A Performability Modeling Framework Considering Service Components Deployment

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

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

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

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

***Abstract*- The analysis of the system behavior from the pure performance viewpoint tends to be optimistic since it ignores failure and repair behavior of the system components. On the other hand, pure dependability analysis tends to be too conservative since performance considerations are not taken into account. The ideal way is to conduct the modeling of performance and dependability behavior of the distributed system jointly for assessing the anticipated system performance in the presence of system components failure and recovery. However, design and evaluation of the combined model of a distributed system for performance and dependability analysis is burdensome and challenging. Focusing on the above contemplation, we introduce a framework to provide tool based support for performability modeling of a distributed software system that proposes an automated transformation process from the high level Unified Modeling Language (UML) notation to the Stochastic Reward Net (SRN) model and solves the model for early assessment of a software performability parameters. UML provides enhanced architectural modeling capabilities but it is not a formal language and does not convey formal semantics or syntax. We present the precise semantics of UML models by formalizing the concept in the temporal logic compositional temporal logic of actions (cTLA). cTLA describes various forms of actions through an assortment of operators and techniques which fit excellently with UML models applied in this work and also provides the support for incremental model checking. The applicability of our framework is demonstrated in the context of performability modeling of a distributed system to show the deviation in the system performance against the failure of system components.**

Keywords: UML; SRN; Performability; Deployment; Reusability

I. INTRODUCTION

Conducting performance modeling of a distributed system separately from the dependability modeling fails to asses the anticipated system performance in the presence of system components failure and recovery. System dynamics is affected by any state changes of the system components due to failure and recovery. This introduces the concept of performability that considers the behavioral change of the system components due to failures and also reveals how this behavioral change affects the system performance. But to design a composite model for a distributed system, perfect modeling of the overall system behavior is essential and sometimes very unwieldy. A distributed system behavior is normally realized by the several objects that are physically disseminated. The overall system behavior is maintained by the partial behavior of the distributed objects of the system [14]. So it is essential to model the distributed objects behavior perfectly for appropriate demonstration of the system dynamics and to conduct the performability evaluation [14]. Hence, we adopt UML collaboration, state machine, deployment, and activity oriented approach as UML is the most commonly used specification language which models both the system requirements and qualitative behavior through an assortment of notations [5] [14]. The way we utilize the UML collaboration and activity diagram to capture the system dynamics, provides the opportunity to reuse the software components. The specifications of collaboration are given as coherent, self-contained building blocks [14]. Reusability of the software component is achieved by designing the collaborative building block which is used as main specification unit in this work. Collaboration with help of activity diagram illustrates the complete behavior of a software system which includes both the local behavior among the participants and necessary interactions among them. Moreover, for specifying deployment mapping of service components, the performability modeling framework considers system execution architecture through UML deployment diagram. Considering system execution architecture while designing the framework resolves the bottleneck of the deployment mapping of service components by revealing a better allocation of service components to the physical nodes [13]. This requires an efficient approach to deploy the service components on the available hosts of a distributed environment to achieve preferably high performance and low cost levels [14]. Later on, UML State machine (STM) diagram is employed in this framework to capture system components behavior with respect to failure and repair events.

In order to guarantee the precise understanding and correctness of the model, the approach requires formal reasoning on the semantics of the language used and to maintain the consistency of the models specification. Temporal logic is a suitable option for that. In particular, the properties of super position supported by cTLA [19] make it possible to describe systems from different view points by individual processes that are superimposed. In this work, we focus on the cTLA that allows us formalizing the collaborative service specifications given by UML activities and also to define the formal semantics of the UML

deployment diagram and STM model precisely. By expressing collaborations as cTLA processes, we can ensure that a composed service maintains the properties of the individual collaborations it is composed of. The semantic definition of collaboration, activity, deployment, and STM model in the form of temporal logic is implemented as a transformation tool [20] which produces TLA^+ modules. These modules may then be used as input for the model checker TLC for syntactic analysis [20].

Furthermore, UML models are annotated according to the *UML profile for MARTE* [7] and *UML profile for Modeling Quality of Service and Fault Tolerance Characteristics* [13] to include quantitative system parameters necessary for performability evaluation. UML specification styles are applied to generate the SRN model automatically following the model transformation rules where model synchronization between the performance and dependability SRN model is achieved by defining guard functions (a special property of the SRN model [6]). This synchronization thus helps to properly model the system performance with respect to any state changes in the system due to components failure [1] [2].

Over decades several performability modeling techniques have been considered such as Markov models, SPN (Stochastic Petri Nets) and SRN [4]. Among all of these, we will focus on the SRN as performability model generated by our framework due to its 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 [6].

Several approaches have been pursued to accomplish a performability analysis model from a system design specification. Sato et al. develop a set of Markov models, for computing the performance and the reliability of Web services and detecting bottlenecks [9]. Another initiative focuses on model-based analysis of performability of mobile software systems by proposing a general methodology that starts from design artifacts expressed in a UML-based notation. Inferred performability models are formed based on the Stochastic Activity Networks notation [10]. Subsequent effort proposes a methodology for the modeling, verification, and performance evaluation of communication components of a distributed application building software which translates UML 2.0 specifications into executable simulation models [11]. Gonczy et al. mentioned a method for high-level UML models of service configurations captured by a UML profile dedicated to service design; performability models are derived by automated model transformations for the PEPA toolkit in order to assess the cost of fault tolerance techniques in terms of performance [12]. However, most of the existing approaches do not consider the fact of how to conduct the system modeling to delineate system functional behavior while generating the performability model using reusable software components. The framework introduced in this work is superior to the existing approaches that have been realized by UML specification style as reusable building block to characterize a system dynamics. The purpose of the reusable building block is twofold: to express the local behavior of several components and to capture the interaction between them. This provides the excellent opportunity to reuse the building blocks, as the interaction among the several components can be encapsulated within one self-contained building block [14]. This reusability provides the means to design a new system's behavior rapidly utilizing the existing building blocks according to the specification. This helps to start the development process from scratch which in turn facilitates the swelling of productivity and quality in accordance with the reduction in time and cost [2]. Moreover, the ensuing deployment mapping given by our framework has greater impact to satisfy QoS requirements provided by the system. The target in this work is to deal with vector of QoS instead of confining them in one dimension. Our provided deployment logic is definitely capable of handling any properties of the service as long as a cost function for the specific property can be produced. The defined cost function is able to react in accordance with the changing size of search space of available hosts presented in the execution environment to assure an efficient deployment mapping [14]. In addition, the separation of performance and dependability modeling view and the introduction of model synchronization to synchronize the two views activities using guard functions relinquishes the complex and unwieldy affect in performability modeling and evaluation of large and multifaceted systems [1].

The objective of this paper is to provide a tool based support for the performability modeling of a distributed system to allow modeling of the performance and dependability related behavior in a combined and automated way. This in turn allows not only to model functional attributes of the service provided by the system but also to investigate dependability attributes to reflect how the changes in the dependability attributes affect the system overall performance. For ease of understanding the complexity behind the modeling of performability attributes, our modeling framework works in two different views such as performance modeling view and dependability modeling view. The framework achieves its objective by maintaining harmonization between performance and dependability modeling view with the support of model synchronization. The paper is organized as follows: Section II introduces our performability modeling framework, Section III depicts UML model description, Section IV describes formalization of UML models, Section V explains service components deployment issue, Section VI clarifies UML models annotations, Section VII delineates model transformation rules, Section VIII introduces the model synchronization mechanism, Section IX describes the hierarchical method for mean time to failure (MTTF) calculation, Section X indicates the tool based support of the modeling framework, Section XI illustrates the case study, and Section XII delineates the concluding remarks with future directions.

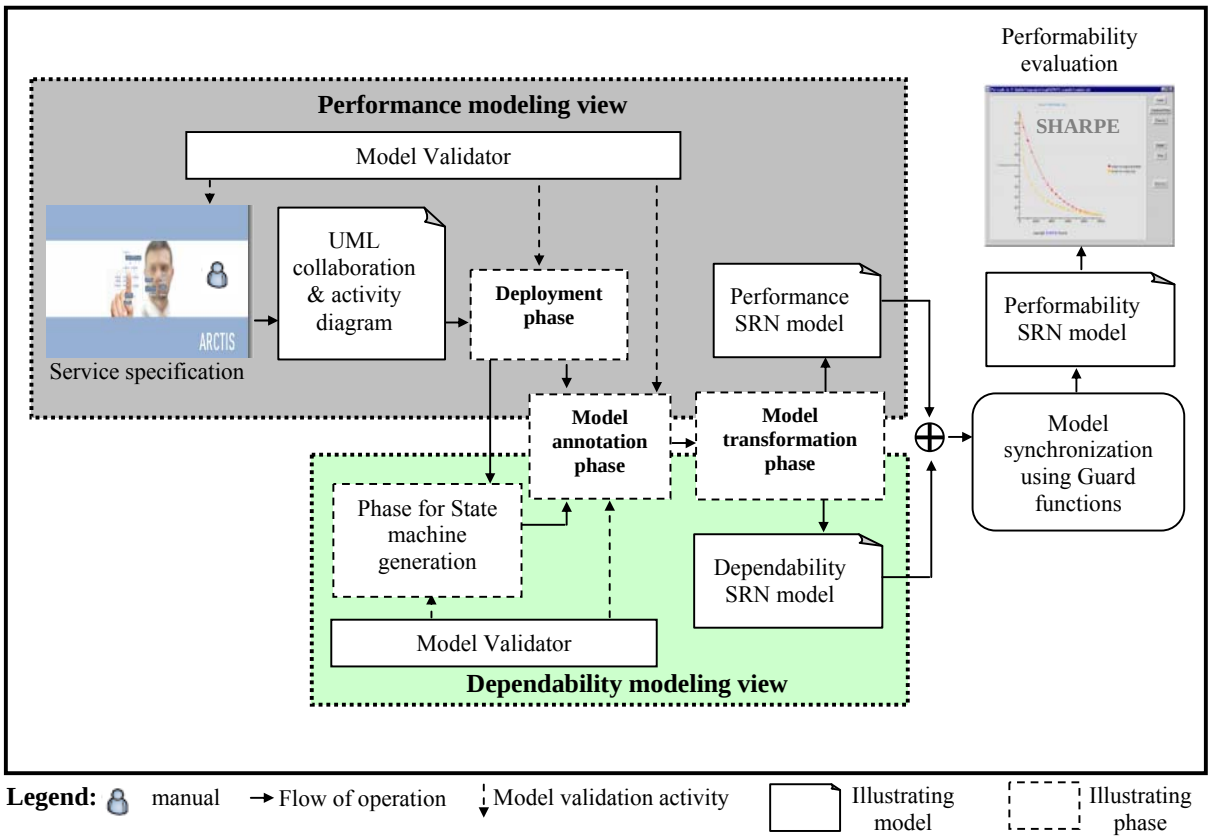


Figure 1. Proposed performability modeling framework

II. OVERVIEW OF PROPOSED FRAMEWORK

Our performability framework is composed of 2 views: performance modeling view and dependability modeling view. The performance modeling view mainly focuses on capturing the system's dynamics to deliver certain services deployed on a distributed system. The performance modeling view is divided into 4 steps shown in Fig. 1 where the service specification step is the part of Arctis tool suite which is integrated as plug-ins into the eclipse IDE [15]. Arctis focuses on the abstract, reusable service specifications that are composed of UML 2.2 collaborations and activities [15]. It uses collaborative building blocks to create comprehensive services through composition. In order to support the construction of building block consisting of collaborations and activities, Arctis offers special actions and wizards.

