline* defines the maximum time bound on the completion of the particular execution segment that must be met. The overhead cost and communication cost between the service components are specified by this tagged value in the annotated UML model that is later on used as well by the performance model to conduct the performance evaluation.

3.4. Scalable and automated model transformation

We already mentioned that SPN model will be generated as analytical model from the UML specification style to conduct the performance evaluation. This needs for an efficient, scalable and automated approach to conduct the model transformation for large, complex and multifaceted distributed system. In this literature, the approach for efficient model transformation is realized by producing model transformation rules that can be applied in generalized way for various application domains. As we generate SPN model as analytical model we will give a brief introduction about SPN model. SPN model has the following elements: Finite set of the places (drawn as circles), finite set of the transition defined as either timed transition (drawn as thick transparent bar) or immediate transition (drawn as thin black bar), set of arcs connecting places and transition, multiplicity associated with the arcs, marking that denotes the number of token in each place. SPN model is mentioned formally by the 6-tuple $\{\mathbf{\Phi, T, A, K, N, m_0}\}$:

$\mathbf{\Phi}$ = Finite set of the places

T = Finite set of the transition

$\mathbf{A} \subseteq \{\Phi \times T\} \cup \{T \times \Phi\}$ is a set of arcs connecting Φ and T

K: $T \rightarrow$ {Timed (time>0), Immediate (time = 0)} specifies the type of the each transition

N: $A \rightarrow \{1, 2, 3\ldots\}$ is the multiplicity associated with the arcs in A

m: $\Phi \rightarrow \{0, 1, 2...\}$ is the marking that denotes the number of tokens for each place in Φ. The initial marking is denoted as $\mathbf{m_0}$.

By utilizing the above formal representation of the SPN model, we initiate the model transformation rules that will generate SPN model from UML collaboration and activity oriented approach that captures the system functional behavior. The model transformation rules are the following:

Rule 1: The SPN model of a collaboration role is represented by the 6-tuple in the following way:

$\mathbf{\Phi}$= $\{P_i, d_i\}$

T = {do, exit}

A = $\{\{(P_i \times do) \cup (do \times d_i)\}, \{(d_i \times exit) \cup (exit \times P_i)\}\}$

K = (do $\rightarrow$ Timed, exit $\rightarrow$ Immediate)

N = $\{(P_i \times do) \rightarrow 1, (do \times d_i) \rightarrow 1, (d_i \times exit) \rightarrow 1, (exit \times P_i) \rightarrow 1\}$

$\mathbf{m_o}$ = $\{(P_i \rightarrow 1\}, (d_i \rightarrow 0)\}$

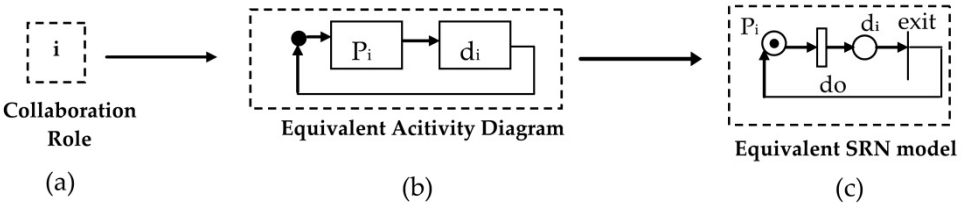

Figure 9. Model transformation rule 1

SPN model of a collaboration role is mentioned in Figure 9 (where P_i = Processing of i^{th} collaboration role and d_i = Processing done of the i^{th} collaboration role).

Rule 2: When the collaboration role of a building block deploys onto a physical node the equivalent SPN model is represented by 6-tuple in following way:

$\mathbf{\Phi} = \{P_i, d_i, PP_n\}$

$\mathbf{T} = \{do, exit\}$

$\mathbf{A} = \{\{(P_i \times do) \cup (do \times d_i)\}, \{(PP_n \times do) \cup (do \times PP_n)\}, \{(d_i \times exit) \cup (exit \times P_i)\}\}$

$\mathbf{K} = (do \rightarrow Timed, exit \rightarrow Immediate)$

$\mathbf{N} = \{(P_i \times do) \rightarrow 1, (do \times d_i) \rightarrow 1, (PP_n \times do) \rightarrow 1, (do \times PP_n) \rightarrow 1(d_i \times exit) \rightarrow 1, (exit \times P_i) \rightarrow 1\}$

