Software Performance Evaluation Utilizing UML Specification and SRN Model and Their Formal Representation

Razib Hayat Khan*, Poul E. Heegaard
Norwegian University of Science & Technology (NTNU), Trondheim, Norway.

* Corresponding author. Tel.: +4745578010, email: rkhan@alumni.ntnu.no.
Manuscript submitted March 8, 2014; accepted October 12, 2014.
doi: 10.17706/jsw.10.5.499-523

Abstract: Performance assessment should be included in the software development process to ensure the quality of the software in operation. Hence, this paper introduces a performance modeling framework for a distributed software system that proposes a transformation process from a high level Unified Modeling Language (UML) notation to a Stochastic Reward Net (SRN) model. The model is solved for relevant performance metrics. In order to capture the system dynamics through our modeling framework, we outline a specification style that focuses on the UML collaboration and activity as reusable specification building blocks, while UML deployment diagram identifies the physical components of the system and the assignment of software artifacts to the identified system components. Efficient deployment mapping of software components on the available physical resources of the system is investigated by deriving the cost function. The UML specification presented in this work facilitates to delineate the complete behavior of the system components according to the functionalities they offer and has been utilized as input for model transformation. The paper describes the mapping rules for defining the transformation process from the UML model to the SRN model. In addition, the tool support of the framework is introduced that ensures automatic translation from UML to SRN model and evaluation of the SRN model. The applicability of our framework is demonstrated in the context of performance modeling of distributed systems.

Key words: Performance, reusability, service deployment, UML, SRN.

1. Introduction

The design and implementation of distributed systems and services are always intricate endeavors and complex tasks (The terms “system” and “service” are used interchangeably in this paper). Systems consist of logical components that interact and are deployed in a physical, resource-constrained infrastructure. Quantitative analysis determines whether a system achieves its non-functional properties based on the functional behavior that is mapped onto a physical, resource-constrained infrastructure. Quantitative analysis is realized by conducting a performance evaluation of the distributed system. It is evident that a successful development process for a distributed system is not guided solely by the perfect modeling and implementation of such a system. This process is also supported by the early assessment of performance-related factors, which helps developers to reveal any bottleneck in the modeling, design, and implementation of a distributed system that can jeopardize meeting end-user expectations. This, in turn, reduces the cost of making any modification of the system architecture once the system is built. In the

worst case, this modification might require restarting the development process from the beginning. However, the perfect modeling of the system functional behavior is a great concern at the early stage of the development process for the acceptable evaluation of a system performance. Being able to assess the performance of a distributed system at an early stage in the development process is considered to be a matter of great importance. However, this is a considerable challenge that includes the following:

- Precise and formally correct description of the system functional behavior, even at the early stage
- Representation of the physical infrastructure that the system is expected to be executed on, including the resource constraints
- Non-functional attributes of the system, including acceptable thresholds (e.g., given as QoS parameters in a Service Level Agreement (SLA))
- Knowledge of the specific system complexity and domain, with the expected usage pattern, deployment strategy, operational issues, environmental influences, and other related factors
- Selection of an approach for performance evaluation to produce pertinent results in accordance with the real system behavior

In order to develop a framework that addresses these challenges, we need the following:

- Modeling functional behavior in a manner that can be combined with the deployment and enables scalable, automated translation into a model that can assess the performance of a distributed system
- Non-functional properties that reflect the performance attributes of the system
- Physical resource constraints to capture the effect of different deployment strategies
- Performance evaluation approach to uncover meaningful evaluation results when the real system does not exist

Considering the above-mentioned factors, the general structure of the performance modeling framework for a distributed system is illustrated in Fig. 1 [1]. The rounded rectangle in the figure represents operational steps, whereas the square boxes represent input/output data. The inputs for the automated model transformation process are as follows:

- A system or service functional behavior
- Information of a system execution environment
- Non-functional parameters

The model representations of the functional behavior, physical platform, and non-functional properties are combined to form annotated models. Then, a given deployment of the service components onto the currently available physical resources (assumed static throughout the evaluation) is added, using annotated models. Finally, this deployment specification is automatically translated into a performance evaluation model and the performance of the system can be assessed. When using the framework given in Fig. 1, several feedback loops will exist, depending on the objective of the case study [1]. Furthermore, the modeling process is supported by a set of tools, including Arctis and Symbolic Hierarchical Automated Reliability/Performance Evaluator (SHARPE) with the incremental model checking facility. Arctis is used for specifying the system functional behavior [2]. The evaluation of the performance models generated by the framework is achieved using SHARPE [3].

Numerous efforts have been considered so far to develop a performance model from the system design specification. Lopez-Grao *et al.* proposed a conversion method where annotated UML activity diagram is formalized using generalized stochastic petrinet model [4]. In Ref. [5], the authors presented an overview of their aspect-oriented Formal Design Analysis Framework (FDAF) and how it can be used to design and analyze performance properties. In Ref. [6], the author experienced annotating UML class diagrams with fragments of the Object-Z specification language. Another approach is proposed by IBM where the Object Constraint Language (OCL) [7] is used as a standard formal specification language to formalize UML

diagrams. In Ref. [8], the author proposed Software/Hardware Engineering (SHE) method that enables constructing formal executable models based on the expressive modeling Parallel Object-Oriented Specification Language (POOSL). The related works presented here still lack of a complete formal semantics and capabilities to prove the properties and refinements as well as fails to provide the formal representation of concurrent activities. Some other efforts have been taken to outline formal definition of UML models and mostly the works are based on the UML statechart diagram and sequence diagram [9] - [12]. UML activity diagram is not thoroughly investigated to generate a performance model for software performance analysis except some of the works in [5], [13], [14]. Moreover, several tools exist in this regard to generate a performance model from UML [15] - [19]. ArgoUML is a well established UML modeling tool with no support of generating performance model [15]. UML-Performance Simulator (UML-PSI) is a software performance evaluation tool based on the process-oriented simulation [16]. PERMABASE uses UML class, sequence, and deployment diagram to generate performance model [17]. Protocol Software Performance Engineering using XMI (proSPEX) is a performance modeling tool for the design and performance analysis of communication protocols specified with UML [18]. Performance by Unified Model Analysis (PUMA) provides a unified interface between different kinds of design information and different kinds of performance models, for example - markov models, Stochastic Petri Nets (SPN), process algebras, queues, and layered queues [19]. Among all the cited tools, only PUMA has the support of generating SPN model. Our model semantics and supported diagrams are different than the above mentioned framework and also our approach supports the model checking facility.

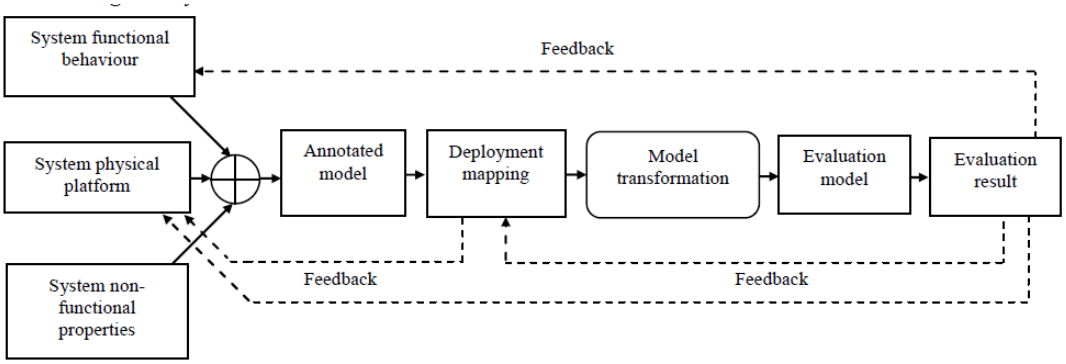


Fig. 1. General structure of the performance and performability framework [1].

The objective of this paper is to provide an extensive modeling framework that offers an automatic transformation process from system specification models to performance evaluation models for generating the performance result of the systems at the early stage of the development process. The transformation must be scalable and efficient in such a way that it can hide the intricate system details. The main focus of this paper is on the framework shown in Fig. 1, mostly without the feedback loops [1]. The paper is organized as follows: Section 2 delineates the overview of the framework, Section 3 describes the formalization of UML model, Section 4 defines the tool support of our framework, Section 5 explains the case study, and Section 6 delineates the conclusion with future directions.

2. Overview of the Framework

2.1. Service Functional Behavior

We adopt a model-driven approach, where the functional behavior of service is specified using UML, which is a universally used modeling specification language that is widely accepted by the scientific community [20]. UML provides a set of diagrams that facilitates illustration of system behavior from

different viewpoints and from different detail levels. UML collaboration is utilized as the main specification unit to define the service functional behavior. The service components are defined as collaboration roles and the interactions among the collaboration roles are specified by the collaboration diagram. The UML collaboration diagram that is mentioned in our work is purely structural. That means the collaboration diagram defines the structure of the service specification as combination of service components and the necessary interactions among them. In particular, we model collaboration in the form of encapsulated building blocks in a self-contained way that can easily be composed with each other [21]. In particular, capturing the properties of the system into collaborative building blocks will allow system developers to reuse those building blocks. This, in turn, makes the developer's tasks easier and faster by removing the need for the system developer to be an expert in all domains of distributed systems. The reusability of encapsulated collaborative building blocks provides tremendous benefits to delineate system functional behavior such as the following:

- When the collaborative building block will be formed in a self-contained way, the system developers can just reuse them to build the system without dealing inner complexities
- Collaborative building blocks might be combined into more comprehensive services. This means that new services can be developed combining existing building blocks rather than starting the development process from scratch. This, in turn, increases the productivity in accordance with cost reduction.
- Modeling system functional behavior by composing the basic specification unit thus reflects the subsystem properties in the resulting system behavior. Thus, the system overall functional behavior will be consistent with the behavior of its components.

As the service specification later on will be transformed into a performance model, provided only the structural specification of the service is not sufficient. This transformation requires a way to define the behavioral aspects of the collaboration to know the exact functional behavior of the service components. In order to remove this shortcoming, we use UML activity diagram. The internal behavior of a 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 delineating the detailed behavior of the system and to know about how the different events of the collaborative building blocks are coupled, UML collaborations and activities are used to complement each other. UML collaborations focus on the role binding and structural aspects, whereas the UML activities complement this by also covering the behavioral aspect [22]. For this purpose, call behavior actions are used. Collaboration is represented by call behavior action referring to the respective activity of building blocks. Each call behavior action represents an instance of a building block. For each activity parameter node of the referred activity, a call behavior action declares a corresponding pin. Pins have the same symbol as the 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 collaborations can be coupled with each other.