In the first step of performance modeling view, a developer consults a library to check if an already existing basic building block or collaboration between several blocks solves a certain task. Missing blocks can also be created from existing building blocks and stored in the library for later reuse. The building blocks are expressed as UML models. The structural aspect, for example the service components and their multiplicity, is expressed by means of UML 2.2 collaborations. For the detailed internal behavior, UML 2.2 activities have been used. The building blocks are combined into more comprehensive service by composition to specify the detailed behavior of how the different events of collaborations are composed. For this composition, UML collaborations and activities are used complementary to each other [15]. In the deployment phase, the deployment diagram of our proposed system is delineated and the relationship between system components and collaborations is outlined to describe how the service is delivered by the joint behavior of the system components. In the model annotation phase, performance information is incorporated into the UML activity diagram and deployment diagram according to the *UML profile for MARTE* [8]. The model transformation phase is devoted to automate generation of a SRN model following the model transformation rules. The SRN model generated in this view is called performance SRN.

The dependability modeling view is responsible for capturing any state changes in the system because of failure and recovery behavior of system components. The dependability modeling view is composed of three steps shown in Fig. 1. In the first step, UML STM diagram is used to describe the state transitions of software and hardware components of the system to capture the failure and recovery events. In the model annotation phase, dependability parameters are incorporated into the STM diagram according to *UML profile for Modeling Quality of Service and Fault Tolerance Characteristics & Mechanisms Specification* [13]. The model transformation phase reflects the automated generation of the SRN model from the STM

diagram following the model transformation rules. The SRN model generated in this view is called dependability SRN.

The model synchronization is used as glue between performance SRN and dependability SRN. The synchronization task guides the performance SRN model to synchronize with the dependability SRN model by identifying the transitions in the dependability SRN. The synchronization between performance and dependability SRN is achieved by defining the guard functions. Once the performance SRN model is synchronized with dependability SRN model, a merged SRN model will be obtained and various performability measures can be evaluated from the merged model using the software package SHARPE [16].

III. UML BASED SYSTEM DESCRIPTION

A. Construction of collaborative building blocks

The performability modeling framework utilizes collaboration as main entity. Collaboration is an illustration of the relationship and interaction among software objects in the UML. Objects are shown as rectangles with naming label inside. The relationships between the objects are shown in a oval connecting the rectangles [5]. The specifications for collaborations are given as coherent, self-contained reusable building blocks. The structure of the building block is described by UML 2.2 collaboration. The building block declares the participants (as collaboration roles) and connection between them. 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. For each collaboration, the activity declares a corresponding call behavior action referring to the activities of the employed building blocks. For example, the general structure of the building block *t* is given in Fig. 2 where it only declares the participants *A* and *B* as collaboration roles and the connection between them is defined as collaboration t_x *(*$x=1 \ldots n_{AB}$ *(*number of collaborations between collaboration roles *A & B))*. The internal behavior of the same building block is shown in Fig. 3(b). The activity $transfer_{ij}$ (where *ij = AB*) describes the behavior of the corresponding collaboration. It has one activity partition for each collaboration role: *A* and *B*. Activities base their semantics on token flow [2]. The activity starts by forwarding a token when there is a response (indicated by the streaming pin *res*) to transfer

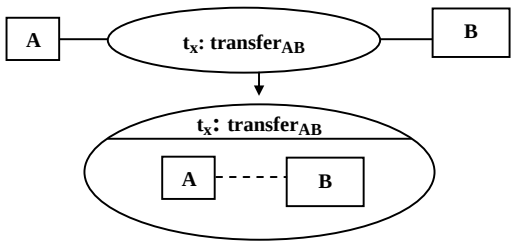


Figure 2. Structure of the Building block

from participant *A* to *B*. The token is then transferred by the participant *A* to participant *B* (represented by the call operation action *forward*) after completion of the processing by the collaboration role *A*. After getting the response of the participant *A,* the participant *B* starts the processing of the request (indicated by the streaming pin *req*).

In order to generate the performability model, the structural information about how the collaborations are composed is not sufficient. It is necessary to specify the detailed behavior of how the different events of collaborations are composed so that the desired overall system behavior can be obtained. For the composition, UML collaborations and activities are used complementary to each other. UML collaborations focus on the role binding and structural aspect, while UML activities complement this by covering also the behavioral aspect for composition. Therefore, the activity contains a separate call behavior action for all collaborations of the system. Collaboration is represented by connecting their input and output pins. Arbitrary logic between pins may be used to synchronize the building block events and transfer data between them. By connecting the individual input and output pins of the call behavior actions, the events occurring in different collaborations can be coupled with each other. Semantics of the different kinds of pins are given in more details in [14]. For example, the detail behavior and composition of the collaboration is given in following Fig. 3(a). The initial node (●) indicates the starting of the activity. The activity is started from the participant *A*. After being activated, each participant starts its processing of request which is mentioned by call operation action Pr_i *(*$Processing_i$*, where i = A, B & C)*. Completion of the processing by the participants are mentioned by the call operation action Prd_i *(*$Processing\_done_i$*, where i = A, B & C).* After completion of the processing, the response is delivered to the corresponding participant. When the processing of the task by the participant *A* completes, the response (indicated by streaming pin *res)* is transferred to the participant *B* mentioned by collaboration *t:* $transfer_{ij}$ (*where ij = AB)* and participant *B* starts the processing of the request (indicated by streaming pin *req).* After completion of the processing, participant *B* transfers the response to the participant *C* mentioned by collaboration *t:* $transfer_{ij}$ (*where ij = BC)*. Participant *C* starts the processing after receiving the response from *B* and activity is terminated after completion of the processing which is illustrated by the terminating node (◉).

B. Modeling failure & repair behavior of software & hardware component using UML STM

State transitions of a system element are described using UML STM diagram. In an STM, a state is depicted as a rectangle and a transition from one state to another is represented by an arrow [5]. In this work, STM is used to describe the failure and recovery behavior of software and hardware components.

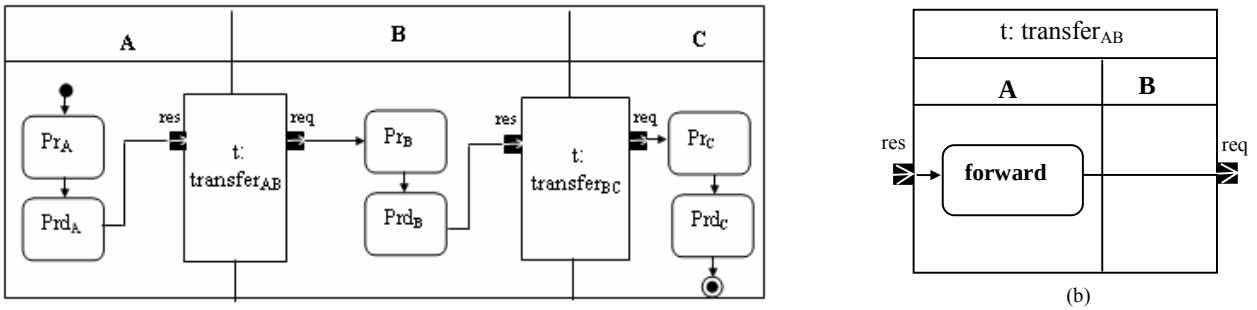


Figure 3. (a) Detail behavior of the event of the collaboration using activity (b) internal behavior of the collaboration

The STM of software process is shown in Fig. 4(a). The initial node (●) indicates the starting of the operation of software process. Then the process enters **Running** state. **Running** is the only available state in the STM. If the software process fails during the operation, the process enters **Failed** state. When the failure is detected by the external monitoring service the software process enters **Recovery** state and the repair operation will be started. When the failure of the process is recovered the software process returns to **Running** state.

The STM of hardware component is shown in Fig. 4(b). The initial node (●) indicates the starting of the operation of hardware component. Then the component enters **Running** state. **Running** is the only available state here. If the active component fails during the operation and the hot standby component is available, the standby component will take charge and the component operation will be continued. When any failure (whether active component or standby component) incurs, the recovery operation will be performed.

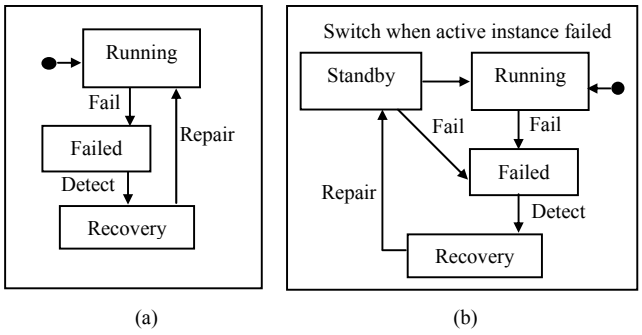


Figure 4. (a) STM of Software process (b) STM of Hardware component

IV. FORMALIZING UML DIAGRAM

So far we introduced the UML diagrams in a descriptive and informal way. In order to understand the precise formalism of the UML models and for the correct way of model transformation, we need to present the UML models with the help of formal semantics. The formal semantics of UML models thus help us implementing the models very efficiently for providing the tool based support of our framework. Before introducing the formalization of the UML models, at first, we illustrate the temporal logic, more specifically compositional Temporal Logic of Actions (cTLA) that will be applied to formalize the UML models. We illustrate in this paper the formal representation of the state machine model. Formalization of other UML models such as collaboration, activity, and deployment diagram and the alignment between UML models and cTLA (which is beyond the scope of this paper) have already been mentioned in [22].

A. *Compositional Temporal Logic of Action (cTLA)*

Lamport's Temporal Logic of Actions (TLA, [21]) is a linear-time temporal logic modeling the system behavior where the system behavior is realized by a set of considerably large number of state sequences [s_0, s_1, s_2 . .] [23]. Thus, the TLA formalisms are applied nicely to define the state machines formally produced by our framework which, in the end, also models considerably long sequences of states s_i starting with an initial state s_0. Compositional TLA (cTLA, [22]) was originated from TLA to offer more easily comprehensible formalisms and proposes a more supple composition of specifications. The concept of process is basically introduced by a cTLA. A cTLA process describes system behavior as the notion of state transition systems [23].