$\mathbf{m_o} = \{(P_i \rightarrow 1\}, (d_i \rightarrow 0), (PP_n \rightarrow q)\}$

Initially place PP_n contains **q** (where integer **q** > 0) tokens which define the upper bound of the execution of the process in parallel by a physical node *n* and the timed transition *do* will fire only when there is a token available in both the place P_i and PP_n. The place PP_n will again get back it's token after firing of the timed transition *do* indicating that the node is ready to execute other processes deployed on that physical node. The equivalent SPN model when a collaboration role deploy on a physical node is mentioned in Figure 10:

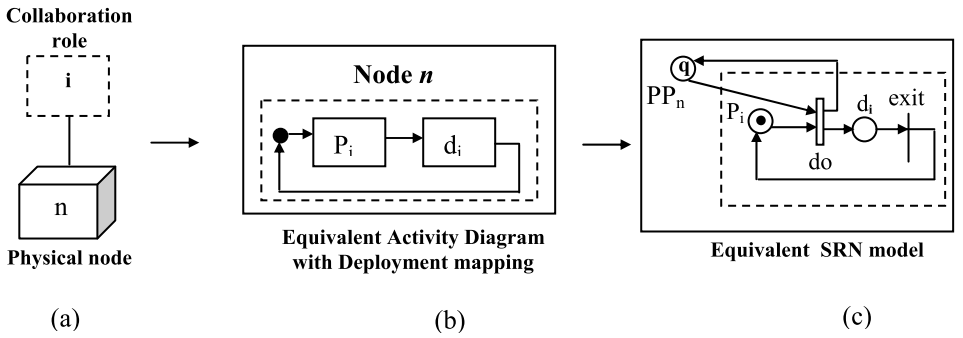

Figure 10. Model transformation rule 2

Rule 3: The SPN model of a collaboration where collaboration connects only two collaboration roles those deploy on the same physical node can be represented by the 6-tuple in the following way in Figure 11:

$\mathbf{\Phi} = \{P_i, d_i, P_j, d_j\ PP_n\}$

$\mathbf{T} = \{do_i, do_j, t_{ij}\}$

$\mathbf{A} = \{(P_i \times do_i) \cup (do_i \times d_i), (PP_n \times do_i) \cup (do_i \times PP_n), (d_i \times t_{ij}) \cup (t_{ij} \times P_i), (P_j \times do_j) \cup (do_j \times d_j), (PP_n \times do_j) \cup (do_j \times PP_n), (d_j \times t_{ij}) \cup (t_{ij} \times P_j)\}$

$\mathbf{K} = \{(do_i, do_j, t_{ij}) \rightarrow \text{Timed}\}$

$\mathbf{N} = \{((P_i \times do_i), (do_i \times d_i), (PP_n \times do_i), (do_i \times PP_n), (d_i \times t_{ij}), (t_{ij} \times P_i), (P_j \times do_j), (do_{ij} \times d_j), (PP_n \times do_j), (do_j \times PP_n), (d_j \times t_{ij}), (t_{ij} \times P_j)) \rightarrow 1\}$

$\mathbf{m_o} = \{(P_i \rightarrow 1), (d_i \rightarrow 0), (P_j \rightarrow 1)\ (d_j \rightarrow 0), (PP_n \rightarrow q)\}$

Here timed transition $\mathbf{t_{ij}}$ in the SPN model is only realized by the overhead cost as service components *i* and *j* deploy on the same physical node which makes the communication cost = 0.