2.2. System Physical Platform

Specification of the system physical platform is incorporated into our modeling framework by the use of a UML deployment diagram. A UML deployment diagram is used in our modeling framework to define the execution architecture of the system by identifying the system physical components, the connections between components, and the assignment of software artifacts to those identified physical components [20]. Service delivered by the system is defined by the joint behavior of the system components, which are

physically distributed. This, in turn, aids in exposing the direct mapping between the software components to the system physical components to exhibit the probable deployment of the service.

2.3. System Non-functional Properties

Specification of system non-functional properties is included in our modeling framework using a UML profile. Profiles in UML are defined using stereotypes and tag definitions that are applied to specific model elements, such as classes, attributes, operations, and activities. A profile is a collection of such extensions that collectively customize the UML for a particular domain or platform [23]. Stereotypes permit us to map model elements to the semantics of an analysis domain and provide values for properties that are necessary to conduct the analysis. Specific tagged values are also applied according to the UML profile. Tagged values are a type of value slot associated with the attributes of specific UML stereotypes [23], [24]. The significance of using UML profile is that it provides a common method of modeling both hardware and software aspects to capture real-time properties of the system. It must be possible to construct UML models that can be used to analyze and predict the salient real-time properties of a system. In particular, it is important to be able to perform such analysis early in the development cycle. Moreover, Modelers should be able to take advantage of different types of model analysis techniques without requiring a deep understanding of the inner workings of those techniques. We use *UML profile for MARTE – Modeling and Analysis of Real-Time Embedded Systems* for describing the non-functional properties of the system [4]. 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 [24]. The modeling part provides the support required from specification to detailed design of real-time and embedded characteristics of systems. MARTE also concerns model-based analysis. Hence, it provides facilities to annotate models with information required to perform specific analysis. The stereotypes and tagged values according to the MARTE profile used in this paper are defined below [23], [24]:

- SaStep: saStep is a type of step that begins and ends when decisions about the allocation of system resources are made.
- ComputingResource: A ComputingResource represents either virtual or physical processing devices capable of storing and executing program code. Hence, its fundamental function is to compute.
- Scheduler: Scheduler is a stereotype that brings access to a resource following a certain scheduling policy mentioned by the tagged value schedPolicy.
- SaSharedResource: SaSharedResource is a type of shared resources that are dynamically allocated by means of an access policy.
- GaExecHost: GaExecHost can be any device that executes behavior, including storage and peripheral devices.
- schedPolicy: schedPolicy defines certain scheduling policies based on which access of system physical resources can be conducted.
- deadline: deadline defines the maximum time limits for the completion of the particular execution segment.
- resmult: resmult indicates the multiplicity of a resource. It may specify the maximum number of instances of the resource considered as available.
- capacity: capacity defines the number of permissible concurrent users.

2.4. Annotated Model

UML is used to specify the service functional behavior by identifying the software components and interactions between them. UML reveals the relations between software components with available

physical resources in the execution environment. However, one shortcoming of UML is not having the capability to incorporate non-functional parameters, which is vital for conducting the quantitative analysis of the system. This requires a mechanism for providing a specification to make quantitative prediction regarding non-functional properties of the system, taking into account both software and hardware characteristics. Thus, we use MARTE specification style to generate annotated UML model, which provides several stereotypes and tagged values (see Section 2.3) for incorporating performance-related parameters into the service specification model defined by the UML [23]. Examples of annotated model have been given in the case study section (see Section 5).

2.5. Deployment Mapping

An efficient allocation of the service components over the physical infrastructure (execution environment) is crucial for achieving good performance. Given a static set of available resources in a fixed topology, the deployment strategy needs to consider the following:

- Overhead cost between two communicating service components. It is assumed that two components on the same physical component have lower overhead than two service components that are not co-located.
- Resource constraints (capacities) of the physical components constituting the infrastructure, such as link capacity, memory storage
- Non-functional properties (attributes with corresponding thresholds), such as processing capacity of the distributed node

In real systems, this task is even more complex, as the infrastructure might alter available system resources and topology dynamically. This alteration occurs due to events such as failures or overload that might significantly delay or block the operation of certain components. The objective of the deployment strategies is to find the best possible service component mapping on the currently available physical resources, satisfying all the non-functional requirements [25]. The resulting deployment mapping has a large influence on the QoS that will be provided by the system. In this paper, we do not develop optimal deployment strategies but rather gives a framework for the performance of a given deployment that can be specified and assessed.

We model the system as a collection of N interconnected physical nodes. Our objective is to find a deployment mapping for this execution environment for a set of service components that comprises the service [24]. Deployment mapping **M** is defined as [$\mathbf{M}=(C \rightarrow N)$], between a number of service component instances $\{c_1, c_2, \ldots\} \in \mathbf{C}$ (captured by collaboration diagram), onto physical nodes $\{n_1, n_2, \ldots.\} \in \mathbf{N}$ (captured by UML deployment diagram). In this setting, the service components communicate with each other via a set of collaborations $\{k_1, k_2, \ldots.\} \in \mathbf{K}$. Hence, a collaboration k_j may exist between two components c_a and c_b. Considering the above description, we focus on the deployment mapping assessment of the service components to the available physical resources by considering several non-functional properties. Assessing the deployment mapping of the service components is realized by defining cost functions, which is a function that expresses the utility of deployment mapping of the service components to the physical resources with their constraints and capabilities to satisfy the non-functional properties of the system. We consider three 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 [24]:

- Service components have execution costs
- Collaborations have communication costs and costs for execution of background process, which are known as overhead costs

- Some of the service components can be restricted in the deployment mapping to specific physical nodes, which are called bound components

Furthermore, we observe the processing cost that physical nodes impose, while hosting the service components and also the target balancing of the cost among the physical nodes available in the network. Communication costs are considered if the collaboration between two service components occurs remotely, i.e., it occurs between two physical nodes [26], [24]. In other words, if two service components are placed onto the same physical node, the communication cost between them will be ignored. This holds for the case study that is conducted in this paper. This is not generally true, and it is not a limiting factor of our framework. The cost for execution of the background process for conducting the communication between the collaboration roles is always considerable no matter whether the collaboration roles are deployed on the same or different physical nodes. Using the above specified input, the deployment logic provides an efficient deployment architecture taking into account the QoS requirements for the specified services. We then define the objective of the deployment logic as obtaining an efficient (low-cost, if possible optimum) mapping of the service components onto the physical nodes that satisfies the requirements in a reasonable time. The deployment logic is mentioned by the cost function F(**M**) [24]. The cost function should reflect the execution, communication, and overhead cost. Ideally, the service turnaround time should be minimized, which, in turn, maximizes the utilization of system resources while minimizing the communication between processing nodes. As a result, a high system throughput can be accomplished taking into account the expected execution and inter-node communication requirements of the service components on the given hardware architecture [27]. The cost function F(**M**) is mainly influenced by our method of service definition. Service is defined in our approach as a collaboration of E service components labeled as c_i (where i = 1,....,E) and K collaborations labeled as k_j, (where j = 1,...,K). In addition, the following labeling methods are used [24]:

- The execution cost between the service components is labeled, f_{c_i}
- The communication cost between the service components is labeled, f_{k_j}
- The cost for execution of the background process for conducting the communication between the service components (overhead cost) is labeled, f_{B_j}

We will assess the quality of the solution of equally distributed cost among the processing nodes and the lowest cost possible, while taking into account the following:

- execution cost f_{c_i} , i = 1,....,E
- communication cost f_{k_j} , j = 1,....,K
- overhead cost f_{B_j} , j = 1,....,K

f_{c_i} , f_{k_j} ,and f_{B_j} are derived from the service specification, and thus, the total offered execution cost for the given service can be calculated as $\sum_{i=1}^{|E|} f_{c_i}$ [24]. Hence, the average load $\boldsymbol{T_a}$ becomes [25]:

$$\boldsymbol{T}_a = \frac{1}{|X|} \sum_{i=1}^{|E|} f_{c_i} \quad (1)$$

where X = the available total nodes in a network N where the service is deployed.

To account for the communication cost f_{k_j} of the collaboration k_j in the service, the function $q_0(\mathbf{M}, c)$ is defined first [25]:

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

This means that $q_0(\mathbf{M}, c)$ returns the physical node n from a set of physical nodes N available in the network that hosts components in the list mapping **M** [24]. Let collaboration $k_1 = (c_1, c_2)$ and set of physical nodes $N = (n_1, n_2)$ taking into account the following:

- The communication cost of k_1 is 0 if components c_1 and c_2 are co-located, i.e., $q_0(\mathbf{M}, c_1) = q_0(\mathbf{M}, c_2)$ (if M = ({c_1, c_2→ n_1}) or ({c_1, c_2→ n_2})).
- The cost is f_{k_j} if the service components are otherwise co-located (i.e., the collaboration is remote) (if M = ({c_1→ n_1} {c_2→ n_2}) or ({c_1→ n_2} {c_2→ n_1})).
- Using an indicator function $I(x)$, this is expressed as $I(q_0(\mathbf{M}, c_1) \neq q_0(\mathbf{M}, c_2)) = 1$, if the collaboration is remote and 0 otherwise.
- To determine which collaboration kj is remote, the set of mapping M is used. Given the indicator function, the overall communication cost of service, FK(M), is the sum [24], [25]:

$$\mathrm{F_K}(\mathbf{M}) = \sum_{j=1}^{|K|} I(q_0(\mathbf{M}, k_{j,1}) \neq q_0(\mathbf{M}, k_{j,2})) \cdot f_{k_j} \tag{3}$$

Given a mapping **M** = {m_n} (where m_n is the set of service components at physical node n) the total load can be obtained as $\hat{l}_n = \sum_{c_i \in m_n} f_{c_i}$ [24]. Furthermore, the overall cost function F(**M**) becomes [24], [25] (where I_j = 1, if k_j external or 0 if k_j internal to a node):

$$\mathrm{F}(\mathbf{M}) = \sum_{n=1}^{|X|} | \hat{l}_n - \boldsymbol{T}_a | + \mathrm{F_K}(\mathbf{M}) + \sum_{j=1}^{|K|} f_{B_j} \tag{4}$$

The absolute value $| \hat{l}_n - \boldsymbol{T}_a |$ is used to penalize the deviation from the desired average load per physical node.