B. *Formalizing state machine diagram using cTLA*

We sketch the cTLA model of STM in Fig. 5 by the specification of software process dependability behavior illustrated in Fig. 4(a) [23]. The header *Software* declares the name of the process type. *Events* is an expression defined as constant record type. The state space is modeled by a set of variables like *state* or *Queue*. Predicate *INIT* specifies the subset of initial states. The state transition systems are mentioned by actions (*e.g., enqueue, dequeu*) which are realized as pairs of current and next states describing a set of transitions each. The current sate is defined as a variable in simple form (*e.g., state*), while the next state is mentioned by the prime form (*e.g., state´*). Variables which won't be changed by an action are listed by the statement UNCHANGED [23]. State transition system is defined by the body of a cTLA process type. One cTLA process represents one state machine that mentions a set of TLA state sequences. The first state s_0 of each modeled state

sequence has to fulfil the initial condition *INIT*. The state changes [s_i, s_{i+1}] either correspond with a process action or with a so-called stuttering step in which the current and the next states are equal (i.e., $s_i = s_{i+1}$) [23]. Incoming events are

```
PROCESS Software ()

CONSTANTS
        Events ≜ {Fail, Detect, Recovery};

VARIABLES
        state: {initState, running, failed, recovered};
        Queue: QUEUE of Events;

INIT ≜ state = initState ⋀ Queue = EMPTY;

ACTIONS
    enqueue (addEvent: Events) ≜
    Queue´ = Queue ◦ addEvent ⋀ state ≠ initState ⋀
                UNCHANGED〈state〉
    dequeue (fetchEvent: Events) ≜
    Queue ≠ EMPTY ⋀ fetchEvent = FIRST(Queue) ⋀
                UNCHANGED 〈state〉
        Initial ≜ state = initState ⋀ state´= idle ⋀
                UNCHANGED 〈Queue〉
        stateseq ≜ state = idle ⋀ state´ = running ⋀
                UNCHANGED 〈Queue〉
 …….. state change for other actions ……

WF: dequeue, initial;
```

Figure 5. cTLA process of Software component

inserted into the data structure *addEvent*, which is a sequence of events. The operator ◦ denotes the concatenation of queue elements. Events are added to the queue by the action *enqueue*, which takes incoming events as action parameters [23]. Retrieving events are modeled by the data structure *fetchEvent* where the first element is obtained by the operations FIRST(). Events are retrieved from the queue by the action *dequeue* which takes retrieving events as action parameters. An initial transition initiates from an initial pseudo state (*initState*) and its execution is associated with the starting of the state machine. Exactly one initial transition is linked with each state machine [23]. A cTLA variable *state* describes the control state by expressing them through the control state identifiers. *Stateseq* captures the current and next state and starts from initial state of the STM diagram. In order to conduct an action in a lively manner, we can associate actions with weak and strong fairness properties. In particular, weak fairness forces the execution of an activity as if it were enabled continuously. Strong fairness forces the execution even if the action is sometimes disabled [23]. The last statement WF: dequeue, initial,... lists the actions that have to be carried out in a way which ensures week fairness property [23].

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

We model the system as collection of *N* interconnected physical nodes. Our objective is to find a deployment mapping for this execution environment for a set of service components available for deployment that comprises the service. Deployment mapping M can be defined as [M=($C \rightarrow N$)] between a number of service components instances *C*, onto physical nodes *N*. 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: (1) Service components have execution costs, (2) Collaborations have communication costs and costs for running of background process known as overhead cost, (3) Some of the service components can be restricted in the deployment mapping to specific physical nodes which are called bound components.

Furthermore, we consider identical physical nodes that are interconnected in a full-mesh and are capable of hosting service components with unlimited processing demand. We observe the processing cost that physical nodes impose while hosting the service components and also the target balancing of cost among the physical nodes available in the network. Communication costs are considered if collaboration between two service components happens remotely, i.e. it happens between two physical nodes [18]. 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 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 physical nodes. Using the above specified input, the deployment logic provides an optimal deployment architecture taking into account the QoS requirements for the service 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 service components onto the physical nodes that satisfies the requirements in a reasonable time. The deployment mapping providing optimal deployment architecture is mentioned by the cost function F(M), that is a function that expresses the utility of deployment mapping of service components on the physical resources with their constraints and capabilities by satisfying non-functional requirements of the system. The cost function is designed to reflect the goal of balancing the execution cost and minimizing the communication cost. This is in turn utilized to achieve reduced task turnaround time by maximizing the utilization of system 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 [14]. The evaluation of cost function F(M) is mainly influenced by our way of service definition. A service is defined in our approach as a collaboration of total *E* service 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_{c_i} , the communication cost between the service components is labeled as f_{k_j} and the cost for executing the background process for conducting the communication between the service components is labeled as f_{B_j} . Accordingly, 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 f_{c_i} , *i* = 1….*E*, communication cost f_{k_j} , *j* = 1….*K*, and cost for executing the background process f_{B_j} , *j* = 1….*K*. f_{c_i} , f_{k_j} , and f_{B_j} are derived from the service specification, thus the offered execution cost can be calculated as $\sum_{i=1}^{|E|} f_{c_i}$. This way, the logic can be aware of the target average cost *T* per physical node (*X*= total number of physical nodes) [18]:

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

In order to cater for the communication cost f_{k_j} , of the collaboration k_j in the service, the function $q_0(M,c)$ is defined first [20]:

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

This means that $q_0(M,c)$ returns the physical node *n* from a vector of physical nodes *N* available in the network that host component in the list mapping M. Let collaboration $k_j = (c_1, c_2)$. The assumption in this paper is that, the communication cost of k_j is 0 (in general, it can be non-zero) 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_{k_j} if service 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 is 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 [20]:

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

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 $\hat{l}_n = \sum_{c_i \in m_n} f_{c_i}$. Furthermore, the overall cost function F(M) becomes [20] (where I_j = 1, if k_j external or 0 if k_j internal to a node):

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

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

VI. ANNOTATION

In order to annotate the UML diagrams, the stereotype *saStep, computingResource, scheduler, QoSDimension*, and the tagged value *execTime, deadline, mean-time-to-repair, mean-time-between-failures,* and *schedPolicy* are used according to the *UML profile for MARTE* and *UML Profile for Modeling Quality of Service & Fault Tolerance Characteristics* [8] [13]. The stereotypes are the following:

- *saStep* defines a step that begins and ends when decisions about the allocation of system resources are made.
- *computingResource* represents either virtual or physical processing devices capable of storing and executing program code. Hence, its fundamental service is to compute.
- *scheduler* is a stereotype that brings access to a resource following a certain scheduling policy mentioned by tagged value *schedPolicy.*
- *QoSDimension* provides support for the quantification of QoS characteristics and attributes *mean-time-to-repair* and *mean-time-between-failures* [13].

The tagged values are the following:

- *execTime*: The duration of the execution time is mentioned by the tagged value *execTime* which is the average time in our case.
- *deadline* defines the maximum time bound on the completion of the particular execution segment that must be met.
- *mean-time-between-failures* defines the mean time of occurring a software and hardware instance failure
- *mean-time-to-repair* defines the mean time that is required to repair a software or hardware instance failure

We also introduce a new stereotype <<*transition*>> and three tag values *mean-time-to-stop, mean-time-to-start*, and *mean-time-to-failure-detect*.

- <<*transition*>> induces a state transition of a scenario.
- *mean-time-to-stop* defines the mean time that is required by a hardware instance to stop working
- *mean-time-to-start* states the mean time that is required by a hardware instance to start working
- *mean- time-to-failure-detect* defines the mean time that is required to detect failures in the system.

Fig. 6 illustrates an example annotated UML model using the activity diagram where the flow between P_A and d_A is annotated using stereotype *saStep* and tagged value

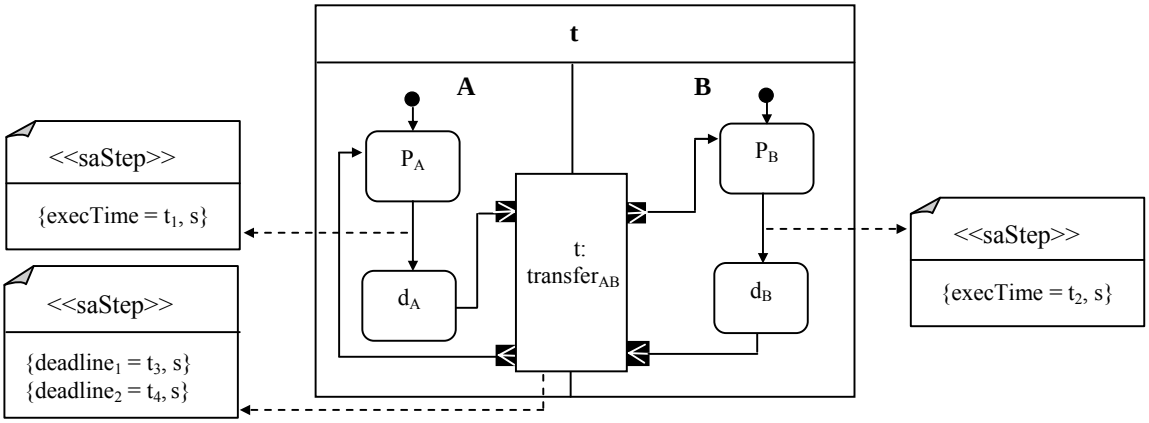


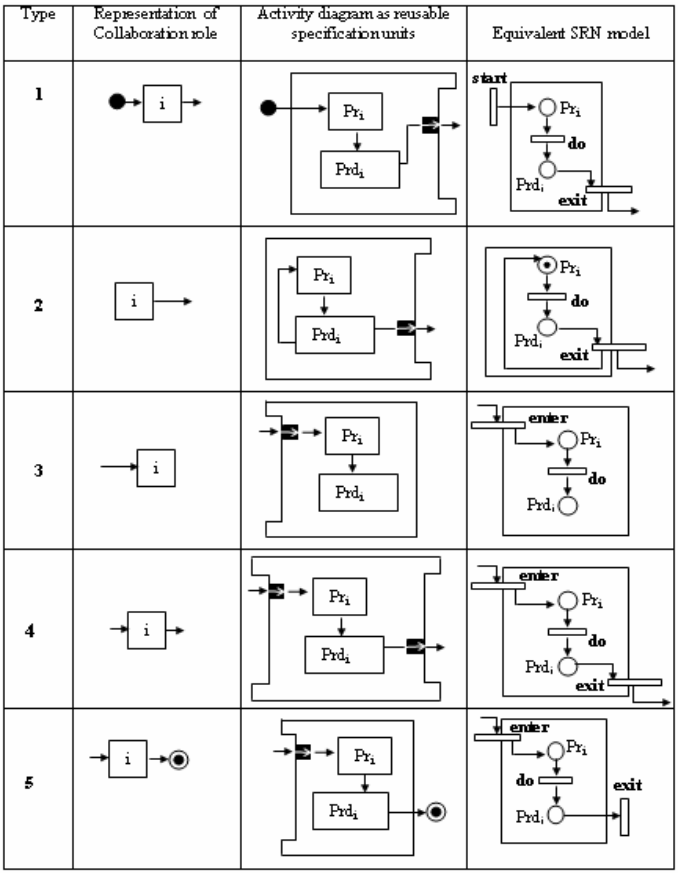
Figure 6. Annotated UML model

execTime which defines that after being deployed in an execution environment the collaboration role *A* needs t_1 seconds and collaboration role *B* needs t_2 seconds to complete their processing by the physical node. After completing the processing, communication between *A* and *B* is achieved in t_3 sec while the overhead time to conduct this communication is t_4 sec which is annotated using stereotype *saStep* and two instances of *deadline – deadline*$_1$ defines the communication time and *deadline*$_2$ is for overhead time.

VII. MODEL TRANSLATION

This section highlights the rules for the model translation from various UML models into SRN models. Since all the models will be translated into the SRN model, we will give a brief introduction about SRN model. SRN is based on the Generalized Stochastic Petri Net (GSPN) [4] and extends them further by introducing prominent extensions such as

TABLE I. SPECIFICATION OF REUSABLE UNITES AND EQUIVALENT SRN MODEL



guard function, reward function, and marking dependent firing rate [6]. A guard function is assigned to a transition. It specifies the condition to enable or disable a transition and can use the entire state of the net rather than just the number of tokens in places [6]. Reward function defines the reward rate for each tangible marking of Petri Net based on which various quantitative measures can be done in the Net level. Marking dependent firing rate allows using the number of tokens in a chosen place multiplied by the basic rate of the transition. SRN model has the following elements: Finite set of the place (drawn as circles), Finite set of the transition defined as either a timed transition (drawn as thick transparent bar) or a immediate transition (drawn as thick black bar), set of the arc connecting the place and transition, multiplicity associated with the arc, and marking that denotes the number of token in each place.

