Razib Hayat Khan

Performance and Performability Modeling Framework Considering Management of Service Components Deployment

Thesis for the degree of Philosophiae Doctor

Trondheim, May 2014

Norwegian University of Science and Technology
Faculty of Information Technology, Mathematics
and Electrical Engineering
Department of Telematics

NTNU – Trondheim
Norwegian University of
Science and Technology

NTNU
Norwegian University of Science and Technology

Thesis for the degree of Philosophiae Doctor

Faculty of Information Technology, Mathematics and Electrical Engineering
Department of Telematics

© Razib Hayat Khan

ISBN 978-82-326-0208-7 (printed ver.)
ISBN 978-82-326-0209-4 (electronic ver.)
ISSN 1503-8181

Doctoral theses at NTNU, 2014:143

Printed by NTNU-trykk

Abstract

A distributed system is a complex system. Developing complex systems is a demanding task when attempting to achieve functional and non-functional properties such as synchronization, communication, fault tolerance. These properties impose immense complexities on the design, development, and implementation of a distributed system that incur massive effort and cost a large amount of money. Therefore, it is vital to ensure that the system must satisfy the functional and non-functional properties once the system development process is finished. Once a distributed system is developed, it is very difficult, time consuming, and expensive to conduct any modification in its architecture. As a result, the quantitative analysis of a complex distributed system at the early stage of the development process is always an essential and intricate endeavor. To meet the challenge of conducting quantitative analysis at the early stage of the system development process, this thesis introduces an extensive framework for performance and performability evaluation of a distributed system. The goal of the performance modeling framework is the assessment of the non-functional properties of the distributed system at an early stage based on the system's functional description and deployment mapping of service components over an execution environment. The performability framework is the extension of the performance modeling framework. The extended part of the performability modeling framework considers the behavioral change of the system components due to failures. This later reveals how such behavioral changes affect the system performance.

The reusable specification of service components is the main specification unit of our framework. The specification of the reusable service component is realized through UML collaboration and activity. Activity diagrams are used to aid the illustration of the complete behavior of a system, which includes both local behavior of the service components and the necessary interactions among them. Reusable building blocks are collaborative in nature, which allows them to span across several participating components. The local behavior and interaction among the participating components are realized in an encapsulated way, which can be further reused to develop new applications. The assignment of service components that capture the system functional behavior of the physical components is recognized as deployment mapping. Deployment mapping has a significant impact on ensuring the non-functional properties provided by the system in a resource limited environment. This thesis also specifies the deployment mapping of service components using UML deployment diagrams. The focus of the deployment mapping is on considering the non-functional requirements such that the performance of a service or a system on a particular physical infrastructure can be assessed in a fully distributed manner and for large scale. In addition, a UML state machine diagram is utilized in our performability modeling framework to capture the dependability behavior of the system components.

To conduct the performance and performability evaluation of a distributed system, the UML model is transformed into analytic models that provide performance and performability evaluation results. The significance of using an analytical model is because of its well-established mathematical formula and the availability of model evaluation tools. We have specified an automated transformation process that is performed in an efficient and scalable way through the use of model transformation rules to achieve model transformation. To analyze the correctness of the model transformation process, we have used temporal logic, specifically cTLA, to formalize the UML specification style. This, in turn, provides the opportunity for model validation. The motivation of applying cTLA is to take advantage of its well-established method to illustrate various forms of structures and actions by exploiting a variety of operators and techniques, which is wonderfully compatible with UML collaborations, activities, deployment, and state machine diagram.

The framework is applied to artificial and real case studies to generate performance and performability results at the early stage of the system development process. The modeling process is supported by a set of tools, including Arctis and SHARPE with the incremental model checking facility. Arctis is used for specifying the system functional behavior. The evaluation of the performance and performability models generated by the framework is achieved using SHARPE.

Preface

This thesis is submitted in partial fulfillment of the requirements for the degree of the philosophiae doctor (PhD) at the Norwegian University of Science and Technology (NTNU). The work was performed at the Department of Telematics, during the period 2008-2013, and has been supervised by Prof. Poul E. Heegaard.

First of all, I would like to thank my supervisors Poul E. Heegaard for the prolific discussions, suggestions, and invaluable support during the whole period of my research. Furthermore, I would like to thank Prof. Kishor S. Trivedi at Duke University, USA and Fumio Machida, NEC, Japan for the collaboration we had and their contribution to my work and publications.