2.6. Model Transformation

In order to conduct the early assessment of performance modeling, the service functional specification defined by the UML is transformed into analytical model. The reason for choosing analytical model is that the analytical models are attractive because of the existence of well established formulas and the availability of analysis tools for conducting quantitative system evaluation. It is very efficient with respect to the execution time and requirements of computational resources to evaluate the analytical model using evaluation tools. Nevertheless, a critical challenge with the analytical model is restrictive modeling assumptions that sometimes hinder capturing the real behavior of the system. This challenge can be met by the use of simulation method under less restrictive assumptions. However, model evaluation using the analytical approach becomes complex when the evaluation needs to be conducted at an early stage of the system development process. There might be a chance of producing erroneous results during the early assessment, which can result in the incorrect development and implementation of the whole system. The solution to the problem lies in the correct representation of the modeling formalisms of the system functional behavior and conducting the model transformation accurately to generate the analytical model.

Conducting the model transformation in a correct, automated, and scalable way requires developing reusable model transformation rules that can handle the model transformation process of large and multifaceted systems. The reusability of model transformation rules makes the model transformation process easier and faster for the system developers who will just apply the rules for model transformation without understanding the inner complexity.

In particular, we will generate SRN models as analytical models from service functional specification. The reason behind focusing on the SRN model is due to having 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 [28]. The SRN model is described formally by the 6-tuple $\{\boldsymbol{\Phi, T, A, TT, Ml, m_0}\}$ in the following manner [24]:

Φ = Finite set of the place

T = Finite set of the transition

A $\subseteq \{\Phi \times T\} \cup \{T \times \Phi\}$ is a set of the arc connecting Φ and *T*

TT: T → {Timed (time>0), Immediate (time = 0)} specifies the type of the each transition.

Ml: A→ {1, 2, 3,…} is the multiplicity associated with the arcs in A.

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

In order to keep the service specification model and analytical model consistent with each other, the process of model transformation is driven by the model transformation rules, which provide an efficient, scalable, and automated approach to conduct model transformation for large, complex, and multifaceted distributed systems. By considering the semantic definition of the SRN models, we provide the model transformation rules to generate performance models. Rules for generating performance SRN models can be divided into two categories:

- Rule 1: Deployment mapping of a collaboration role
- Rule 2: Deployment mapping of collaboration

Rule 1: Deployment mapping of a collaboration role: Rule 1 addresses the generation of an SRN model of a collaboration role with deployment mapping, which is shown in Fig. 2 (where P_i = Processing of i^{th} collaboration role and d_i = Processing performed of the i^{th} collaboration role) [1]. Mainly, rule 1 has been utilized to model the load of a physical node. For each physical node, there must be an upper bound on the execution of the process in parallel with that node. The execution of the process is only possible when the node has the capacity to do so. When the collaboration role of a building block deploys onto a physical node, the equivalent SRN model is illustrated in Fig. 2 [1]. Initially, place PP_n contains $\boldsymbol{q}$ (where integer $\mathbf{q} > 0$) tokens, which define the upper bound on the execution of the process in parallel with a physical node n, and the timed transition *do* will fire (which symbolizes the execution of the process *i*) only when there is a token available in both the place P_i and PP_n. The place PP_n will again receive its token back after firing of the timed transition *do,* indicating that the node is ready to execute other processes deployed on that node.

When the collaboration role of a building block deploys onto a physical node, the equivalent SRN model is represented by the 6-tuple in the following manner:

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

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

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

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

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

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

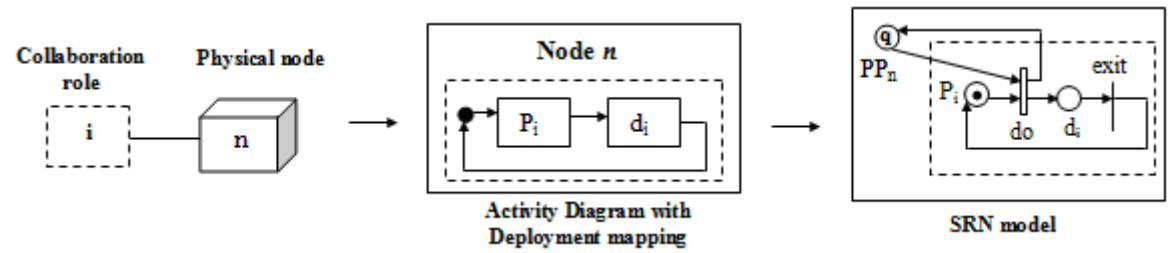


Fig. 2. Model transformation rule 1 [1].

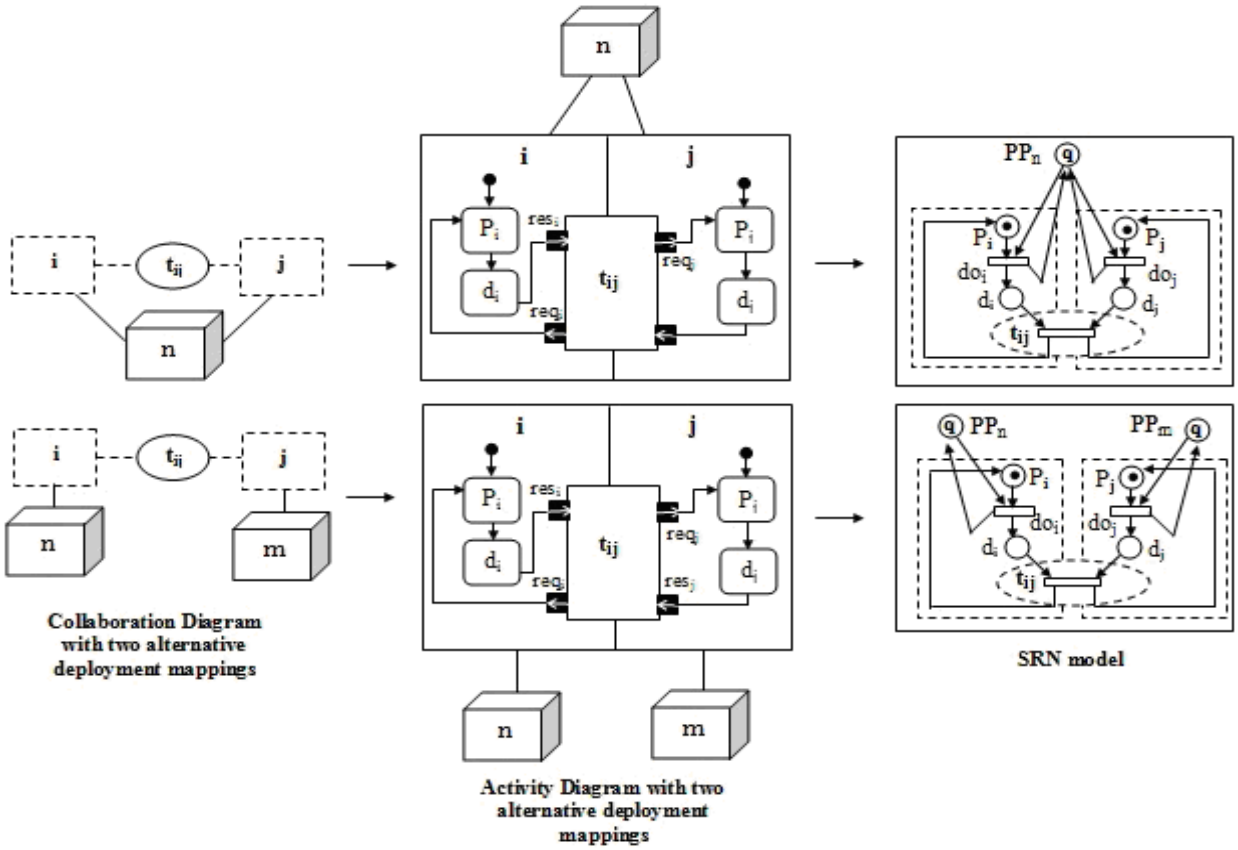


Fig. 3. Model transformation rule 2 [1].

Rule 2: Deployment mapping of collaboration: Rule 2 addresses the generation of an SRN model of collaboration, which is illustrated in Fig. 3 [1]. The collaboration connects only two collaboration roles in bidirectional manner, where roles are deployed on the same or different physical nodes. When collaboration roles *i* and *j* are deployed on the same physical node *n*, the timed transition t_{ij} in the SRN model is only realized by the overhead cost, as in this case, communication cost = 0. When collaboration roles *i* and *j* are deployed on the different physical nodes ***n*** and ***m***, the timed transition t_{ij} in the SRN model is realized by both the overhead cost and communication cost.

The SRN model of a collaboration, where collaboration connects only two collaboration roles in a bidirectional manner and the roles are deployed on the same physical node can be represented by the 6-tuple in the following way:

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

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

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

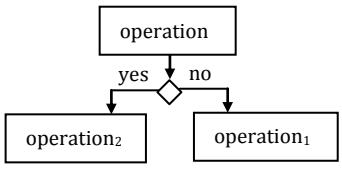
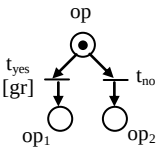
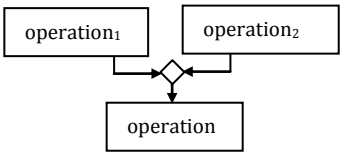
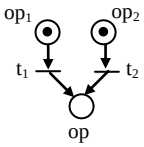
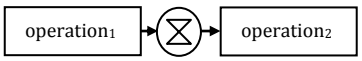
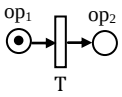
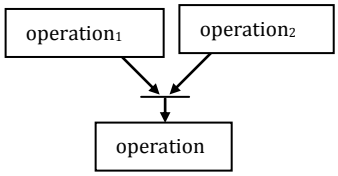
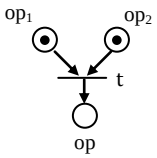
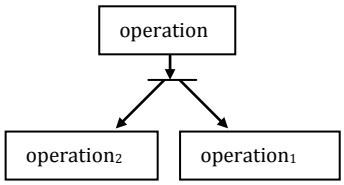
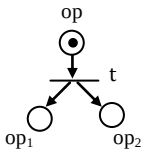
$\boldsymbol{TT} = \{(do_i, do_j, t_{ij}) \rightarrow Timed\}$

Ml= {((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_j × d_j), (PP_n × do_j), (do_j × PP_n), (d_j × t_{ij}), (t_{ij} × P_j)) →1}

$\boldsymbol{m_o}$ = {(P_i→*1*),(d_i →*0*), (P_j→*1*) (d_j →*0*), (PP_n →*q*)}

Similar to the above, the UML model of collaboration can be represented by the 6-tuple, where collaboration connects only two collaboration roles in bidirectional way and the roles are deployed on the different physical nodes.