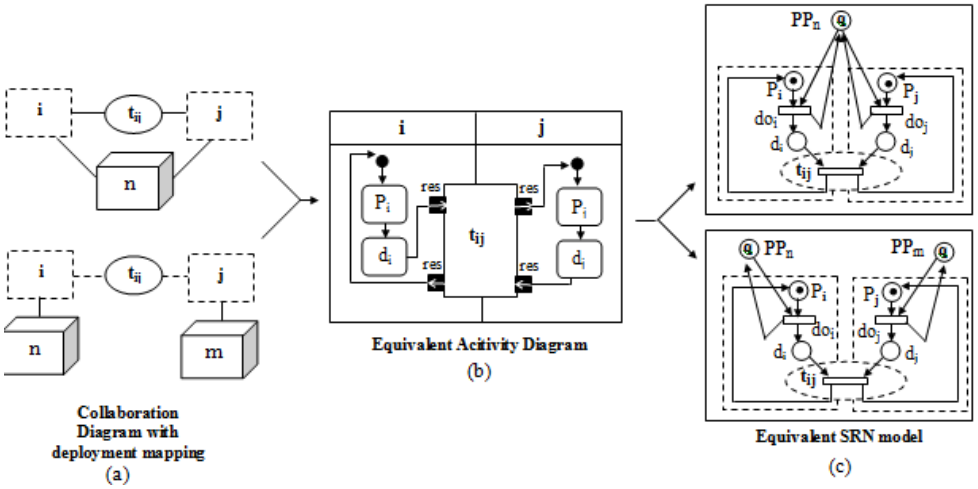

Figure 11. Model transformation rule 3

The SPN model of a collaboration where collaboration connects only two collaboration roles those deploy on the different physical node can be represented by the 6-tuple in the following way in Figure 11:

$\mathbf{\Phi} = \{P_i, d_i, P_j, d_j\ PP_n, PP_m\}$

$\mathbf{T} = \{do_i, do_j, t_{ij}\}$

$\mathbf{A} = \{(P_i \times do_i) \cup (do_i \times d_i), (PP_n \times do_i) \cup (do_i \times PP_n), (d_i \times t_{ij}) \cup (t_{ij} \times P_i), (P_j \times do_j) \cup (do_j \times d_j), (PP_m \times do_j) \cup (do_j \times PP_m), (d_j \times t_{ij}) \cup (t_{ij} \times P_j)\}$

$\mathbf{K}$ = {(do_i, do_j, t_{ij}) ⟶ Timed}

$\mathbf{N}$= {((P_i × do_i), (do_i × d_i), (PP_n × do_i), (do_i × PP_n), (d_i × t_{ij}), (t_{ij} × P_i), (P_j × do_j), (do_{ij} × d_j), (PP_m × do_j), (do_j × PP_m), (d_j × t_{ij}), (t_{ij} × P_j)) ⟶ 1}

$\mathbf{m_o}$ = {(P_i⟶ 1), (d_i ⟶ 0), (P_j⟶ 1), (d_j ⟶ 0), (PP_n ⟶ q), (PP_m ⟶q)}

Here timed transition $\mathbf{t_{ij}}$ in the SPN model is realized by both the overhead cost and communication cost as service components *i* and *j* deploy on the different physical node.

3.5. Performance model Evaluation

We focus on measuring the throughput of the system from the developed SPN model. We are interested in throughput calculation as a measure of job that a system can process in a given time period which in turn justify the efficiency of our deployment logic mentioned in section 3.2 in accordance with system performance evaluation. Before deriving formula for throughput estimation, we consider several assumptions that will allow us to determine the parameters necessary for the throughput calculation of our system.

1. Executions of the processes occur independently each other.
2. All the communications occur in parallel.
3. Finally the communications between interconnected nodes will be started following the completion of all the processing and communication inside each physical node.

The above assumption is important for retrieving the parameters necessary for the throughput calculation from our system specification. We define the throughput as function of expected number of jobs in the system, E(N) and cost of the network, C_Net which defines the time required to complete the expected number of jobs in the system. The value of E(N) is calculated by solving the SPN model using SHARPE [16]. Cost of the network, C_Net is defined in the following: First the cost of a subnet (Csn) will be calculated as follows:

$$Csn_x = \sum_{i=1}^{|m|} fc_i + \max\left\{f_{B_j}\right\} + F_k(M)$$

$$= \sum_{i=1}^{|m|} fc_i + \max\left\{f_{B_j}\right\} \quad (5)$$

Here:

- Csn_x = cost of the x^{th} subnet (where x = 1….n; n is the total number of subnet that comprises the network)
- fc_i = execution cost of the i^{th} process of the x^{th} subnet
- m = total number of service components deployed on the x^{th} subnet
- f_{Bj} = overhead cost of collaboration j (where j =1….n; n is the total number of collaboration in the x^{th} subnet)
- f_{kj} = communication cost of collaboration j (where j =1….n; n is the total number of collaboration in the x^{th} subnet)

- $F_k(M) = 0$ (defined in section 3.2.); as in this case processes connected by the collaboration deploy on the same physical node

Now we evaluate the cost between each pair of subnet with respect to the subnet's own processing cost, overhead cost and the cost associated with the communication with other subnet in the network.

$$Csnp_x = \left\{ \max\left(Csn_i , Csn_j \right) \right\} + f_{B_j} + F_k(M) \} \tag{6}$$