It was a great experience to teach at the department and work with Prof. Stig. Frode Mjølsnes and Prof. Danilo Gligoroski. I also would like to thank my colleagues for providing me support to work in a professional and rousing environment. It was a very nice, proficient, and amusing to work at ITEM under the shade of multi-cultural environment. Thanks to Andres, Addissu, Based, Frank, Laurent, Mate, Mauritz, Nor shahniza, Qashim, Vidar, Mona, Pål, and Randi.

Last but not Least, I would like to thank my family, my only brother, Feroz, my parents, and my dear one for their love, patience, and enormous support during my stay far way from my motherland Bangladesh for completing my research work.

Contents

List of Papers

▪ **Paper 1**

Translation from UML to Markov model: A performance modeling framework
Razib Hayat Khan, Poul E. Heegaard
Proceedings of the International Conference on Systems, Computing Sciences, and Software Engineering, Springer, 2009

▪ **Paper 2**

Translation from UML to Markov model: A performance modeling framework for managing behavior of multiple collaborative sessions and instances
Razib Hayat Khan, Poul E. Heegaard
Proceedings of the 3rd IEEE International Conference on Computer Science and Information Technology, IEEE computer society, 2010

▪ **Paper 3**

Translation from UML to SPN model: A performance modeling framework for managing behavior of multiple collaborative sessions and instances
Razib Hayat Khan, Poul E. Heegaard
Proceedings of the 2nd International Conference on Computer Design and Application, IEEE computer society, 2010

▪ **Paper 4**

A performance modeling framework incorporating cost efficient deployment of collaborating components
Razib Hayat Khan, Poul E. Heegaard
Proceedings of the 2nd International Conference on Software Technology and Engineering, IEEE computer society, 2010

▪ **Paper 5**

A performance modeling framework incorporating cost efficient deployment of multiple collaborating instances
Razib Hayat Khan, Poul E. Heegaard
Proceedings of the International Conference on Software Engineering and Computer Systems, Springer-Verlag Berling Heidelberg, 2011

▪ **Paper 6**

Derivation of Stochastic Reward Net (SRN) from UML specification considering cost efficient deployment management of collaborating service components
Razib Hayat Khan, Poul E. Heegaard
International Journal of New Computer Architectures and Their Applications (IJNCAA), The society of Digital Information and Wireless Communications, 2011

▪ **Paper 7**

From UML to SRN: A performability modeling framework considering service components deployment
Razib Hayat Khan, Fumio Machida, Poul E. Heegaard, Kishor S Trivedi
Proceedings of the 8th International Conference on Networking and Services, IARIA, 2012

▪ **Paper 8**

A performability modeling framework considering service components deployment

Razib Hayat Khan, Fumio Machida, Poul E. Heegaard, Kishor S Trivedi
International Journal on Advances in Networks and Services, 2012

▪ **Paper 9**

Software performance evaluation utilizing UML specification and SRN model and their formal representation
Razib Hayat Khan, Poul E. Heegaard
Submitted to a Journal for reviewing

Other relevant papers by the author (not included in the thesis)

Translation from UML to SPN model: A performance modeling framework
Razib Hayat Khan, Poul E. Heegaard,
Proceedings of Networked Services and Applications – Engineering, Control and Management (EUNICE), 2010
This paper is a preliminary version of the paper 3

A performance modeling framework considering service components deployment issue
Razib Hayat Khan, Poul E. Heegaard
Proceedings of the International Conference on Computer Communication and Management, IACSIT press, 2011
This paper is the preliminary version of the paper 6

Performance modeling of distributed system using SPN
Razib Hayat Khan, Poul E. Heegaard, Petri Nets, InTech publications, 2012
This is a book chapter based on the paper 5

From UML to SRN: A tool based support for performability modeling of distributed system considering reusable software components
Razib Hayat Khan, Poul E. Heegaard, Fumio Machida
Proceedings of the IASTED International Conference on Modeling and Simulation, 2012
This paper is the preliminary version of the paper 8

Nomenclatures

The table below summarizes the notational elements used in Part I.