Table 1. Transformation Rules of UML Nodes [1]

UML Node	SRN model
 Decision node	
 Merge node	
 Timer node	
 Join node	
 Fork node	

Furthermore, we present the transformation rules in Table 1 to generate SRN models for some of the components of the activity diagram that might have been presented in collaborative building blocks [1]. Regarding the decision node, the activation of outgoing flow is achieved based on a condition that means the decision is not random. Therefore, we attach a guard function with immediate or timed transition in the equivalent SRN model of the decision node during the model transformation. The guard function is associated with a condition based on which the activation of the transition will be permitted. For example, immediate transition t_{yes} in Table 1 is attached with a guard function [gr] in the SRN model, which will capture the same condition to be activated as the outgoing flow indicated as "yes" in the activity diagram. If

the condition of the guard function [gr] is fulfilled, the attached transition will be activated. In the same manner, it is also possible to attach a guard function with the immediate transition t_{no} instead of transition t_{yes} and to conduct the same process for the activation of outgoing flow.

3. Formalizing UML Specification Style Using CTLA

The UML specification style introduced in this paper has been utilized to define the exact and complete behavior of the service specification that focuses on the functionalities they offer and will be used as an input model for the model transformation process. Though UML provides comprehensive architectural modeling capabilities, it lacks the ability to present the modeling specifications formally and does not convey formal semantics or syntax. As a result, we delineate the precise semantics of UML collaborations, activities, and deployment by formalizing the concept in the temporal logic compositional Temporal Logic of Actions (cTLA) style, which is defined as cTLA/c [29]. This is a variant of TLA which provides couplings based on jointly executed transitions enabling to glue interacting constraints nicely. Moreover, cTLA makes the description of state transition systems in a process-like style [29]. The motivation behind expressing the semantics using cTLA is to describe various forms of structures and actions through an assortment of operators and techniques, which correspond superbly with UML collaboration, activity, and deployment diagrams.

The concept of UML collaboration introduced in this work is rather structural and describes a structure of collaborating elements. To illustrate the structural concept of the collaboration, collaborations are mapped into a cTLA/c process, where the process is realized between the collaboration roles internal to the collaborations. In [29], the detailed process for formalizing the UML collaboration specification is introduced, where the focus was not only to specify the behavior internally to the collaboration but also to define the mechanism to couple the collaborations with others during the composition if necessary.

UML activities have been utilized to express the behavior of collaborations. UML collaboration is complemented by an activity, which uses one separate activity partition for each collaboration role. In terms of the cTLA/c, an activity partition corresponds to a collaboration role. The semantics of UML activities are based on the Petri nets [21]. Thus, an activity essentially describes a state transition system, with the token movements as the transitions and the placement of tokens within the graph as the states. Consequently, the variables of a cTLA/c specification model the actual token placement on the activity, while its actions specify the flow of tokens between states. Flows may cross partition borders. According to the cTLA/c definition and as partitions are implemented by distributed components, flows moving between partitions are modeled by communication buffers, while states assigned to activity nodes are represented in cTLA/c by local variables. To define the semantics of activities using cTLA/c, we opted for an approach that directly uses the mechanisms of cTLA. Illustration of some activity element types as separate cTLA processes are given in [29], which help to understand the semantics of the activity elements. Moreover, the production rules of cTLA actions for UML activities have been presented in [29] to produce the system actions from the local process actions as a set of rules, so that each activity element can be defined separately.

The concept of a UML deployment diagram is also structural and describes a structure of the execution environment by identifying a system's physical layout. It also specifies which pieces of service components run on what pieces of physical nodes and how nodes are connected by communication paths. We use a specific tuple class as an additional invariant that is also a part of the style cTLA/c to model UML deployment [1].

The mapping process to demonstrate the correspondence between formalized UML diagram and SRN model has been demonstrated in [1] which is beyond the scope of this paper.

4. Tool Support of Our Framework

Tool support is an essential part of our modeling framework. A tool provides editing support and an automated means of model transformation with the capability to verify the model. It also provides a faster way of model development and evaluation. We have used two tools:

- Arctis for defining service functional behavior
- SHARPE for generating model evaluation result

The tools are tailored to serve their own purpose in this work, but the integration achieved between these tools through our modeling framework performs a novel and complete task that spans from the modeling of service functional behavior to the performance evaluation of that service.

The service specification models of our modeling framework, such as the UML collaboration and activity diagram (described in Section 2.1), are generated using the Arctis tool [2]. Arctis focuses on the abstract, reusable service specifications that are composed of UML collaborations and activities. It uses collaborative building blocks as reusable specification units to provide the structural and behavioral aspects of the service components. To support the construction of building blocks that consist of collaborations and activities, Arctis offers special actions and wizards. Arctis provides an editor to specify services, which allows the user to create collaborations from scratch or compose existing ones taken from a library to create composite collaborations. Special actions are available to update each composite building block, which require that the activities and their partitions as well as call behavior actions must be synchronized with the collaboration. For example, Arctis automatically generates a corresponding activity for the behavioral specification of the composition. For each collaboration role, an activity partition is created and each collaboration is represented by a call behavior action with its pins. This skeleton is then completed manually with activity flows and nodes that model the extra logic to couple the sub-collaborations. When a building block is complete and syntactically correct, Arctis transforms the UML activity into TLA+, the language for TLA, and initializes the model checker TLC [30]. TLC can verify a specification for various temporal properties that are stated as theorems. For each activity, a set of theorems is automatically generated, which claims certain properties to be maintained by activities in general. When TLC detects that a theorem is violated, it produces an error trace displaying the state sequence that leads to the violation. However, the automated model verification process with TLC is currently only applicable to UML collaborations and activity diagrams, as missing plug-ins to generate UML deployment diagram and incorporate non-functional parameters to annotate UML model for Arctis are under development.

SHARPE is a tool that accepts specifications of mathematical models and requests for model analysis [3]. It is a tool to specify and analyze performance, reliability, and performability models. It is a toolkit that provides a specification language and solution methods for most of the commonly used model types. Non-functional requirements of the distributed system can also be evaluated using SHARPE, such as response time, throughput, job success probability, etc. The SHARPE tool specifies a SRN model using several sections. First section specifies a place name and the initial number of tokens in the place. Second section specifies a name for a timed transition, a transition type (*ind* if the transition rate is marking-independent and *dep* if it is marking-dependent), a place name if and only if the rate is dependent, and a rate. Third section specifies a name for an immediate transition, a transition type (*ind* if the transition weight is marking-independent and *dep* if it is marking-dependent), a place name if and only if the weight is dependent, and a weight. Fourth section specifies the arcs from places to transitions. The multiplicity indicates the number of tokens that must be present in the place for the transition to fire. Section five specifies the arcs from transitions to places. The multiplicity indicates the number of tokens that are deposited in the place when the transition is fired. Section six specifies inhibitor arcs from places to transitions. The multiplicity indicates how many tokens must be in place to inhibit the transition from firing.

Section seven specifies the built in functions of SHARPE or user-defined functions that can be used to derive result.

Although the Arctis and SHARPE tools have been utilized to describe service definitions using UML collaboration and activity as well as model evaluations, the following steps of our modeling framework have been developed:

- System physical platform using UML deployment diagram
- Deployment mapping
- Generation of annotated UML model
- Automated Model transformation
- Model validation for XML

The steps of the performance modeling framework, such as UML deployment diagram and deployment mapping (see Section 2.2 and 2.5), and performance parameters incorporation into UML models (see Section 2.4) are generated as XML documents. We have defined XML schema files for corresponding XML documents. The XML schema file describes the structure of an XML document that is used to validate the corresponding XML document to ensure that the XML data are in correct format and the XML document is syntactically correct. Hence, erroneous data or typos in the XML document will be fixed during the XML validation and inform users to correct the corresponding data. To ensure this claim, we have defined several constraints and checks in the schema files.

5. Case Study

As a representative example, we consider two case scenarios to show the applicability of our performance modeliWe agree to cancel your articleng framework.

5.1. Clustering of Modules

The first scenario is dealing with heuristically clustering of modules and assignment of clusters to nodes [24], [26], [31]. Even though the scenario is artificial, it is complex enough 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,...,C_{10}) to be deployed and K = 14 collaborations between them illustrated in Fig. 4(a) [26]. 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, and C_9 regarding their location. They must be bound to nodes n_2, n_1, and n_3 respectively [24]. The internal behavior of the collaboration K_i of our example scenario is realized by the call behavior action through the UML activity diagram shown in Fig. 5(b) [1]. The collaboration in Fig. 5(b) is the encapsulated building block that will be stored in a library for later reuse. The detailed behavior of the collaboration role C is realized through UML activity diagram demonstrated in Fig. 5(a) [1] which is mainly composed of the collaborations demonstrated in Fig. 5(b). However, there is no behavior modeled in detail, only those collaborations between processes deployed on different physical nodes. The UML collaboration diagram can be modeled by the activity that may capture the detailed behavior but the level of details must be selected with care in order for the model to scale while generating the performance model [24].

In this example, the target execution environment consists of only N = 3 identical, interconnected nodes with a single provided property, namely processing power and with infinite communication capacities depicted in Fig. 4(b) [1] [24]. The optimal deployment mapping can be observed in Table 2 [26]. The lowest possible deployment cost, according to "(4)" is 17 + 100 + 70 = 187 [26], [31]. In order to annotate the UML diagram in Fig. 4(b) and 5(a), we use the stereotypes *SaStep, ComputingResource,* and *Scheduler*, and the tagged values *execTime, deadline*, and *schedPolicy*, which are already described in Section 2.4 [24].

Collaboration K_i is associated with two instances of *deadline* (see Fig. 5(a)) as collaborations in the example scenario are associated with two kinds of cost: communication cost and overhead cost [1].