Here:

- $Csnp_y$ = cost of the y^{th} subnet pair (y = 1….n; n is the total number of subnet pair in the network where each subnet pair corresponds between two subnets)
- Csn_i , Csn_i = cost of the i^{th} and j^{th} subnet (where (i, j) ϵ x and i≠j)
- $F_k(M) = 1$ (defined in section 3.2.2); as in this case processes connected by the collaboration deploy on the different physical nodes

$$C\_Net = \max\left\{ Csnp_1 , ..., ..., Csnp_n \right\} \tag{7}$$

$$Throughput = \frac{E(N)}{C\_Net} \tag{8}$$

4. Case study

As a representative example, we consider the scenario dealing with heuristically clustering of modules and assignment of clusters to nodes [15, 17]. This scenario is sufficiently complex to show the applicability of our performance modeling framework. The problem is defined in our approach as a service of collaboration of $E = 10$ components or collaboration roles (labeled $C_1 \ldots C_{10}$) to be deployed and $K = 14$ collaborations between them depicted in Figure 12. We consider three types of requirements in this specification. Besides the execution cost, communication cost and overhead cost, we have a restriction on components C_2, C_7, C_9 regarding their location. They must be bound to nodes n_2, n_1, n_3, respectively. The internal behavior of the collaboration K_i of our example scenario is realized by the call behavior action through same UML activity diagram mentioned in Figure 6(b). The detail behavior of the collaboration role C is realized through same UML activity diagram already illustrated in Figure 6(a). However, there is no behavior modeled in detail, only that collaboration between processes deployed on different physical nodes. The UML collaboration diagram can be modeled by the activity that may model the detail behavior but the level of details must be selected with care in order for the model to scale while generating the performance model.

In this example, the target environment consists only of $N = 3$ identical, interconnected nodes with a single provided property, namely processing power and with infinite communication capacities depicted in Figure 13. The optimal deployment mapping can be observed in Table. 1. The lowest possible deployment cost, according to equation (4) is 17 + 100 + 70 = 187 [15, 17].