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

The rules are the following:

Rule 1

The SRN model of a collaboration (Fig. 7), where collaboration connects only two collaboration roles, is formed by combining the basic building blocks type 2 and type 3 from Table I. Transition ***t*** in the SRN model is only realized by the overhead cost if service components A and B deploy on the same physical node as in this case, communication cost = 0, otherwise ***t*** is realized by both the communication & overhead cost.

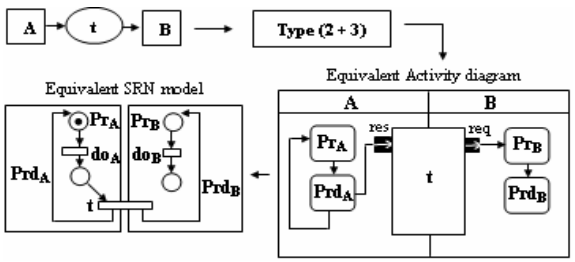


Figure 7. Graphical representation of rule 1

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

Rule 2

For a composite structure, when a collaboration role A connects with *n* collaboration roles by *n* collaborations like a star graph (where n > 1) where each collaboration connects only two collaboration roles, the SRN model is formed by combining the basic building block of Table I which is shown in Fig. 8. In the first diagram of Fig. 8, if component A contains its own token, equivalent SRN model of the collaboration role A will be formed using basic building block type 1 from Table I. The same applies to the component B and C in the second diagram in Fig. 8.

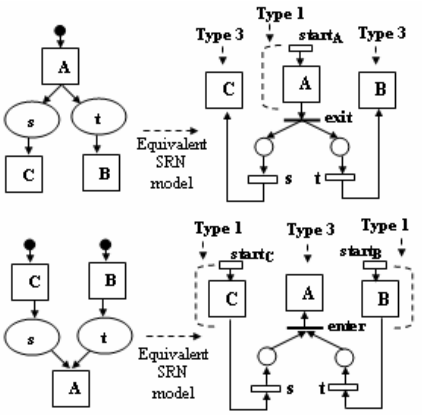


Figure 8. Graphical representation of rule 2

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

Rule 3

Rule 3 demonstrates the equivalent SRN model of the STM of hardware and software components which are shown in the Fig. 9.

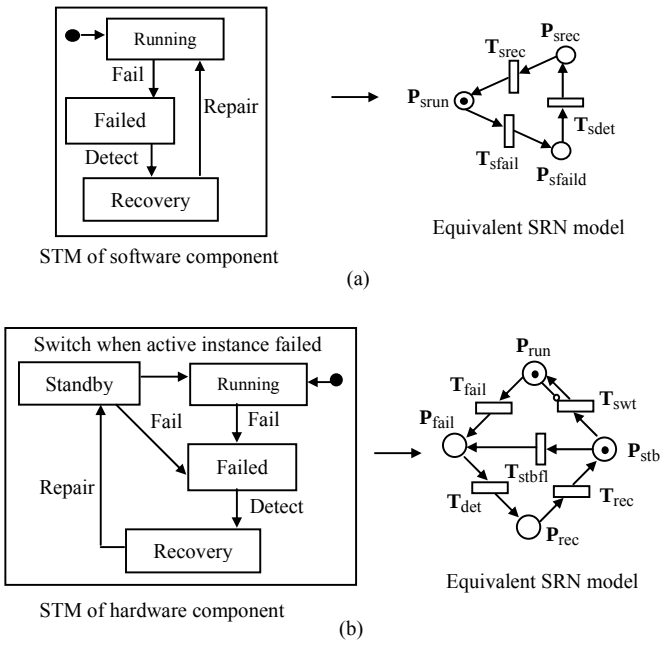


Figure 9 (a) SRN of Software process (b) SRN of hardware component

The SRN model for hardware component is shown in Fig. 9(b). A token in the place P_{run} represents the active hardware component and a token in P_{stb} represents a hot standby hardware component. When the transition T_{fail} fires, the token in P_{run} is removed and the transition T_{swt} is enabled. By the T_{swt}, which represents the failover, hot standby hardware component becomes an active component.

VIII. MODEL SYNCHRONIZATION

The model synchronization is achieved hierarchically which is illustrated in Fig. 10. Performance SRN is dependent on the dependability SRN. Transitions in dependability SRN may change the behavior of the performance SRN. Moreover, transitions in the SRN model for the software process also depend on the transitions in the SRN model of the hardware component. These dependencies in the SRN models are handled through model synchronization by incorporating guard functions [6].

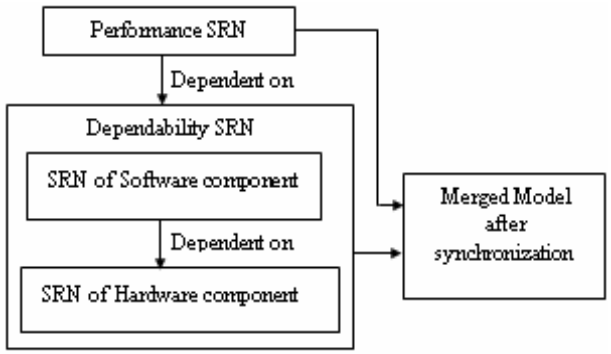


Figure 10. Model synchronization hierarchy

The model synchronization is focused in detail below:

A. Synchronization between the dependability SRN models in the dependability modeling layer

SRN model for the software process (Fig. 9(a)) is expanded by incorporating one additional place P_{hf}, three immediate transitions t_{hf}, t_{hsfl}, t_{hfr}, and one timed transition T_{recv} to synchronize the transitions in the SRN model for the software process with the SRN model for the hardware component. The expanded SRN model (Fig. 11(a)) is

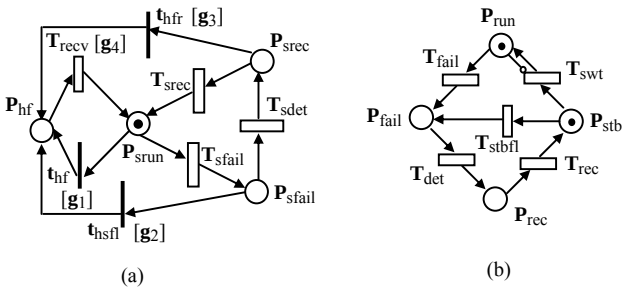


Figure 11. (a) Synchronized transition in the SRN model of the software process with the (b) SRN model of the hardware component

associated with four additional arcs such as $(P_{sfail} \times t_{hsfl}) \cup (t_{hsfl} \times P_{hf})$, $(P_{srec} \times t_{hfr}) \cup (t_{hfr} \times P_{hf})$, $(P_{srun} \times t_{hf}) \cup (t_{hf} \times P_{hf})$ and $(P_{hf} \times T_{recv}) \cup (T_{recv} \times P_{srun})$. The immediate transitions t_{hf}, t_{hsfl}, t_{hfr} will be enabled only when the hardware node (in Fig. 11 (b)) fails as failure of hardware node will stop operation of the software process. The timed transition T_{recv} will be enabled only when the hardware node will again start working after being recovered from failure. Four guard functions g_1, g_2, g_3, g_4 allow the four additional transitions t_{hf}, t_{hsfl}, t_{hfr} and T_{recv} of software process to work consistently with the change of states of the hardware node. The guard functions definitions are given in the Table II.

TABLE II. GUARD FUNCTIONS DEFINITION

Function	Definition
g_1, g_2, g_3	if (# P_{run} = = 0) 1 else 0
g_4	if (# P_{run} = = 1) 1 else 0

B. Synchronization between the dependability SRN & performance SRN

In order to synchronize the collaboration role activity, performance SRN model is expanded by incorporating one additional place P_{fl} and one immediate transition f_A shown in Fig. 12. After being deployed when collaboration role "A" starts execution, a checking will be performed to examine whether both software and hardware components are running or not. If both the components work the timed transition do_A will fire which represents the continuation of the execution of the collaboration role A. But if software resp. hardware components fail the immediate transition f_A will be fired which represents the quitting of the operation of collaboration role A. Guard function gr_A allows the immediate transition f_A to work consistently with the change of states of the software and hardware components.

Performance SRN model of parallel execution of collaboration roles are expanded by incorporating one additional place P_{fl} and immediate transitions f_{BC}, w_{BC} shown in Fig. 12. In our discussion, during the synchronization of the parallel processes it needs to ensure that failure of one process eventually stops providing service to the users. This could be achieved by immediate transition f_{BC}. If software resp. hardware components (Fig. 11) fail immediate transition f_{BC} will be fired which symbolizes the quitting of the operation of both parallel processes B and C rather than stopping either process B or C, thus postponing the execution of the service. Stopping only either the process B or C will result in inconsistent execution of the whole SRN and produce erroneous result. If both software and hardware components work fine the timed transition w_{BC} will fire to continue the execution of parallel processes B and C. Guard functions gr_{BC}, grw_{BC} allow the immediate

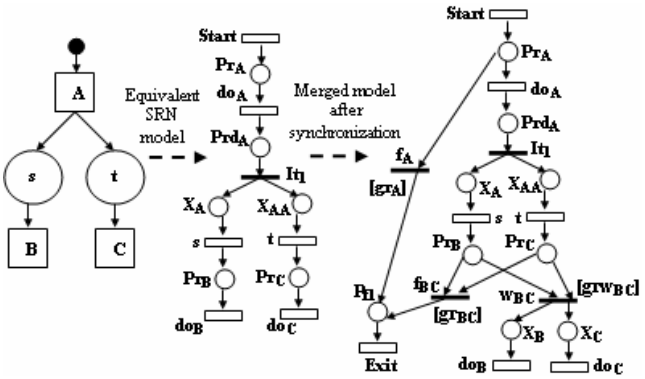


Figure 12. Synchronize the performance SRN model with dependability SRN

transition $f_{BC,}$ w_{BC} to work consistently with the change of the states of the software and hardware components. The guard function definitions are shown in the Table III.

Algorithms for model transformation rules and model synchronization process have been mentioned in Appendix A.

TABLE III. GUARD FUNCTIONS DEFINITION

Function	Definition
gr_A, gr_{BC}	if (# P_{srun} = = 0) 1 else 0
grw_{BC}	if (# P_{srun} = = 1) 1 else 0

IX. HIERARCHICAL MODEL FOR MTTF CALCULATION

System is composed of different types of hardware devices such as CPU, memory, storage device, cooler. Hence, to model the failure behavior of a hardware node absolutely, we need to consider failure behavior of all the hardware devices. But it is very demanding and not efficient with respect to execution time to consider behavior of all the hardware components during the SRN model generation. SRN model becomes very cumbersome and inefficient to execute. In order to solve the problem, we evaluate the mean time to failure (MTTF) of system using the hierarchical model in which a fault tree is used to represent the MTTF of the system by considering MTTF of every hardware component in the system. Later on, we consider this MTTF of the system in our dependability SRN model for hardware components (Fig. 9(b)) rather than considering failure behavior of all the hardware components individually. The below Fig. 13 introduces one example scenario of capturing failure behavior of the hardware components using fault tree where system is composed of different hardware devices such as one CPU, two memory interfaces, one storage device and one cooler. The system will work when CPU, one of the memory interfaces, storage device and cooler will run. Failure of both memory interfaces or failure of either CPU or storage device or cooler will result in the system unavailability.