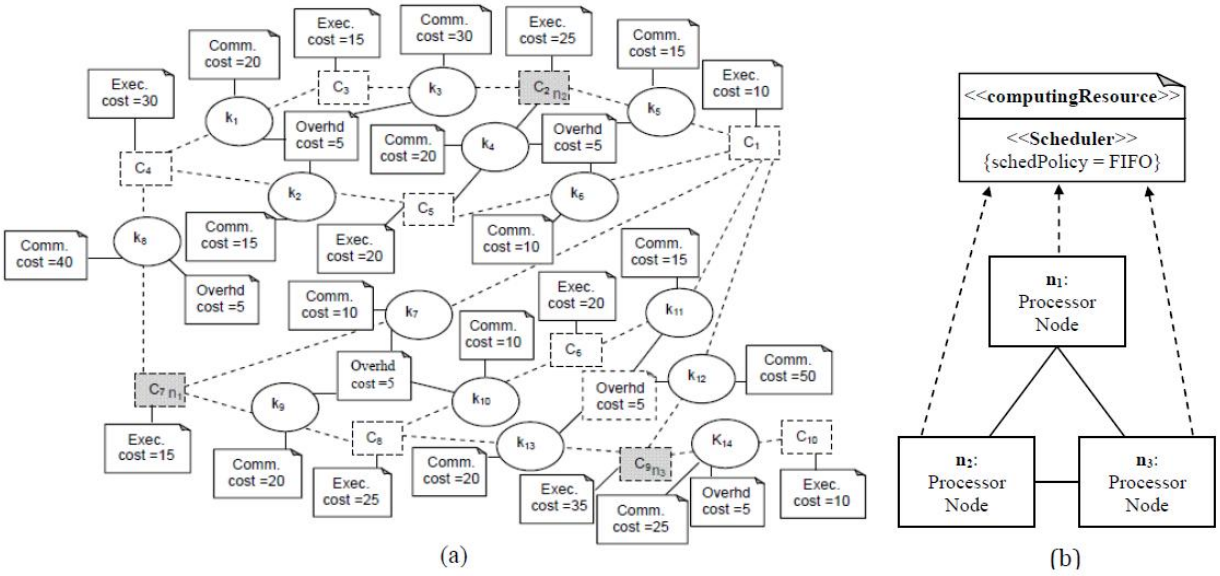


Fig. 4(a). Collaborations and components [26] (b). Deployment diagram of example scenario [1].

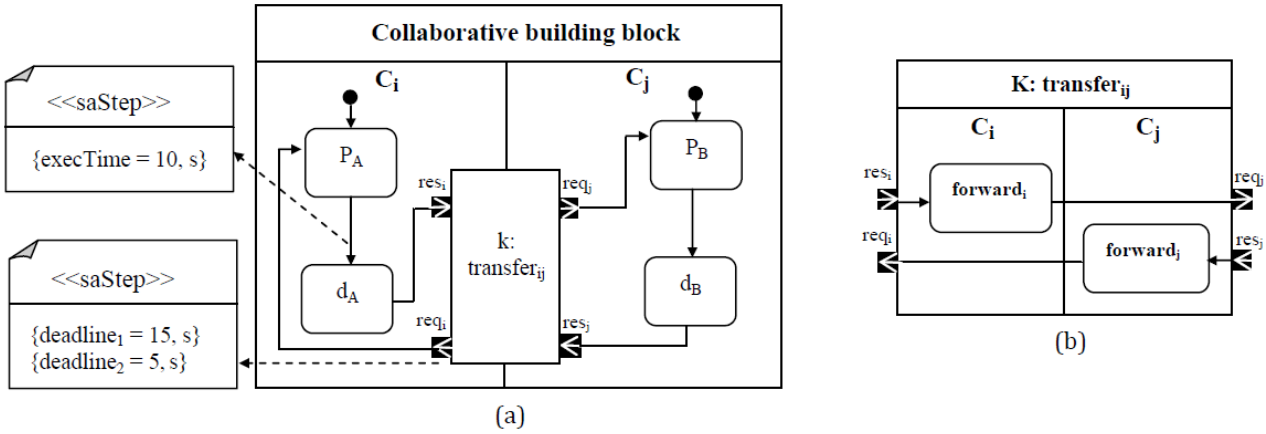


Fig. 5(a). Detail behavior of collaborative building block (b). Internal behavior of collaboration [1].

Table 2. Optimal Deployment Mapping in the Example Scenario [26]

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
$\sum$ cost			117	

By considering the annotated UML model and the model transformation rules, the corresponding SRN model of our example scenario is illustrated in Fig. 6 [24]. Fig. 6 sketches the resulting SRN model by illustrating details of entire places and transitions, which is produced by applying the model transformation rules (see Section 2.6) [24]. According to the transformation rules 1, each collaboration role is defined by the two states P_i and d_i and the passing of token from state P_i to d_i is realized by the timed transition. Initially, there will be a token from place P_1 to P_{10}. According to rule 1, in order to define the upper bound

on the execution of the processes in parallel by a network node, we introduce three places PP_1, PP_2, and PP_3 in the SRN model for the corresponding three physical nodes and initially, these three places will contain q (Here, q = 1) tokens, where q will define the maximum number of the process that will be handled by a physical node at particular time instant. To ensure the upper bound on the execution of the processes in parallel of a network node n_1, we introduce arcs from place PP_1 to transitions 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 firings of transitions t_4, t_7, and t_8 not only depend on the availability of the token in the places 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 on the execution of the processes in parallel of network nodes n_2 and n_3, we introduce arcs from place PP_2 to transitions t_2, t_3, and t_5 and from place PP_3 to transitions t_1, t_6, t_9, and t_{10}. For generating the SRN model from annotated UML model, firstly, we will consider the collaboration roles deploy on the processor node n_1, which are C_4, C_7, and C_8 (see Table 2). Here, component C_7 is connected with C_4 and C_8. The communication cost between the components is 0 but still there is a cost for execution of the background processes. So according to rule 2, after the completion of the state transitions from P_7 to d_7 (states of component C_7), from P_4 to d_4 (states of component C_4) and from P_8 to d_8 (states of component C_8) the states d_7, d_4, and d_7, d_8 are connected by the timed transitions k_8 and k_9 to generate the SRN model. In the same way, the SRN models for collaboration roles deployed on the processor nodes n_2 and n_3 have been generated. In order to generate the system level SRN model, we need to combine the entire three SRN models generated for three processor nodes by considering the interconnection among them. To compose the SRN models of processor nodes n_1 and n_2, states d_4 and d_3 are connected by the timed transition k_1 and states d_4 and d_5 are connected by the timed transition k_2 according to rule 2. Likewise, composition of the SRN models of processor nodes n_2 and n_3, and processor nodes n_1 and n_3 has been achieved [24].

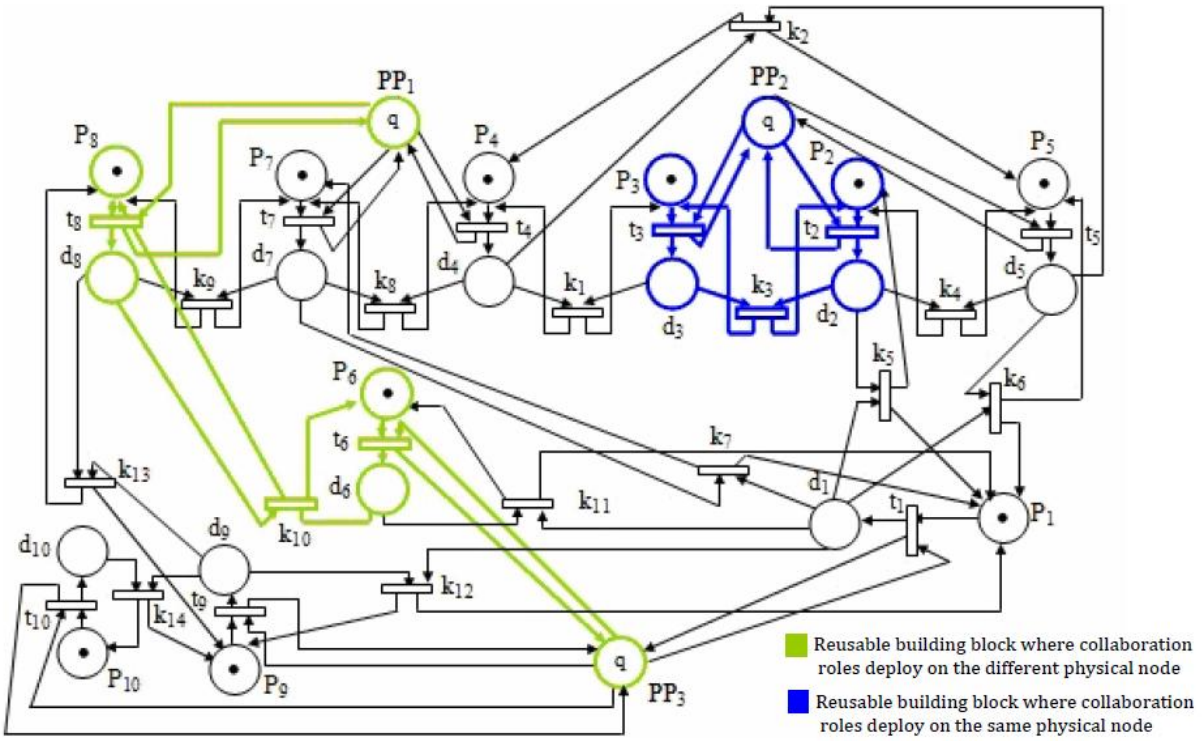


Fig. 6. SRN model of our example scenario [24].

The throughput calculation according to "(7)" (see Appendix A) for the different deployment mappings including the optimal deployment mapping is shown in Table 3 [24]. The optimal deployment mapping

presented in Table 3 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 the approach presented in this work confirms the optimality in every case [24].

Table 3. Deployment Mapping in the Example Scenario Along with Throughput [24]

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 (mini)	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

5.2. Taxi Control System

Another representative example has been considered, Taxi control system, where several taxis are connected to a control centre and update their status (busy or free). The control centre accepts the tour orders from clients via SMS or mobile call. The orders are processed by the call centre, which sends out tour requests to the taxis. Fig. 7 illustrates the scenario as the UML collaboration [1]. Participants in the service are represented by the collaboration roles taxi, control centre, and client. The control centre has a default multiplicity of one, whereas there can be many taxis and clients in the system denoted by multiplicity [1,…,*]. Between the roles, the collaborations denote the occurrence of the behavior: the taxi and control centre are interacting with collaboration *status update & tour request*, and the control centre is cooperating with the client by means of collaboration *tour order & notify*, whereas the interaction between the taxi and client is realized by the collaboration *start tour*. The internal behavior of the collaborations *status update & tour request*, *tour order & notify*, and *start tour* are demonstrated using UML activity, which are shown in Fig. 8 [1]. The specifications for the collaborations are given as coherent, self-contained building blocks and have one activity partition for each collaboration role.