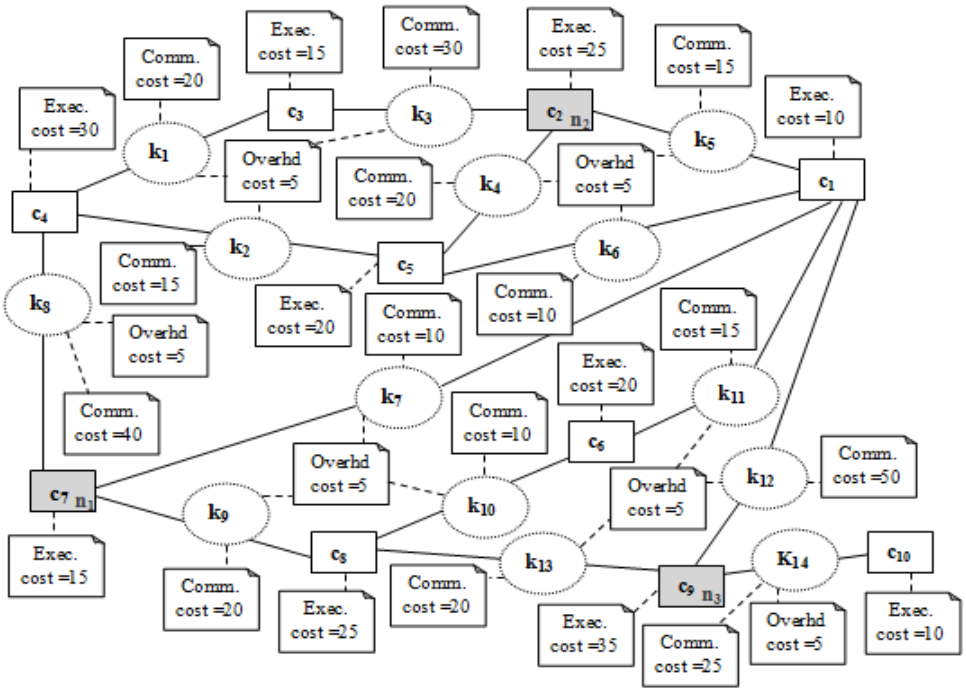

Figure 12. Collaborations and components in the example scenario

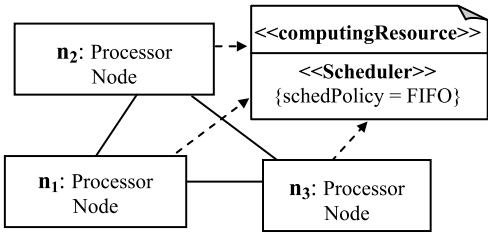

Figure 13. The target network of hosts

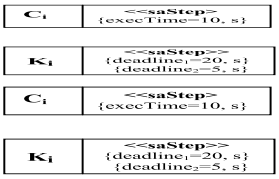

Figure 14. Annotated UML model

In order to annotate the UML diagram in Figure 12 and 13 we use the stereotypes *saStep, computingResource, scheduler* and the tag values *execTime, deadline* and *schedPolicy* which are already described in section 3.3. Collaboration K_i is associated with two instances of *deadline*

(Figure 14) as collaborations in example scenario are associated with two kinds of cost: communication cost and overhead cost.

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}
			17	100
∑ cost			117	

Table 1. Optimal deployment mapping in the example scenario [15, 17]

By considering the above deployment mapping and the transformation rules, the corresponding SPN model of our example scenario is depicted in Figure 15. Figure 15 sketches the resulting SPN model by illustrating details of all the places and transitions. According to the transformation rule 1, each collaboration role is defined by the two places P_i and d_i and the passing of token from place P_i to d_i is realized by the timed transition t_i