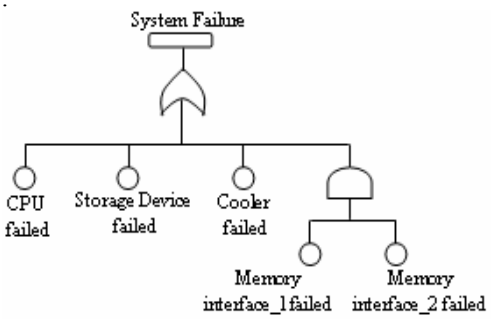


Figure 13. Fault tree model of System Failure

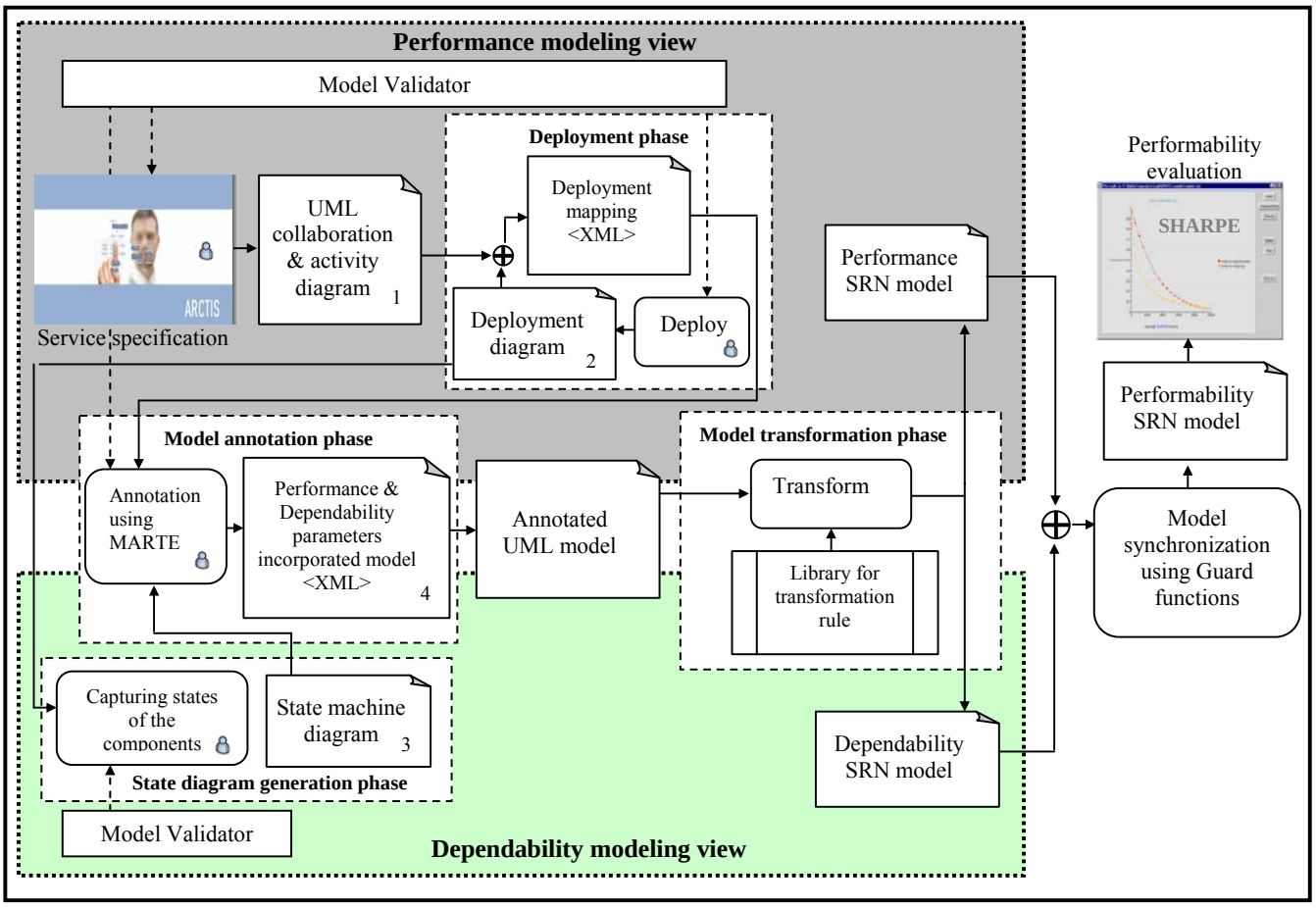


Figure 14. Tool support of our performability modeling framework

X. TOOL BASED SUPPORT OF THE PERFORMABILITY MODELING FRAMEWORK

The theoretical foundation of the approach is described in details in the above sections. We highlight the tool support of our performability modeling framework in Fig. 14. The partial input model of our framework is generated using Arctis tool which is integrated as plug-in into the eclipse IDE. In the evaluation side, SHARPE tool is used. We generate the annotated UML model from the UML collaboration diagram, deployment diagram, STM diagram, and the performance and dependability related parameters. From Fig. 14, it is evident that we need to define 4 inputs accordingly: in the performance modeling view, the first input UML collaboration diagram and the detail behavior of collaborative building block will be generated using the GUI (Graphical User Interface) editor of Arctis tool which will be saved as XML file and the other two inputs of performance modeling view will be generated as XML file such as deployment diagram and performance attributes incorporated UML model after deployment mapping. The inputs of the dependability modeling view such as STM diagram and dependability attributes incorporated UML model will be generated as XML file as well. We also define one output file in text format which is generated as a result of the model annotation phase denoting the annotated UML model. The annotated UML model file is then further used as an input for the model transformation phase to achieve automation in model transformation. In the model transformation phase, we automate the transformation process from annotated UML model to the SRN performability model following the model transformation rules and afterwards, merging of SRN performance and dependability model using guard functions. The input files are specified in XML formats. This is because of the fact that XML gives benefits to guarantee the robustness, flexibility to extend the existing file, and data validation. The output files are all in text format as the SHARPE tool, that evaluates the performance of the system, accepts the input as text format.

XI. CASE STUDY

As a representative example, we consider a scenario dealing with heuristically clustering of modules and assignment of clusters to nodes [17]. This scenario is sufficiently complex to show the applicability of our performability framework. The problem is defined in our approach as collaboration of $E = 10$ service components or collaboration roles (labeled $C_1 \ldots C_{10}$) to be deployed and $K = 14$ collaborations between them illustrated in Fig. 15. We consider three types of requirements in this specification. Besides the execution cost, communication cost, and cost for running background process, we have a restriction on components C_2, C_7, C_9 regarding their location. They must be bound to nodes $\boldsymbol{n_2}$, $\boldsymbol{n_1}$, $\boldsymbol{n_3}$ respectively. In this scenario, new service is generated by integrating and combining the existing service components that will be delivered conveniently by the system. For example, one new service is

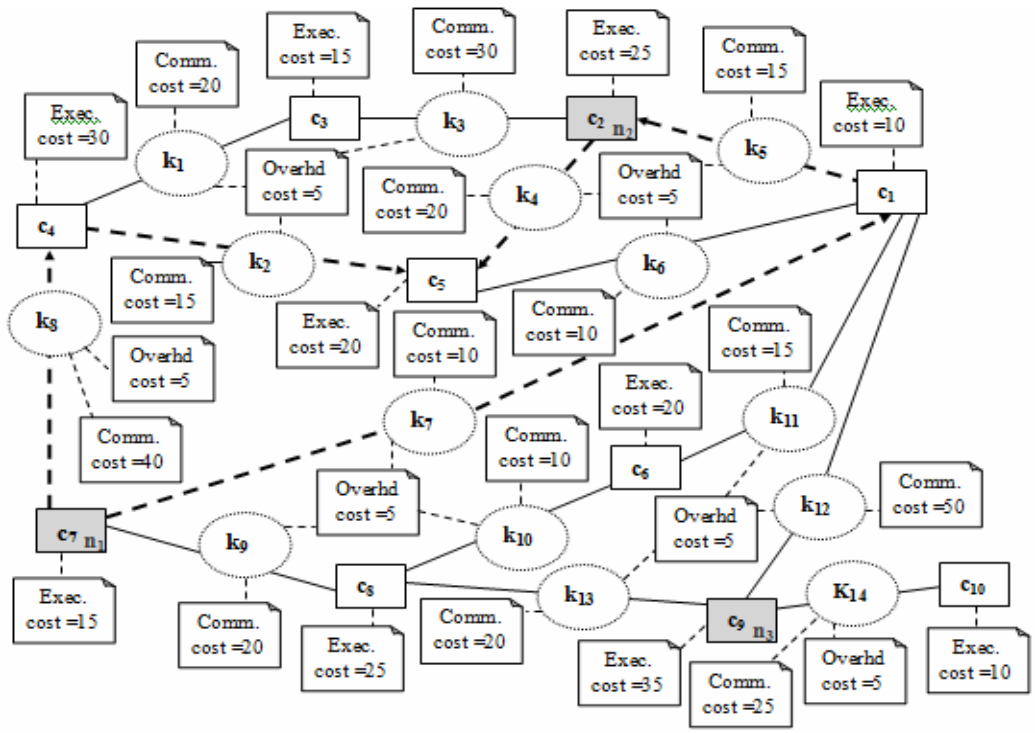


Figure 15. Collaboration & Components in the example Scenario

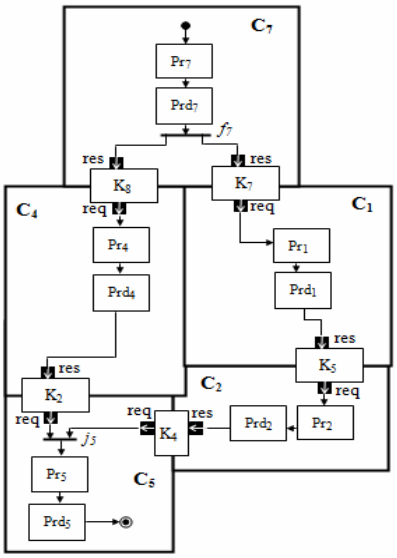


Figure 16. Composition of collaboration

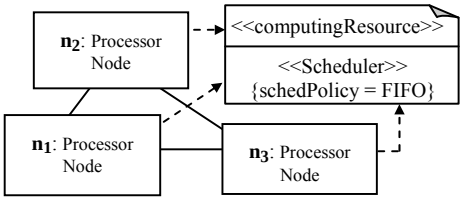


Figure 17. The target network of hosts

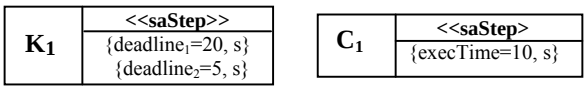


Figure 18. Annotated UML model

composed by combining the service components C_1, C_2, C_4, C_5, C_7 shown in Fig. 15 as thick dashed line. The internal behavior of the collaboration K_i is realized by the call behavior actions through the same UML activity diagram already demonstrated in Fig. 3(b). The composition of the collaboration role C_i of the delivered service by the system is demonstrated in Fig. 16. The initial node (●) indicates the starting of the activity. After being activated, each participant starts its processing of request which is mentioned by call behavior action Pr_i (Processing of the ith service component). Completions of the processing by the participants are mentioned by the call behavior action Prd_i (Processing done of the ith service component). The activity is started from the component C_7 where the semantics of the activity is realized by the token flow. After completion of the processing of the component C_7, the response is divided into two flows which are shown by the fork node f_7. The flows are activated towards component C_1 and C_4. After getting the response from the component C_1, processing of the components C_2 will be started. The response and request are mentioned by the streaming pin *res* and *req*. The processing of the component C_5 will be started after getting the responses from both component C_4 and C_2 which is realized by the join node j_5. After completion of the processing of component C_5, the activity is terminated which is mentioned by the end node (◉).