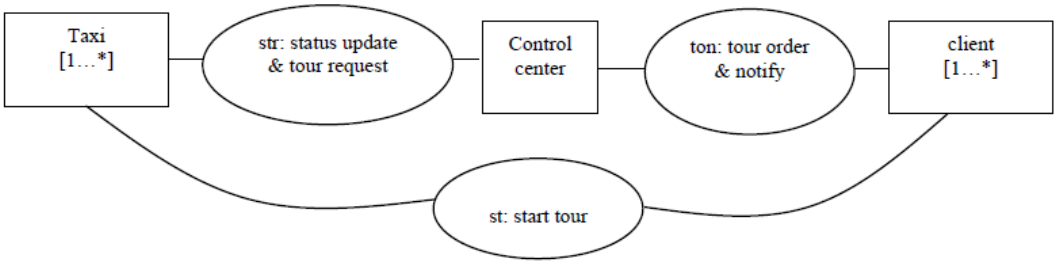


Fig. 7. Collaborations and components in the taxi control system [1].

For composition of the building blocks to delineate the detailed behavior of the taxi control system, activity parameter nodes are used that can be connected to other elements. For each activity parameter node of the referred activity, a corresponding pin is declared. There are different types of pins (activity parameter nodes) illustrated on the building blocks such as starting pin, streaming pin, and terminating pin. The pins shown in Fig. 8 (res_T, req_C, res_{C1}, req_T, etc.) are all streaming pins, which pass tokens throughout

the active phase of the building blocks and are used to connect the building blocks to delineate the detailed behavior of the taxi control system [1]. The detailed behavior of the taxi control system is shown in Fig. 9 [1], which is mainly composed of the collaborations demonstrated in Fig. 8. When a new taxi arrives or a busy taxi becomes free after completing the tour, the taxi performs a log-in operation into the system and set that taxi status to free. Then, the control centre will be notified of the status update. The control centre is responsible for adding the taxi in the free taxi queue. When there is a taxi available in the free taxi queue, the control centre sends the tour order information to the free taxi if there is any pending tour order. After receiving the tour order information, the taxi notifies the control centre of its acceptance of the request. The control centre adds the taxi into the busy taxi pool and notifies the taxi about the changing of its status. The taxi then performs the system log-off operation and checks the tour order to determine whether the client is still waiting for a taxi or not. Based on the results given by the control centre, the taxi conducts the tour and then again performs log-in operation into the system and changes its status. The control centre receives notification about any client request. The control centre is responsible for adding the request in the queue. After receiving a request from the client, there might be two possibilities in the control centre (which is realized by the decision node dec): either the control centre looks for an available taxi, or the request might be cutoff because of the number of client requests exceed the capacity of the control centre to handle the client requests. If the control centre locates an available taxi, it notifies the user about the availability of taxi, but the number of client requests will be cutoff if it exceeds the capacity of the control centre. When a taxi is ready to conduct the tour and a client is waiting for the taxi, the taxi will start the tour. When the tour finishes, the taxi becomes free and is then ready to pick up another client.

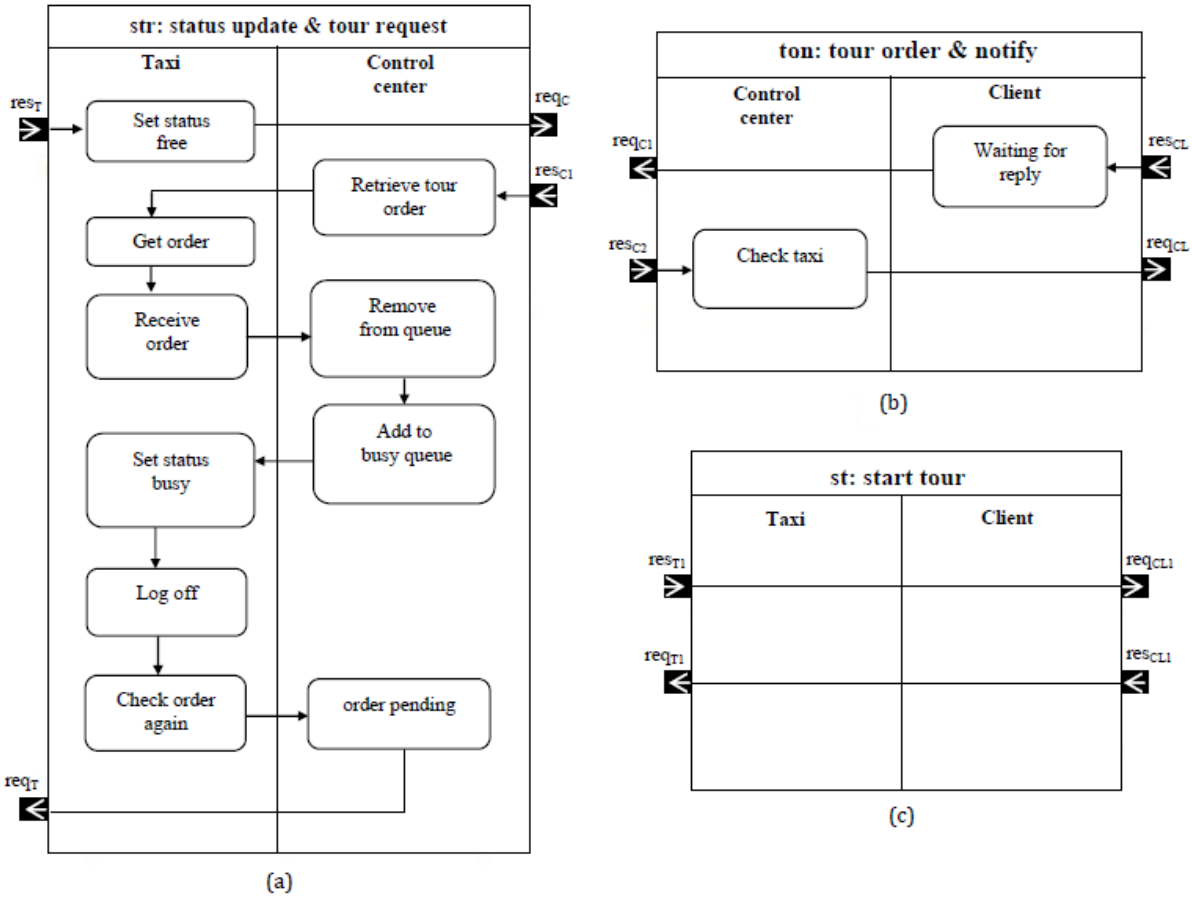


Fig. 8. Internal behavior of the collaboration using UML activity of taxi control system [1].

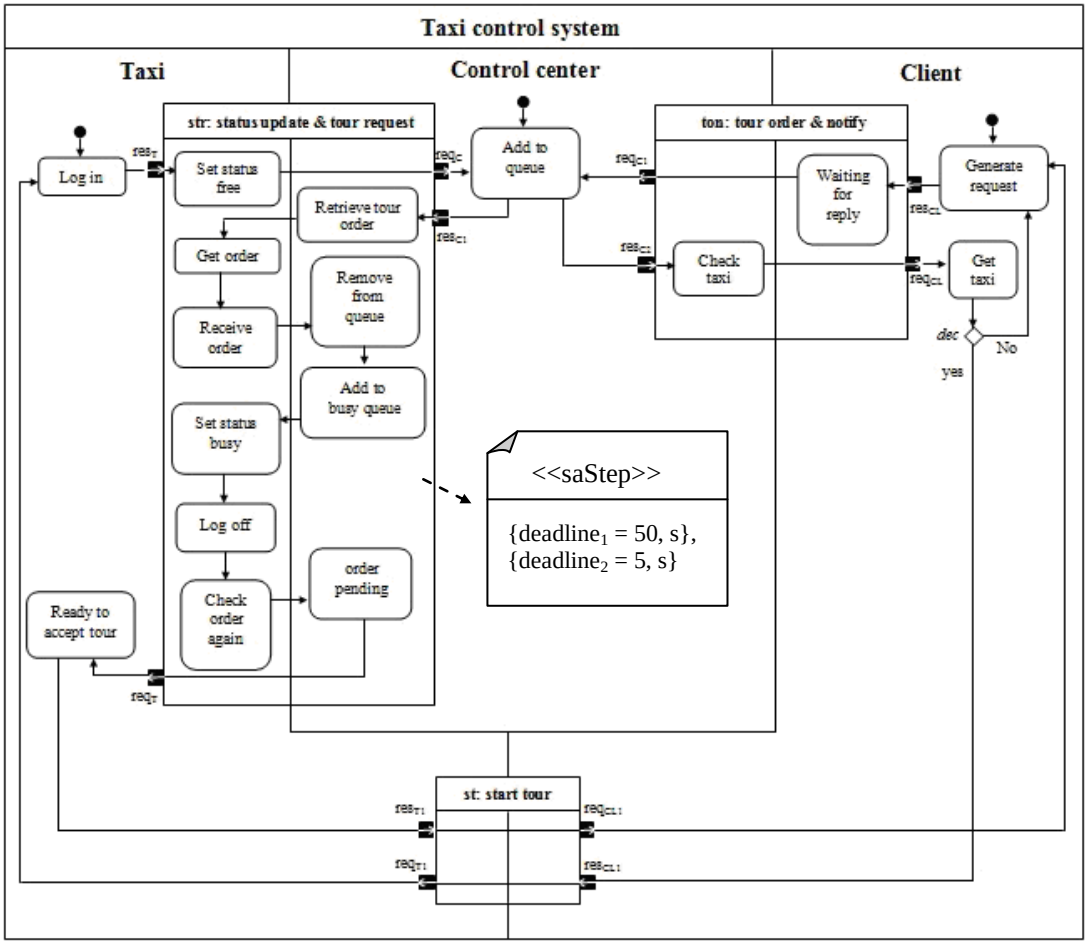


Fig. 9. Detailed illustration of the service behavior using UML activity [1].