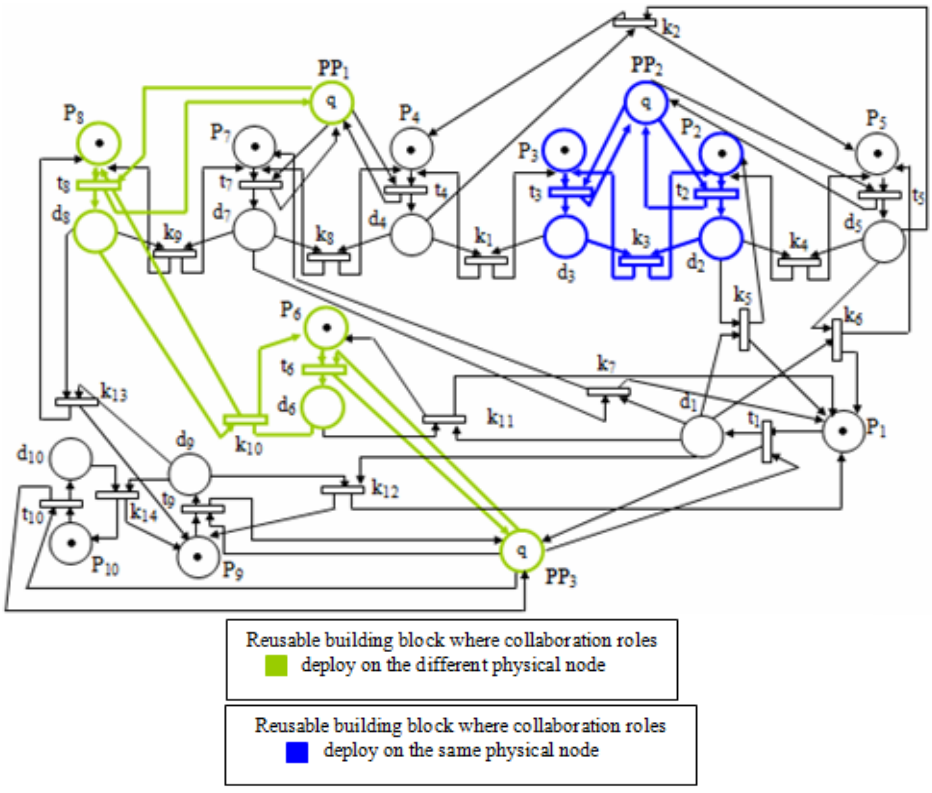

Figure 15. SPN model of our example scenario

which is derived from the annotated UML model. Initially, there will be a token from place P_1 to P_{10}. According to rule 2, in order to define the upper bound of the execution of parallel processes by a network node, we introduce three places PP_1, PP_2 and PP_3 in the SPN model for the corresponding three physical nodes and initially, these three places will contain $\boldsymbol{q}$ ($\boldsymbol{q} > 0$) tokens where $\boldsymbol{q}$ will define the maximum number of the process that will be handled in parallel by a physical node at certain time. In order to ensure the upper bound of the parallel processing of a network node $\boldsymbol{n_1}$, we introduce arcs from place PP_1 to transition t_4, t_7 and t_8. That means, components C_4, C_7 and C_8 can start their processing if there is token available in place PP_1 as the firing of transitions t_4, t_7 and t_8 not only depend on the availability of the token in the place P_4, P_7 and P_8 but also depend on the availability of the token in the place PP_1. Likewise, to ensure the upper bound of the parallel processing of a network node $\boldsymbol{n_2}$ and $\boldsymbol{n_3}$, we introduce arcs from place PP_2 to transition t_2, t_3 and t_5 and from place PP_3 to transition t_1, t_6, t_9, t_{10}.

For generating the SPN model from annotated UML model, firstly, we will consider the collaboration roles deploy on the processor node $\boldsymbol{n_1}$ which are C_4, C_7 and C_8. Here components C_7 connects to C_4 and C_8. The communication cost between the components is zero but there is still cost for execution of the background process. So according to rule 3, after the completion of the transition from place P_7 to d_7 (places of component C_7), from P_4 to d_4 (places of component C_4) and from P_8 to d_8 (places of component C_8) the places d_7, d_4 and d_7, d_8 are connected by the timed transition k_8 and k_9 to generate the SPN model. Collaboration roles C_2, C_3 and C_5 deploy on the processor node $\boldsymbol{n_2}$. Likewise, after the completion of the transition from place P_2 to d_2 (places of component C_2), from P_3 to d_3 (places of component C_3) and from P_5 to d_5 (places of component C_5) the places d_2, d_3 and d_2, d_5 are connected by the timed transition k_3 and k_4 to generate the SPN model according to rule 3. Collaboration roles C_6, C_1, C_9 and C_{10} deploy on the processor node $\boldsymbol{n_3}$. In the same way, after the completion of the transition from place P_1 to d_1 (places of component C_1), from P_6 to d_6 (places of component C_6), P_9 to d_9 (places of component C_9) and from P_{10} to d_{10} (places of component C_{10}) the places d_1, d_6; d_1, d_9 and d_9, d_{10} are connected by the timed transition k_{11}, k_{12} and K_{14} to generate the SPN model following rule 3. In order to generate the system level SPN model we need to combine the entire three SPN model generated for three processor nodes by considering the interconnection among them. In order to compose the SPN models of processor node $\boldsymbol{n_1}$ and $\boldsymbol{n_2}$, places d_4 and d_3 are connected by the timed transition k_1 and places d_4 and d_5 are connected by the timed transition k_2 according to rule 3. Likewise, to compose the SPN models of processor node $\boldsymbol{n_2}$ and $\boldsymbol{n_3}$, places d_2 and d_1 are connected by the timed transition k_5 and places d_5 and d_1 are connected by the timed transition k_6 according to rule 3. In order to compose the SPN models of processor node $\boldsymbol{n_1}$ and $\boldsymbol{n_3}$, places d_7 and d_1 are connected by the timed transition k_7, places d_8 and d_6 are connected by the timed transition k_{10} and places d_8 and d_9 are connected by the timed transition k_{13} according to rule 3. By the above way, the system level SPN model is derived and all these are done automatically. The algorithm for automatic generation of SPN model from the annotated UML model is beyond the scope of this chapter.

The throughput calculation according to equation (8) for the different deployment mapping including the optimal deployment mapping is shown in Table 2. The optimal deployment mapping presented in Table 1 (first entry) also ensures the optimality in case of throughput

calculation though we present here the throughput calculation of some of the deployment mappings of the software artifacts but obviously, the approach presented here confirms the optimality.