In this example, the target environment consists of $N = 3$ identical, interconnected nodes with no failure of network link, with a single provided property, namely processing power, and with infinite communication capacities shown in Fig. 17. The optimal deployment mapping can be observed in Table IV. The lowest possible deployment cost, according to equation (4) is: 17 + 100 + 70 = 187.

In order to annotate the UML diagrams in Fig. 16 and 17, we use the stereotypes *<<saStep>>* *<<computingResource>>*, *<<scheduler>>* and the tagged values *execTime*, *deadline* and *schedPolicy* which are already explained in section 5. Collaboration K_i (Fig. 18) is associated with two instances of *deadline* as collaborations in example scenario are associated with two kinds of cost:

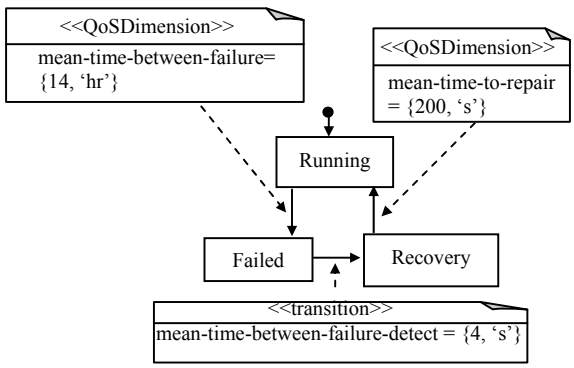


Figure 19. Annotated STM diagram of software component

TABLE IV. OPTIMAL DEPLOYMENT MAPPING

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

communication cost and cost for running background process (BP). In order to annotate the STM UML diagram of software process (shown in Fig. 19), we use the stereotype *<<QoSDimension>>*, *<<transition>>* and attributes *mean-time-between-failures, mean-time-between-failure-detect* and *mean-time-to-repair* which are already mentioned in section VI. Annotation of the STM of hardware component can be demonstrated in the same way as STM of software process.

By considering the specification of reusable collaborative building blocks, deployment mapping, and the model transformation rule, the corresponding SRN model of our example scenario is illustrated in Fig. 20. In our discussion we consider M/M/1/n queuing system so that at most *n* jobs can be in the system at a time [3]. For generating the SRN model, firstly, we will consider the starting node (●). According to rule 1, it is represented by timed transition (denoted as *start*) and the arc connects to place Pr_7 (states of component C_7). When a token is deposited in place Pr_7, immediately a checking is done about the availability of both software and hardware components by inspecting the corresponding SRN models shown in Fig. 11. The availability of software and hardware components allows the firing of timed transition t_7 mentioning the continuation of the further execution. Otherwise, immediate transition f_7 will be fired mentioning the ending of the further execution because of software resp. hardware component failure. The enabling of immediate transition f_7 is realized by the guard function $\boldsymbol{gr_7}$. After the completion of the state transition from Pr_7 to Prd_7 (states of component C_7), immediately, the flow is divided into two branches (denoted by the immediate transition It_1) according to model transformation rule 2 (Fig. 8). The token is passed to place Pr_1 (states of component C_1) and Pr_4 (states of component C_4) after the firing of transitions K_7 and K_8. According to rule 1, collaboration K_8 is realized only by overhead cost as C_4 and C_7 deploy on the same processor node $\boldsymbol{n_1}$ (Table IV). The collaboration K_7 is realized both by the communication cost and overhead cost as C_1 and C_7 deploy on the two different nodes $\boldsymbol{n_3}$ and $\boldsymbol{n_1}$ (Table IV). When a token is deposited into place Pr_1 and Pr_4, immediately, a checking is done about the availability of both software and hardware components by inspecting the corresponding dependability SRN models illustrated in Fig. 11. The availability of software and hardware components allows the firing of immediate transition w_{14} which eventually enables the firing of timed transition t_1 mentioning the continuation of the further execution. The enabling of immediate transition w_{14} is realized by the guard function $\boldsymbol{grw_{14}}$. Otherwise, immediate transition f_{14} will be fired mentioning the ending of the further execution because of software resp. hardware component failure. The enabling of immediate transition f_{14} is realized by the guard function $\boldsymbol{gr_{14}}$. After the completion of the state transition from Pr_1 to Prd_1 (states of component C_1) the token is passed to Pr_2 (states of component C_2) according to rule 1, where timed transition K_5 is realized both by the communication and overhead cost. When a token is deposited into place Pr_2, immediately a checking is done about the availability of both software and hardware components by inspecting the corresponding dependability SRN models shown in Fig. 11. The availability of software and hardware components allows the firing of the immediate transition w_{24} which eventually enables the firing of timed transition t_2 and t_4 mentioning the continuation of the further execution. The enabling of immediate transition w_{24} is realized by the guard function $\boldsymbol{grw_{24}}$. Otherwise, immediate transition f_{24} guided by guard function $\boldsymbol{gr_{24}}$ will be fired mentioning the ending of the further execution because of software resp. hardware component failure. Afterwards, the merging of the result is realized by the immediate transition It_2 following the firing of transitions K_2 and K_4. Collaboration K_2 is realized both by the overhead cost and communication cost as C_4 and C_5 deploy on the different processor nodes $\boldsymbol{n_1}$ and $\boldsymbol{n_2}$ (Table IV). K_4 is replaced by the timed transition which is realized by the overhead cost as C_2 and C_5 deploy on the same node $\boldsymbol{n_2}$ (Table IV). When a token is deposited in place Pr_5 (state of component C_5), immediately, a checking is done about the availability of both software and hardware components by inspecting the corresponding SRN models illustrated in Fig. 11. The availability of software and hardware components allows the firing of timed transition t_5 mentioning the

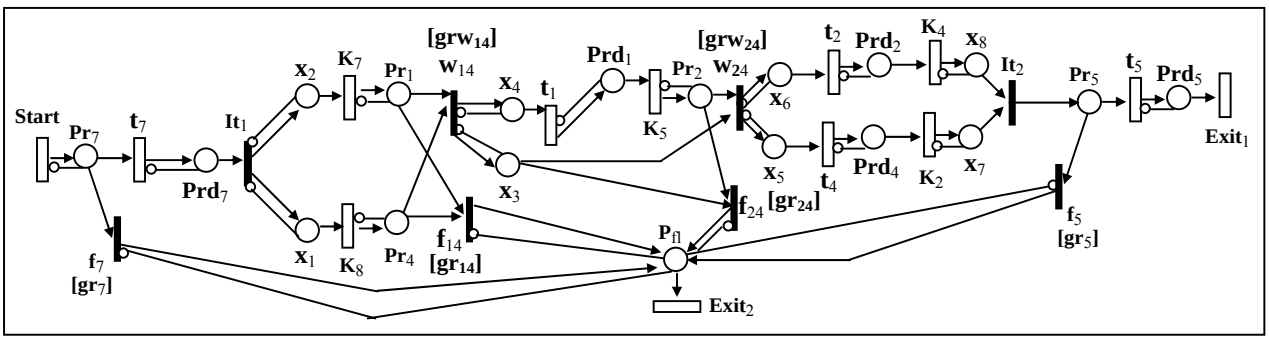


Figure 20. Equivalent SRN model of the example service

continuation of the further execution. Otherwise, immediate transition f_5 will be fired mentioning the ending of the further execution because of software resp. hardware component failure and the ending of the execution of the SRN model is realized by the timed transition $Exit_2$. The enabling of immediate transition f_5 is realized by the guard function $\boldsymbol{gr_5}$. After the completion of the state transition from Pr_5 to Prd_5 (states of component C_5) the ending of the execution of the SRN model is realized by the timed transition $Exit_1$. The definitions of guard functions $\boldsymbol{gr_7}$, $\boldsymbol{grw_{14}}$, $\boldsymbol{gr_{14}}$, $\boldsymbol{grw_{24}}$, $\boldsymbol{gr_{24}}$, and $\boldsymbol{gr_5}$ are mentioned in Table V, which is dependent on the execution of the SRN model of the corresponding STM of software and hardware instances illustrated in Fig. 11.

TABLE V. GUARD FUNCTIONS DEFINITION

Function	Definition
gr_7, gr_{14}, gr_{24}, gr_5	if (# P_{srun} = = 0) 1 else 0
grw_{14}, grw_{24}	if (# P_{srun} = = 1) 1 else 0

We use SHARPE [16] to execute the obtained synchronized SRN model and calculate the system's throughput and job success probability against failure rate of system components. Graphs in Fig. 21 show the throughput and job success probability of the system against the changing of the failure rate (sec^{-1}) of hardware and software components in the system.

XII. CONCLUSION AND FUTURE WORK

We presented a novel approach for model based performability evaluation of a distributed software system. The approach spans from system's dynamics demonstration through UML diagram as reusable building blocks to efficient deployment of service components in a distributed manner focusing on the QoS requirements. The main advantage of using the reusable software components allows the cooperation among several software components to be reused within one self-contained, encapsulated building block. Moreover, reusability thus assists in creating the distributed software systems from existing software components rather than developing the system from scratch which in turn facilitates the improvement of productivity and quality in accordance with the reduction in time and cost. We put emphasis to establish some important concerns relating to the specification and solution of performability models emphasizing the analysis of the system's dynamics. We design the framework in a hierarchical and modular way which has the advantage of introducing any modification or adjustment at a specific layer in a particular submodel rather than in the combined model according to any change in the specification. Among the important issues that come up in our development are flexibility of capturing the system's dynamics using our new reusable specification of building blocks, ease of understanding the intricacy of combined model generation, and evaluation from that specification by proposing model transformation. However, our eventual goal is to develop support for runtime redeployment of components, this way keeping the service within an allowed region of parameters defined by the requirements. As a result, with our proposed framework we can show that our logic will be a prominent candidate for a robust and adaptive service execution platform. The special property of SRN model like guard function keeps the performability model simpler by applying logical conditions that can be expressed graphically using input and inhibitor arcs which are limited by the following semantics: a logical "AND" for input arcs (all the input conditions must be satisfied), a logical "OR" for inhibitor arcs (any inhibitor condition is sufficient to disable the transition) [18]. However, the size of the underlying reachability set to generate a SRN model is major limitation for large and complex systems. Further work includes tackling the state explosion problems of reachability marking for large distributed systems. In addition, developing GUI editor is another future direction to generate UML deployment and state diagram and to incorporate performability related parameters. The plug-ins can be integrated into the Arctis tool which will provide the automated and incremental model checking while conducting model transformation.