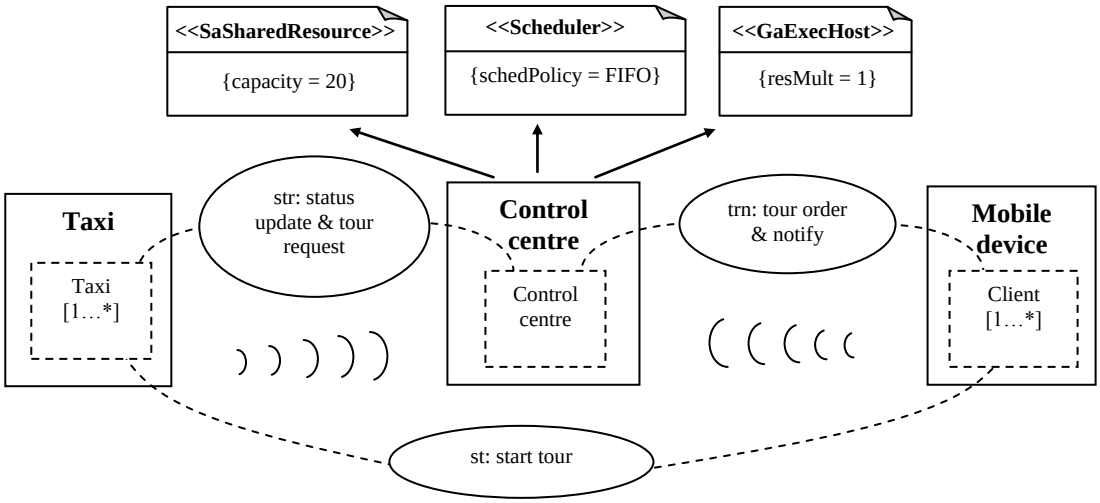


Fig. 10. Deployment diagram and deployment mapping of a taxi control system [1].

UML deployment diagram for the taxi control system is illustrated in Fig. 10 [1], where the system components taxi and the user mobile device are connected with the control centre via a wireless communication channel.

Annotation of the UML model for the taxi control system is illustrated in Fig. 9 and Fig. 10 [1]. In Fig. 9, the annotated activity diagram is defined as follows: after being deployed in the execution environment, communication between taxi and control centre is achieved in 50 sec, whereas the overhead time to

conduct this communication is 5 sec, which is annotated using the stereotype *SaStep* and two instances of the tagged value *deadline* (according to the UML profile for MARTE) – $deadline_1$ defines the communication time, and $deadline_2$ is used for overhead time. The communication between the client and the control centre and the communication between the taxi and the client can be annotated in the same manner as above. Annotation of UML deployment diagram of the taxi control system according to the MARTE profile is demonstrated in Fig. 10. The control centre is connected with the client and taxi using a wireless communication channel, where the tagged value *schedPolicy* specifies that the control centre follows a FIFO scheduling policy to serve the queued jobs. Moreover, the tagged value *resmult* indicates that the maximum number of instances of the resource control centre is 1, and the tagged value *capacity* indicates that the maximum number of permissible concurrent users handled by the control centre is 20.

Fig. 10 also demonstrates the deployment mapping of a taxi control system, where it is a straightforward and one-to-one mapping between the service components and the physical components [1].

We will illustrate how the model transformation rules have been utilized to generate the performance SRN model for the example taxi control system. Generation of an analytical model (SRN) for the performance evaluation of the taxi control system by applying the model transformation rules is demonstrated in Fig. 11 [1]. Considering the deployment mapping of the taxi control system (see Fig. 10), UML models with annotations (see Fig. 9 and Fig 10), and the model transformation rules (see Section 2.6) the SRN model of the taxi control system has been generated. The collaborative building block *status update & tour request* is transformed into a SRN model according model transformation rule 2 (see Fig. 3). The generated places and transition from collaborative building block *status update & tour request* are P_{li}, P_{afq}, P_{rat}, and t_{tr}. The timed transition t_{tr} is realized by both the communication time and overhead time, as the collaboration roles taxi and control centre are deployed on different physical nodes. Likewise, the collaborative building block *tour order & notify* is transformed into a SRN model according to model transformation rule 2 (see Fig. 3). The generated places and transition from collaborative building block *tour order & notify* are P_{gr}, P_{afq}, P_{gt}, and t_{to}. The timed transition t_{to} is realized by the communication time and overhead time, as the collaboration roles client and control centre are deployed on different physical nodes. Client activity partition of the *tour order & notify* collaborative building block also contains a decision node, which is transformed into a SRN model according to the decision node transformation rule (see Table 1). Two flows are created from place P_{gt}: one towards the immediate transition it_1 and another towards the timed transition t_{st}. The immediate transition it_1 is realized by a guard function [gr], which is only enabled when the client requests exceed the capacity of the control centre. The definition of guard function [gr] is given as: [If (#(P_{gt}) > n) 1 else 0], where (#(P_{gt})) defines the number of client requests that have arrived in the system and 'n' defines the capacity of the control centre. In addition, the collaborative building block *start tour* is transformed into a SRN model according to model transformation rule 2 (see Fig. 3), where the timed transition t_{st} is realized by the communication time and overhead time.

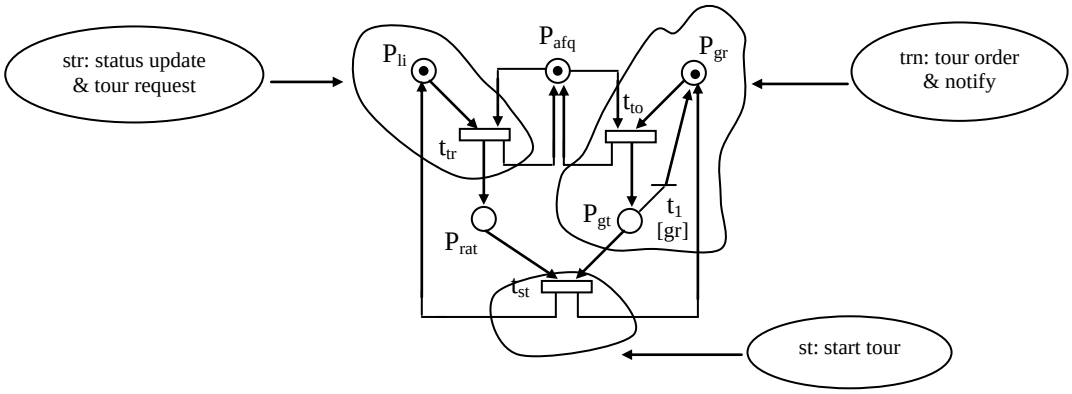


Fig. 11. Performance SRN model of taxi control system [1].

The obtained SRN model for the taxi control system has been solved to calculate the mean response time for serving client requests (using little's law [32]) for various numbers of client and taxi combinations. The result is shown in the graph presented in Fig. 12 (Here, the rate of each timed transition in the SRN model is considered 0.033) [1]. The top curve shows the mean response time for serving a client request when there is 1 taxi available in the system, and it also focuses on how the mean response time for individual client requests increases gradually with an increasing number of clients in the system. This curve shows the mean response time of around 520 sec for serving a client request for over 20-25 customers and 1 taxi available in the system. The response time for serving a client request depends not only on the available taxis but also on the number of client requests arrived in the system. For fixed number of taxis available in the system, if the number of client requests increases in the system the response time for serving a client request will be higher. Again for fixed number of client requests, the response time will be lower with the increasing number of taxis in the system. The middle and bottom curves show the mean response time for serving a client request when there are 5 and 20 taxis available in the system, respectively. It is clearly shown that the response time increases with the higher number of customers in the system. But all 3 graphs show the cutoff point when the number of clients equals 20, as in our example, we assume that the control centre capacity for handling concurrent client's request at particular time instant is 20. It is not logical that the control centre would be able to handle unlimited number of clients at particular time instant. However, it is possible to solve the SRN model for a large number of client and taxi combinations.

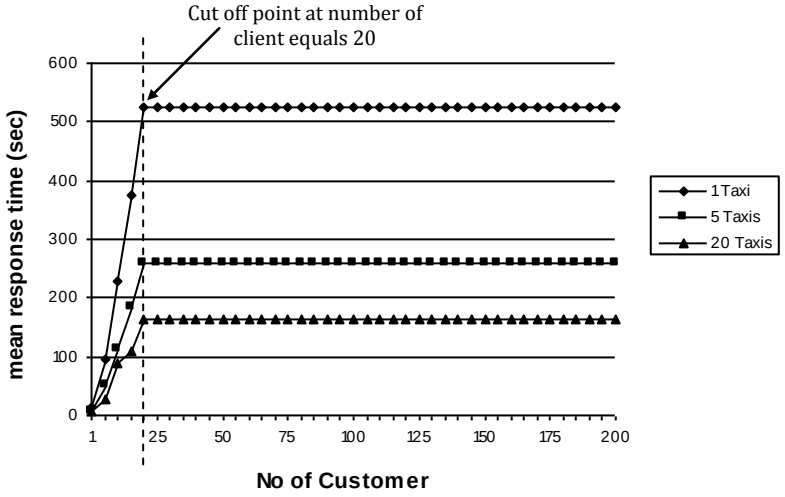


Fig. 12. Numerical results of the analytical model (SRN) of the taxi control system.

6. Conclusion

The examples we considered as case studies and presented in this paper were chosen to cover real scenario and also included the well-known problem of assigning clusters to nodes as an artificial case study. The examples were compact enough in some respects to be completely presented within the space constraints of the article. We claim, however, that the modeling approach scales well and can also handle specimens of real system specifications. Moreover, we might expect more complex forms of collaborations than those demonstrated in the paper, which can be solved with additional levels of simplification. In addition, our provided deployment logic can handle any properties of the service as long as a cost function for the specific property can be produced. The defined cost function can react in accordance with the changing size of the search space of an available hosts presented in the execution environment to assure an efficient deployment mapping. For the model transformation, we have described generalized rules to handle complexity and scalability. Model transformation rules have been defined based on the

encapsulated self-contained reusable UML building blocks that can describe complex behavior and interaction of the system participants. The novelty of the model transformation process is that the encapsulated building blocks have been utilized as the input in the model transformation process to generate SRN models instead of considering single component of the UML diagram as input, which solves the state explosion problem [1].

While performing the model transformation, the UML diagrams such as collaboration, activity, and deployment are formalized using cTLA that captures the behavior of the collaborative system specification. We apply cTLA as a foremost background technique to understand the formalism of collaborative specifications expressed in other languages, such as UML. The provision of a formal semantics does not end in itself but, is a vital part for the automated model transformation. It is also basis for meaningful semantic checks as, for instance, to be done with the model checking approach introduced in [30].

The extensibility of our modeling approach can be defined in several directions, such as developing missing plug-ins for Arctis to define UML deployment diagram and to generate annotated UML model, permitting the deployment logic to dynamically provide new configurations of the deployment mapping when changes in the execution environment or workload configuration are encountered, considering more comprehensive scenarios for large networks including more physical nodes and clusters, developing an automated feedback method to find UML anti-patterns and then change the functional design accordingly, comparing numerous execution environments, and to find the optimal deployment mapping of service components over a physical environment.