Deployment Mapping			Possible total cost	Throughput
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 (min)*	*0.0663 (max)*
{ c_4, c_7},	{c_2, c_3, c_5, c_6,}	{c_1, c_8, c_9, c_{10}}	232	0.0603
{c_4, c_6, c_7, c_8}	{c_2, c_3, c_5}	{c_1, c_9, c_{10}}	218	0.0575
{c_5, c_7, c_8}	{c_2, c_3, c_4}	{ c_1, c_6, c_9, c_{10}}	227	0.0574
{c_1, c_6, c_7, c_8}	{c_2, c_3, c_4}	{c_5, c_9, c_{10}}	247	0.0545
{ c_3, c_7, c_8}	{c_2, c_4, c_5}	{c_1, c_6, c_9, c_{10}}	252	0.0538
{c_4, c_7, c_8}	{ c_1, c_2, c_3, c_5}	{ c_6, c_9, c_{10}}	217	0.0532
{ c_1, c_6, c_7, c_8}	{c_2, c_3, c_5}	{ c_4, c_9, c_{10}}}	257	0.052
{c_3, c_6, c_7, c_8}	{c_1, c_2, c_4, c_5},	{c_9, c_{10}}	302	0.0469
{c_6, c_7, c_8}	{ c_1, c_2, c_4, c_5}	{c_3, c_9, c_{10}}}	288	0.0464

Table 2. Deployment mapping in the example scenario along with throughput

5. Conclusion

The contribution of this chapter is to develop a framework that focuses on the performance evaluation of the distributed system using SPN model. The developed framework recognizes the fact of rapid and efficient way of capturing the system dynamics utilizing reusable specification of software components that has been utilized to generate SPN performance model. The deployment logic presented here, is applied to provide the optimal, initial mapping of components to hosts, i.e. the network is considered rather static. 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. 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. Our modeling framework support that, our logic will be a prominent candidate for a robust and adaptive service execution platform for assessing a deployment of service components on an existing physical topology. Future work includes providing a tool based support of the developed performance modeling framework.

Author details

Razib Hayat Khan and Poul E. Heegaard
Norwegian University of Science & Technology, Trondheim, Norway

Kazi Wali Ullah
Aalto University, Helsinki, Finland

6. References

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

[2] J. P. Lopez, J. Merseguer, and J. Campos, "From UML Activity Diagrams to SPN: Application to Software Performance Engineering", Workshop on Software & Performance, ACM SIGSOFT software engineering notes, pp. 25-36, NY, 2004

[3] S. Distefano,M. Scarpa, and A. Puliafito, "Software Performance Analysis in UML Models", Workshop on Techniques Methodologies and Tools for Performance Evaluation of Complex Systems, Vol. 2005, pp. 115-125, 2005

[4] A. D'Ambrogio, "A Model Transformation Framework for the Automated Building of Performance Models from UML Models", Workshop on Software & Performance, ACM SIGSOFT software engineering notes, NY, 2005

[5] R H Khan, P E Heegaard, "Translation from UML to SPN model: A Performance Modeling Framework", EUNICE, pp. 270-271, LNCS, Springer, 2010

[6] R H Khan, P E Heegaard, "Translation from UML to SPN model: A Performance Modeling Framework for Managing Behavior of Multiple Collaborative Sessions & Instances" ICCDA, pp. V5-72-V5-80, IEEE computer Society, China, 2010

[7] R H Khan, P E Heegaard, " A Performance Modeling Framework Incorporating Cost Efficient Deployment of Collaborating Components" ICSTE, pp. V1-340-V1-349, IEEE computer Society, San Juan, USA, 2010

[8] Moreno Marzolla, "Simulation-based Performance Modeling of UML Software Architectures", PhD thesis, Universit`a Ca' Foscari di Venezia, 2004

[9] Frank Kræmer, "Engineering System: A Compositional and Mode-Driven Method Based on Collaborative Building block", PhD Thesis, NTNU, Trondheim, Norway, 2008

[10] F. A. Kramer, R. Bræk, P. Herrmann, "Synthesizes Components with Sessions from Collaboration-oriented Service Specifications", Proceedings of SDL, V-4745, LNCS, Springer, 2007.

[11] OMG UML Superstructure, Version-2.2

[12] R H Khan, P E Heegaard, "A Performance Modeling Framework Incorporating Cost Efficient Deployment of Multiple Collaborating Instances" ICSECS, pp. 31-45, Springer, 2011

[13] J. Trowitzsch, A. Zimmermann, "Using UML State Machines and Petri Nets for the Quantitative Investigation of ETCS", Valuetools, 2006

[14] Abdullatif, R. Pooly, "A Computer Assisted State Marking Method for Extracting Performance Models from Design Models", International Journal of Simulation, Vol. 8, No. 3, pp. 36-46, 2008

[15] Mate J Csorba, "Cost Efficient Deployment of Distributed Software Services", PhD Thesis, NTNU, Norway, 2011

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

[17] Efe, K., "Heuristic Models of Task Assignment Scheduling in Distributed Systems", Computer, 1982