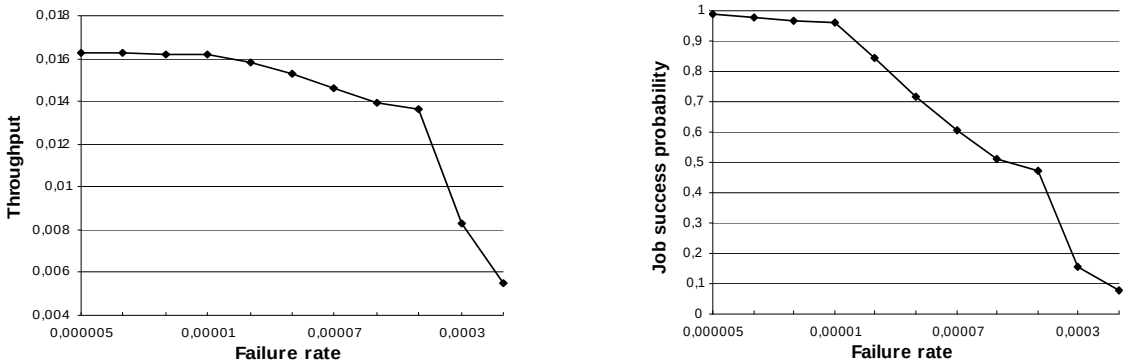


Figure 21. Numerical result of our example scenario

REFERENCES

[1] R H Khan, F Machida, P. Heegaard, and K S Trivedi, "From UML to SRN: A performability modeling framework considering service components deployment", Proceeding of the ICNS, pp. 118-127, IARIA, 2012

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

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

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

[5] OMG 2009, "OMG UML Superstructure", Version-2.2

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

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

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

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

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

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

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

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

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

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

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

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

[18] Muppala, Ciardo and K. Trivedi, Stochastic reward nets for reliability prediction", Communications in Reliability, Maintainability and Serviceability, SAE International, 1994

[19] P. Herrmann and H. Krumm, "A Framework for Modeling Transfer Protocols", Computer Networks, Vol - 34, No - 2, pp.317–337, 2000

[20] Vidar Slåtten, "Model Checking Collaborative Service Specifications in TLA with TLC", Project Thesis, Norwegian University of Science and Technology, Trondheim, Norway, August 2007

[21] Lamport, "Specifying Systems", Addison-Wesley, 2002

[22] R. H. Khan and Poul E. Heegaard, "Software Performance evaluation utilizing UML Specification and SRN model and their formal representation", Submitted to a journal for reviewing.

[23] F. Kræmer, P. Herrmann, and R. Bræk, "Aligning UML 2 state machines & temporal logic for the effecient execution of services", Proceedings of the DOA, Springer, 2006

APPENDIX A

Algorithm 1: rule_1 (ExecCost, CommCost, Ovrhdcost, Mappings, CollaborationRoles)

```
1   If CollaborationRoles A self token generator then
2       Places += "Pr_A 1"
3   else (A has a external token generator)
4       Places += "Pr_A 0"
5   Places += "Prd_A 0"
6   Places += "Pr_B 0"
7   Places += "Prd_B 0"
8   Timed_Transitions += "do_A ind" + 1/ execution cost for collaborationRole A
9   Timed_Transitions += "do_B ind" + 1/ execution cost for collaboration role B
10  Timed_Transitions += "exit ind" + 1/ rate for the end transition
11  If CollaborationRoles A and B are deployed on the same node then
12      Timed_Transitions += "t ind" + 1/ overhead cost
13  else
14      Timed_Transitions += "t ind" + 1/ (overhead cost + communication cost)
15  If CollaborationRole A has a external token generator then
16      Timed_Transitions += "Start ind" + 1/ rate of the token generator
17  Inhibitor_Arcs += "Pr_A Start 1"
18  Inhibitor_Arcs += "Prd_A do_A 1"
19  Inhibitor_Arcs += "Pr_B t 1"
20  Inhibitor_Arcs += "Prd_B do_B 1"
21. Input_Arcs += "Pr_A do_A 1"
22  Input_Arcs += "Prd_A t 1"
23  Input_Arcs += "Pr_B do_B 1"
24  Input_Arcs += "Prd_B exit 1"
25  Output_Arcs += "do_A Prd_A 1"
26  Output_Arcs += "do_B Prd_B 1"
27  Output_Arcs += "t Pr_B 1"
28  if CollaborationRole A self token generator then
29      Output_Arcs += "t PrA 1"
30  else
31      Output_Arcs += "Start PrA 1"
32  Print Places, Timed_Transitions, Input_Arcs, Output_Arcs, Inhibitor_Arcs
33  return
```

Algorithm 2: rule_2_a (ExecCost, CommCost, Ovrhdcost, Mappings, CollaborationRoles)

```
1   Places += "Pr_A 0"
2   Places += "Prd_A 0"
3   Places += "Pr_B 0"
4   Places += "Prd_B 0"
5   Places += "Pr_C 0"
6   Places += "Prd_C 0"
7   Places += "X_c 0"
8   Places += "X_b 0"
9   Immediate_Transitions += "it ind 1"
10  Timed_Transitions += "Start ind" + 1 / rate of the
                         external token generator
11  Timed_Transitions += "do_A ind "+ 1 / execution cost
                         of collaboration role A
12  Timed_Transitions += "do_B ind "+ 1 / execution cost
                         of collaboration role B
13  Timed_Transitions += "doC ind "+ 1 / execution cost
                         of collaboration role C
14  if CollaborationRoles A and B are deployed on the
    same node then
15          Timed_Transitions += "t_B ind" + 1/ overhead
                                 cost
15  else
            Timed_Transitions += "t_B ind" + 1/ (overhead
                                 cost + communication cost)
16  if CollaborationRoles A and C are deployed on the same
    node then
17          Timed_Transitions += "t_C ind " + 1/ overhead
                                 cost
18  else
19          Timed_Transitions += "t_C ind"+1/ (overhead
                                 cost + communication cost)
20  Input_Arcs += "Pr_A do_A 1"
21  Input_Arcs += "Prd_A it 1"
22  Input_Arcs += "Pr_B do_B 1"
23  Input_Arcs += "Pr_C do_C 1"
24  Input_Arcs += "X_B t_B 1"
25  Input_Arcs += "X_C T_C 1"
26  Output_Arcs += "Start Pr_A 1"
27  Output_Arcs += "do_A Prd_A 1"
28  Output_Arcs += "it X_b 1"
29  Output_Arcs += "it X_c 1"
30  Output_Arcs += "t_B Pr_B 1"
31  Output_Arcs += "t_C Pr_C 1"
32  Output_Arcs += "do_B Prd_B 1"
33  Output_Arcs += "do_C Prd_C 1"
34  Inhibitor_Arcs += "Pr_A Start 1"
35  Inhibitor_Arcs += "Prd_A do_A 1"
36  Inhibitor_Arcs += "X_b it 1"
37  Inhibitor_Arcs += "X_c IT 1"
38  Inhibitor_Arcs += "Pr_B t_B 1"
39  Inhibitor_Arcs += "Pr_C t_C 1"
40  Inhibitor_Arcs += "Prd_B do_B 1"
41  Inhibitor_Arcs += "Prd_C do_C 1"
42  Print Places, Immediate_Transitions, Timed_Transitions,
    Input_Arcs, Output_Arcs, Inhibitor_Arcs
43  return
```

Algorithm 3: rule_2_b (ExecCost, CommCost, Ovrhdcost, Mappings, CollaborationRoles)

```
1   Places += "Pr_A 0"
2   Places += "Prd_A 0"
3   Places += "Pr_B 0"
4   Places += "Prd_B 0"
5   Places += "Pr_C 0"
6   Places += "Prd_C 0"
7   Places += "X_b 0"
8   Places += "X_c 0"
9   Immediate_Transitions += "it ind 1"
10  Timed_Transitions += "Start_B ind " + 1 / rate of the
                         external token generator for B
11  Timed_Transitions += "Start_C ind " + 1 / rate of the
                         external token generator for C
12  Timed_Transitions += "do_A ind " + 1 / execution cost
                         of CollaborationRoles A
13  Timed_Transitions += "do_B ind " + 1 / execution cost
                         of CollaborationRoles B
14  Timed_Transitions += "do_C ind " + 1 / execution cost
                         of CollaborationRoles C
15  if CollaborationRoles A and B are deployed on the same
    node then
16          Timed_Transitions += "t_B ind" + 1/ overhead
                                 cost
17  else
18          Timed_Transitions += "t_B ind" + 1/ (overhead
                                 cost + communication cost)
19  if CollaborationRoles A and C are deployed on the same
    node then
20          Timed_Transitions += "t_C ind" + 1/ overhead
                                 cost
21. else
22          Timed_Transitions += "t_C ind" + 1/ (overhead
                                 cost + communication cost)
23  Input_Arcs += "Pr_A do_A 1"
24  Input_Arcs += "Pr_B do_B 1"
25  Input_Arcs += "Prd_B t_B 1"
26  Input_Arcs += "X_b it 1"
27  Input_Arcs += "Pr_C do_C 1"
28  Input_Arcs += "Prd_C t_C 1"
29  Input_Arcs += "Xc it 1"
30  Output_Arcs += "it Pr_A 1"
31  Output_Arcs += "do_A Prd_A 1"
32  Output_Arcs += "Start_B Pr_B 1"
33  Output_Arcs += "do_B Prd_B 1"
34  Output_Arcs += "t_B X_b 1"
35  Output_Arcs += "Start_C Pr_C 1"
36  Output_Arcs += "do_C Prd_C 1"
37  Output_Arcs += "t_C X_c 1"
38  Inhibitor_Arcs += "Pr_B Start_B 1"
39  Inhibitor_Arcs += "Prd_B do_B 1"
40  Inhibitor_Arcs += "X_b t_B 1"
41  Inhibitor_Arcs += "Pr_C Start_C 1"
42  Inhibitor_Arcs += "Prd_C do_C 1"
43  Inhibitor_Arcs += "X_c t_C 1"
45  Inhibitor_Arcs += "Pr_A it 1"
46  Inhibitor_Arcs += "Prd_A do_A 1"
47  Print Places, Immediate_Transitions, Timed_Transitions,
        Input_Arcs, Output_Arcs, Inhibitor_Arcs
48  return
```

Algorithm 4: rule_3_ hardware_srn()

```
1       Places += "H_run 1"
2       Places += "H_fail 0"
3       Places += "H_recover 0"
4       Places += "H_backup 1"
5       Timed_Transitions += "T_fl ind" + 1/ cost for the
                transition between H_run and H_fail
6       Timed_Transitions += "T_dt ind" + 1/ cost for the
                transition between H_fail and H_recover
7       Timed_Transitions += "T_rcv ind" + + 1/ cost for
                the transition between H_recover and H_backup
8       Timed_Transitions += "T_bfl ind" + 1/ cost for the
                transition between H_backup and H_fail
9       Timed_Transitions += "T_sw ind" + 1/ cost for the
                transition between H_backup and H_run
10      Input_Arcs += "H_run T_fl 1"
11      Input_Arcs += "H_fail T_dt 1"
12      Input_Arcs += "H_recover T_rcv 1"
13      Input_Arcs += "H_backup T_sw 1"
14      Input_Arcs += "H_backup T_bfl 1"
15      Output_Arcs += "T_fl H_fail 1"
16      Output_Arcs += "T_dt H_recover 1"
17      Output_Arcs += "T_rcv H_backup 1"
18      Output_Arcs += "T_sw H_run 1"
19      Output_Arcs += "T_bfl H_fail 1"
20      Inhibitor_Arcs += "H_run T_sw 1"
21      Print Places, Timed_Transitions, Input_Arcs,
            Output_Arcs, Inhibitor_Arcs
22  return
```