Appendix A: Performance Model Evaluation

We focus on measuring the throughput of the system from the SRN model. We are interested in throughput calculation, which can be defined as a measure of jobs that a system can process in a given time period. This in turn justifies the efficiency of our deployment logic mentioned in section 2.5 in accordance with system performance evaluation. Before deriving formula for throughput estimation, we consider several assumptions that will allow us determining the parameters necessary for the throughput calculation of the system:

- Executions of the processes occur independently of each other
- Communication between processes is independent
- Communication between interconnected nodes will be started following entire completion of the processing and communication inside each physical node

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 [24]. The value of $E(N)$ is calculated by solving the SRN model using SHARPE [3]. Cost of the network, C_{net} is defined below. At first, the cost of a subnet (C_{sn}) will be calculated as follows:

$$C_{sn_x} = \sum_{i=1}^{|m|} f_{c_i} + \max\left\{ f_{B_j} + I(q_0(\mathbf{M}, k_{j,1}) \neq q_0(\mathbf{M}, k_{j,2}))\, f_{k_j} \right\}$$
$$= \sum_{i=1}^{|m|} f_{c_i} + \max\left\{ f_{B_j} \right\} \quad (5)$$

where

- C_{sn_X} = cost of the x^{th} subnet (where x = 1….n; n is the total number of subnet that comprises the network)
- f_{c_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_{B_j} = overhead cost of collaboration j (where j =1....n; n is the total number of collaboration in the x^{th} subnet)
- f_{k_j} = communication cost of collaboration j (where j =1....n; n is the total number of collaboration in the x^{th} subnet)
- $I(q_0(\mathbf{M}, k_{j,1}) \neq q_0(\mathbf{M}, k_{j,2})) = 0$ (according to Section 2.5); 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.

$$C_{snp_y} = \left\{ \max(C_{sn_a}, C_{sn_b}) + (f_{B_j} + I(q_0(\mathbf{M}, k_{j,1} \neq q_0(\mathbf{M}, k_{j,2}) f_{k_j}) \right\}$$
$$= \left\{ \max(C_{sn_a}, C_{sn_b}) + (f_{B_j} + f_{k_j}) \right\} \quad (6)$$

where

- C_{snp_y} = cost of the y^{th} subnet pair (y = 1....n; n is the total number of subnet pairs in the network, where each subnet pair corresponds between two subnets)
- C_{sn_a}, C_{sn_b} = cost of the a^{th} and b^{th} subnet (where (a, b) ϵ x and $a \neq b$)
- f_{B_j} = overhead cost of collaboration j (where j =1....n; n is the total number of collaboration between the subnet pair (C_{sn_a}, C_{sn_b}))
- f_{k_j} = communication cost of collaboration j (where j =1....n; n is the total number of collaboration between the subnet pair (C_{sn_a}, C_{sn_b}))
- $I(q_0(\mathbf{M}, k_{j,1}) \neq q_0(\mathbf{M}, k_{j,2})) = 1$ (according to Section 2.5); as in this case, processes connected by the collaboration deploy on the different physical nodes.

So Cost of the network:

$$C_{net} = \max\left\{ C_{snp_1}, ..., ..., C_{snp_n} \right\}$$

Finally, the throughput is:

$$\frac{E(N)}{C_{net}} \quad (7)$$

References

[1] Khan, R. H. (2014). Performance and performability modeling framework considering management of service components deployment. PhD thesis, Norwegian University of Science and Technology.

[2] Kraemer, F. A. (2013). ARCTIS, Retrieved May, 2013, from http://arctis.item.ntnu.no.

[3] Trivedi, K. S., & Sahner, R. (2012). Symbolic hierarchical automated reliability / performance evaluator (sharpe), Duke University.

[4] Lopez, G., Merseguer, J., & Campos, J. (2004). From UML activity diagrams to SPN: application to software performance engineering. *Proceedings of the 4th International Conference on Software and Performance.*

[5] Cooper, K., Dai, L., & Deng, Y. (2003). Modeling performance as an aspect: A UML based approach. *Proceedings of the 4th Workshop on AOSD Modeling with UML*.

[6] Moura, P., Borges, R., & Mota, A. (2003). Experimenting formal methods through UML. *Proceedings of the Brazilian Workshop on Formal Methods*.

[7] Warmer, J., & Kleppe, A. (1999). *The Object Constraint Language: Precise Modeling with UML*. Addison-Wesley.

[8] Theelen, B. (2004). *Using the SHE method for UML-based performance modeling.* System specification and design languages.

[9] Yin, Y. *et al*. (2010). The integrated application based on real-time extended UML and improved formal method in real-time embedded software testing. *Journal of Networks*, *5(12)*, 1410-1416.

[10] Merayo, M. G., Nu′n˜ez, J., & Rodrı′Guez, I. (2008). *Formal Testing from timed finite state machines*. *Computer Networks*, *52*, 432–460.

[11] Andolfi, F. *et al*. (2000). Deriving performance models of software architecture for message sequence charts. *Proceeding of the 2nd International Conference on Software and Performance* (pp. 45-57).

[12] Cardoso, J. & Blanc, C. (2001). Ordering actions in sequence diagram of UML. *Proceedings of the International Conference on Information Technology Interfaces* (pp. 19-22).

[13] Eshuis, R., & Wieringa, R. (2001). A comparison of petri net & activity diagram variants. *Proceedings of the Collaboration on Petri Net Technologies for Modeling Communication Based Systems*.

[14] Petriu, D. & Shen, H. (2001). Applying the UML performance profile: Graph grammar based deviation of LQN models from UML specifications. *Tools*, pp. 14-17.

[15] Tigris. (2012). *ArgoUML*. Retrieved 19 November, 2012, from http://argouml.tigris.org.

[16] Marzolla, M. and Balsamo, S. (2004). UML-PSI: The UML performance simulator. *Proceedings of the International Conference on Quantitative Evaluation of Systems* (pp. 340-341).

[17] Akehurst, D. *et al*. (1999). Predictive performance analysis for distributed systems–PERMABASE. *Proceedings of the Workshop on Software Performance Predication Extracted from Design.*

[18] Wet, N. D., & Kritzinger, P. (2005). *Using UML models for the performance analysis of network system. Journal of Computer Networks*, *49(5)*, 627-642.

[19] Petriu, C. *et al*. (2005). Performance by Unified Model Analysis (PUMA). *Proceedings of the 5th International Conference on Software and Performance* (pp. 1-12).

[20] OMG. (2009). *UML: Superstructure*. Version-2.2.

[21] Kraemer, F. A. (2008). Engineering reactive systems: A compositional and model-driven method based on collaborative building blocks. PhD thesis, Norwegian University of Science and Technology.

[22] Kraemer, F. A. & Herrmann, P. (2006). Service specification by composition of collaborations-an example*. Proceedings of the Web Intelligence – Intelligent Agent Technology Workshops* (pp. 129-133).

[23] OMG. (2009). UML profile for MARTE: Modeling and Analysis of Real-Time Embedded Systems. Version – 1.0.

[24] Khan, R. H., Heegaard, P. E., & Ullah, K. W. (2012). Performance evaluation of distributed system using SPN.

[25] CORBA M. J. (2012). Cost-efficient deployment of distributed software services. PhD thesis, Norwegian University of Science and Technology.

[26] Csorba, M. J., Heegaard, P. E. & Herrmann, P. (2008). Cost-efficient deployment of collaborating components. *Proceedings of the 8th IFIP International Conference on Distributed Applications and*

Interoperable Systems (pp. 253–268).
[27] Khan, R. H., & Heegaard, P. E. (2011). A Performance modeling framework incorporating cost efficient deployment of multiple collaborating components. *Proceedings of the 2nd International Conference on Software Engineering and Computer Systems* (pp. 31-45).
[28] Ciardo, G., Muppala, J., and Trivedi, K. S. (1992). Analyzing Concurrent and Fault-Tolerant Software Using Stochastic Reward Nets. *Journal of Parallel and Distributed Computing*.
[29] Kraemer, F. A. & Herrmann, P. (2007). Formalizing collaboration-oriented service specifications using temporal logic. *Proceedings of the International Conference on Networking and Electronic Commerce Research Conference* (pp. 194-220).
[30] Slåtten, V. (2007). *Model checking collaborative service specifications in TLA with TLC.* Project Thesis, Norwegian University of Science and Technology.
[31] Efe, K. (1982). Heuristic models of task assignment scheduling in distributed systems.
[32] Trivedi, K. S. (2001). *Probability and statistics with reliability, queuing and computer science application*, Wiley- Interscience publication.

Razib Hayat Khan completed his PhD at Department of Telematics, Norwegian University of Science and Technology (NTNU), Norway. He completed his M.Sc. in information and communication systems security specialized in security in open distributed system from Royal Institute of Technology (KTH), Sweden in 2008. He worked as a visiting researcher at Duke University, Durham, USA and served as a research engineer, doing multimedia technologies at Ericsson AB, Sweden. He also worked under VRIEND project as part of his M.Sc. thesis which was sponsored by Philips, AkzoNobel, Corus, and DSM. He received his B.Sc. degree in computer science and information technology from Islamic University of Technology (IUT), Gazipur, Bangladesh in 2004. He served as a lecturer in Stamford University, Dhaka, Bangladesh during the period November 2004 – August 2006. His research interests include software performance and dependability modeling, Information systems security. At present he is working with performance and dependability issues in Communication system.

Poul E. Heegaard is currently serving as a professor at Department of Telematics, Norwegian University of Science and Technology (NTNU). He completed his PhD from department of Telematics, NTNU in 1998. He served as an associate professor at Department of Telematics, NTNU, in adjunct position (20%) during 1999-2003, reduced position (50%) during 2003-2006, and full time position since 2006. He was research scientist during 1989-1999 and senior research scientist from 1999 at SINTEF Telecom and Informatics. He also served as a research scientist during 1999-2001 and senior scientist during 2001-2009 at Telenor Research and Development. He was a visiting researcher (March, 2007) hosted by Dr. Sandmann at University of Bamberg, Germany. He was also served as a visiting researcher during Sep 2007 – July 2008 hosted by Prof. Trivedi at Duke University, Durham, NC. He acquired the position of Departmental head, department of Telematics, NTNU during 2009-2013. His research interests include QoS evaluation and management in communication systems and services, rare event simulation techniques, survivability quantification, monitoring, routing and management in dynamic networks, distributed, autonomous and adaptive management and routing.