Algorithm 5: rule_3_software_srn()

```
1       Places += "S_run 1"
2       Places += "S_fail 0"
3       Places += "S_recover 0"
4       Timed_Transitions += "T_sfl ind " + 1/ cost for the
                transition between S_run and S_fail
5       Timed_Transitions += "T_sdt ind " + 1/ cost for the
                transition between S_fail and S_recovery
6       Timed_Transitions += "T_srcv ind " + + 1/ cost for
                the transition between S_recover and S_run
7       Input_Arcs += "S_run T_sfl 1"
8       Input_Arcs += "S_fail T_sdt 1"
9       Input_Arcs += "S_recover T_srcv 1"
10      Output_Arcs += "T_sfl S_fail 1"
11      Output_Arcs += "T_sdt S_recover 1"
12      Output_Arcs += "T_srcv S_backup 1"
13      Print Places, Timed_Transitions, Input_Arcs,
            Output_Arcs
14      return
```

Algorithm 6: software_sync_srn()

```
1       Places += "S_run 1"
2       Places += "S_fail 0"
3       Places += "S_recover 0"
4       Places += "P_hf 0"
5       Timed_Transitions += "T_sfl ind" + 1/ cost for the
                transition between S_run and S_fail
6       Timed_Transitions += "T_sdt ind" + 1/ cost for the
                transition between S_fail and S_recover
7       Timed_Transitions += "T_srcv ind" + 1/ cost for the
                transition between S_recover and S_run
8       Timed_Transitions += "T_recv ind" + 1/ cost for the
            transition between P_hf and S_run + "guard hd_up()"
9       Immediate_Transitions += "t_hfl ind 1 guard
                hd_down()"
10      Immediate_Transitions += "t_hf ind 1 guard
                hd_down()"
11      Immediate_Transitions += "t_hfr ind 1 guard
                hd_down()"
12      Input_Arcs += "S_run T_sfl 1"
13      Input_Arcs += "S_fail T_sdt 1"
14      Input_Arcs += "S_recover T_srcv 1"
15      Input_Arcs += "S_run t_hf 1"
16      Input_Arcs += "S_fail t_hfl 1"
17      Input_Arcs += "S_recover t_hfr 1"
18      Output_Arcs += "T_sfl S_fail 1"
19      Output_Arcs += "T_sdt S_recover 1"
20      Output_Arcs += "T_srcv S_run 1"
21      Output_Arcs += "t_hfl P_hf 1"
22      Output_Arcs += "t_hf P_hf 1"
23      Output_Arcs += "t_hfr P_hf 1"
24      Output_Arcs += "T_recv S_run 1"
25      Print Places, Timed_Transitions, Immediate_Transitions,
            Input_Arcs, Output_Arcs
26      return
```

hd_up()

```
1       if place H_run has one token then
2               return TRUE
3       else
4               return FALSE
5       return
```

hd_down()

```
1       if place H_run has zero token then
2               return TRUE
3       else
4               return FALSE
5   return
```

Algorithm 7: collaboration_role_sync_srn()

```
1     Places += "Pr_A 0"
2     Places += "Prd_A 0"
3     Places += "P_fl 0"
4     Immediate_Transitions += "f_A ind 1 guard sw_down()"
5     Timed_Transitions += "Start ind" + 1 / rate of the
                           external token generator
6     Timed_Transitions += "do_A ind" + 1 / execution cost
                           of collaboration role A
7     Timed_Transitions += "End_1 ind" + 1 / rate of the End_1
                           transition
8     Timed_Transitions += "End_2 ind" + 1 / rate of the End_2
                           transition
9     Input_Arcs += "Pr_A do_A 1"
10    Input_Arcs += "Pr_A f_A 1"
11    Input_Arcs += "Prd_A End_1 1"
12    Input_Arcs += "f_A End_2 1"
13    Output_Arcs += "Start Pr_A 1"
14    Output_Arcs += "do_A Prd_A 1"
15    Output_Arcs += "f_A P_fl 1"
16    Inhibitor_Arcs += "Pr_A Start 1"
17    Inhibitor_Arcs += "Prd_A do_A 1"
18    Inhibitor_Arcs += "P_fl f_A 1"
19    Print Places, Timed_Transitions, mmediate_Transitions,
          Input_Arcs, Output_Arcs, Inhibitor_Arcs
20    return
```

sw_down()

```
1     if place H_run has zero token then
2           return TRUE
3     else
4           return FALSE
5     return
```

Algorithm 8: building_block_sync_srn()

```
1     Places += "Pr_A 0"
2     Places += "Prd_A 0"
3     Places += "Pr_B 0"
4     Places += "Prd_B 0"
5     Places += "P_fl 0"
6     Immediate_Transitions += "f_A ind 1 guard
                               sw_down()"
7     Immediate_Transitions += "f_B ind 1 guard
                               sw_down()"
8     Timed_Transitions += "do_A ind" + 1 / execution
                           cost of collaboration role A
9     Timed_Transitions += "do_A ind" + 1 / execution
                           cost of collaboration role B
10    Timed_Transitions += "Start ind" + 1 / rate of
                           Start
11    if CollaborationRoles A and B are deployed on
      the same node then
12          Timed_Transitions += "T ind" + 1/
                                 overhead cost
13    else
14          Timed_Transitions += "T ind" + 1/
                (overhead cost + communication cost)
15    Input_Arcs += "Pr_A do_A 1"
16    Input_Arcs += "Pr_A f_A 1"
17    Input_Arcs += "Prd_A T 1"
18    Input_Arcs += "Pr_B do_B 1"
19    Input_Arcs += "Prd_B f_B 1"
20    Output_Arcs += "Start Pr_A 1"
21    Output_Arcs += "f_A P_fl 1"
22    Output_Arcs += "f_B P_fl 1"
23    Output_Arcs += "do_A Prd_A 1"
24    Output_Arcs += "do_B Prd_B 1"
25    Output_Arcs += "T Pr_B 1"
26    Inhibitor_Arcs += "Pr_A Start 1"
27    Inhibitor_Arcs += "Prd_A do_A 1"
28    Inhibitor_Arcs += "Pr_B T 1"
29    Inhibitor_Arcs += "PrdB doB 1"
30    Inhibitor_Arcs += "P_fl f_A 1"
31    Inhibitor_Arcs += "P_fl f_B 1"
32    Print Places, Timed_Transitions,
      Immediate_Transitions, Input_Arcs, Output_Arcs,
      Inhibitor_Arcs
33    return
```

sw_down()

```
1     if place S_run has zero token then
2           return TRUE
3     else
4           return FALSE
5     return
```

Algorithm 9: parallal_process_sync_srn()

```
1   Places += "Pr_A 0"
2   Places += "Prd_A 0"
3   Places += "Xa_1 0"
4   Places += "Xa_2 0"
5   Places += "P_fl 0"
6   Places += "Pr_B 0"
7   Places += "Prd_B 0"
8   Places += "Pr_C 0"
9   Places += "Prd_C 0"
10  Places += "X_B 0"
11  Places += "X_C 0"
12  Immediate_Transitions += "it ind 1"
13  Immediate_Transitions += "f_BC ind 1 guard sw_up()"
14  Immediate_Transitions += "f´_BC ind 1 guard sw_down()"
15. Timed_Transitions += "Start ind" + 1 / Start transition rate
16  if CollaborationRoles A and B are deployed on the same node then
17          Timed_Transitions += "T_B ind" + 1/ overhead cost
18  else
19          Timed_Transitions += "T_B ind" + 1/ (overhead cost + communication cost)
20  if CollaborationRoles A and C are deployed on the same node then
21          Timed_Transitions += "T_C ind" + 1/ overhead cost
22  else
23          Timed_Transitions += "T_C ind" + 1/ (overhead cost + communication cost)
24  Timed_Transitions += "End ind" + 1 / End transition rate
25  Input_Arcs += "Pr_A do_A 1"
26  Input_Arcs += "Prd_A it 1"
27  Input_Arcs += "Xa_1 T_B 1"
28  Input_Arcs += "Xa_2 T_C 1"
29  Input_Arcs += "Pr_B f_BC 1"
30  Input_Arcs += "Pr_B f´_BC 1"
31  Input_Arcs += "Pr_C f_BC 1"
32  Input_Arcs += "Pr_C f´_BC 1"
33  Input_Arcs += "P_fl End 1"
34  Input_Arcs += "X_B do_B 1"
35  Input_Arcs += "X_C do_C 1"
36  Output_Arcs += "Start Pr_A 1"
37  Output_Arcs += "do_A Prd_A 1"
38  Output_Arcs += "it Xa_1 1"
39  Output_Arcs += "it Xa_2 1"
40  Output_Arcs += "T_B Pr_B 1"
41  Output_Arcs += "T_C Pr_C 1"
42  Output_Arcs += "f_BC X_B 1"
43  Output_Arcs += "f_BC XC 1"
44  Output_Arcs += "f´_BC P_fl 1"
45  Output_Arcs += "do_B Prd_B 1"
46  Output_Arcs += "do_C Prd_C 1"
47  Inhibitor_Arcs += "Pr_A Start 1"
48  Inhibitor_Arcs += "Prd_A do_A 1"
49  Inhibitor_Arcs += "Xa_1 it 1"
50  Inhibitor_Arcs += "Xa_2 it 1"
51  Inhibitor_Arcs += "Pr_B T_B 1"
52  Inhibitor_Arcs += "Pr_C T_C 1"
53  Inhibitor_Arcs += "P_fl f´_BC 1"
54  Inhibitor_Arcs += "X_B f_BC 1"
55  Inhibitor_Arcs += "X_C f_BC 1"
56  Inhibitor_Arcs += "Prd_B do_B 1"
57  Inhibitor_Arcs += "Prd_C do_C 1"
58  Print Places, Timed_Transitions, Immediate_Transitions,
        Input_Arcs, Output_Arcs, Inhibitor_Arcs
59  return
```

sw_up()

```
1   if place S_run has one token then
2           return TRUE
3   else
4           return FALSE
5   return
```

sw_down()

```
1   if place S_run has zero token then
2           return TRUE
3   else
4           return FALSE
5   return
```

Algorithm 9: basic_bulding_block_srn()

```
1       if CollaborationRoles A has a self token generator then
2               Places += "Pr_i 1"
3       else
4               Places += "Pr_i 0"
5       Places += "Prd_i 0"
6       Timed_Transitions += "do ind " + 1/execution cost for
                                        collaboration role i
7       if i is getting token from external token generator then
8               Timed_Transitions += "Start ind" + 1 / Start
                                                        rate
9               Output_Arcs += "Start Pr_i 1"
10              Inhibitor_Arcs += "Pr_i Start 1"
11              Inhibitor_Arcs += "Prd_i do 1"
12      else if i is getting token from another CollaborationRoles
13              Timed_Transitions += "Enter ind" + 1 / cost of
                                                the  transition
14              Output_Arcs += "Enter Pr_i 1"
15      else
16              Output_Arcs += "Exit Pr_i 1"
17              Input_Arcs += "Pr_i do 1"
18      if i is passing its token then
19              Timed_Transitions += "Exit ind " + 1 / rate for
                                                        Exit
20              Input_Arcs += "Prd_i Exit 1"
21              Output_Arcs += "do Prd_i 1"
22      Print Places, Timed_Transitions, Input_Arcs, Output_Arcs,
                                                Inhibitor_Arcs
23      return
```