Notation	Description
A	Set of arcs connecting Φ and T in a SRN model
C	Set of components in a service S
f_{B_j}	Overhead cost of a collaboration
f_{c_i}	Execution cost of an instance
f_{k_j}	Communication cost of a collaboration
F(M)	A function to evaluate the cost of a given deployment
F_K(M)	Communication cost of a mapping
I()	Indicator function
K	Set of collaborations in a service S
$\hat{l}_n$	Total execution load of a physical node
m	Marking that denotes the number of tokens for each place in Φ
m_0	Initial marking for each place in Φ
M	Deployment mapping C→N for a service
Ml	Multiplicity associated with the arcs in A in a SRN model
N	Set of all exiting physical nodes
q_0(M, c)	A function returns the physical node n from a set of physical nodes N available in the network that host component in the list mapping M
S	Set of services to deploy
T	Finite set of transitions in a SRN model
TT	Specifies the type of the each transition in set a T in a SRN model
T_a	Global load balancing estimate
Φ	Finite set of places in a SRN model

Definition of Terms

Term	Definition
Collaborative building block	UML collaboration is the main specification unit of the collaborative building block which captures the interaction between the software components. The behavioral aspect of the collaborative building is defined by the UML activity.
Collaboration role	Software component is defined as collaboration role.
Deployment mapping	The allocation of software components to the available physical resources of the system is defined as deployment mapping.
Distributed system	A distributed system consists of multiple autonomous computers that communicate through a computer network. The computers interact with each other in order to achieve a common goal.
Distributed software system	A computer program that runs in a distributed system is called a distributed software system.
Domain	A domain in software engineering is a conceptual model of all the topics related to a specific problem. It describes the various entities, their attributes, roles, and relationships, plus the constraints that govern the problem domain.
Model synchronization	Model synchronization guides performance SRN to synchronize with the dependability SRN by identifying the transitions in the dependability SRN.
Model transformation	To transform the UML model into analytical model (e.g, markov, SPN, SRN) is defined as model transformation.
Reusable building block	Collaborative building block is called as reusable building block as it is archived in a library for later reuse.
Self-contained encapsulated building block	Collaborative building block is also defined as self-contained encapsulated building block as each building block captures the local behavior and interaction between the software components in it. Each building block is self described and independent to each other.
Service	A software system to achieve some goals in a computing environment.
Service turnaround time	Service turnaround time may simply deal with the total time it takes for a service to provide the required output to the user after the service is started [122].
Software application	Software application, also known as an application or an app, is computer software designed to help the user to perform specific tasks.
Software component	A software component is a module that encapsulates a set of related functions.

Software system	Software system consists of a number of separate programs, configuration files, which are used to set up these programs, system documentation, which describes the structure of the system, and user documentation, which explains how to use the system.
System	A system is a set of interacting or interdependent components forming an integrated whole.
System bottleneck	A system bottleneck is a phenomenon where the performance or capacity of an entire system is limited by a single or limited number of physical components or resources.
System physical component	System physical component is the device or node where the software components deploy.
System throughput	System throughput is the sum of the data rates that are delivered to all terminals in a network.
Star graph	In graph theory, a star S_k is the complete bipartite graph $K_{1,k}$: a tree with one internal node and k leaves.
UML Profile	Profile in UML is defined using stereotypes, tag definitions, and constraints 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 UML for a particular domain or platform.

Abbreviations

AD	Activity Diagram
ADAGE	Ad-hoc Data Grids Environment
ADL	Architecture Description Language
ASM	Abstract State Machine
AUML	Agent UML
BPEL	Business Process Execution Language
BPMI	Business Process Management Initiative
BPMN	Business Process Modeling Notation
CASE	Computer Aided Software Engineering
CBD	Component Based Development
CLPFD	Constraint Logic Programming Finite Domain
CORBA	Common Object Request Broker Architecture
CPU	Central Processing Unit
CSP	Communication Sequential Process
CSM	Core Scenario Model
CSMA/CD	Carrier Sense Multiple Access with Collision Detection
CSMA/DCR	Carrier Sense Multiple Access with Deterministic Collision Resolution
CSP	Communicating Sequential Processes
CTMC	Continuous Time Markov Chain
cTLA	compositional Temporal Logic of Actions
DSSA	Domain Specific Software Architecture
DTMC	Discrete Time Markov Chain
EQN	Extended Queuing Network
GSMP	Generalized Semi-Markov Process
GSPN	Generalized Stochastic Petri Net
GUI	Graphical User Interface
HPC	High Performance Computing
IDE	Integrated Development Environment
ITN	Intermodal Transportation Network
JMT	Java Modeling Tool
LQN	Layered Queuing Network
LTSA	Labelled Transition System Analyser
MARTE	Modeling and Analysis of Real Time Embedded Systems
MDA	Model Driven Architecture
MOF	Meta Object Facility
MM	Machine Model

NFP	Non-functional Parameters
OCL	Object Constraint Language
OMG	Object Management Group
OPN	Object Petri Net
OPNM	Object Petri Net Model
P2P	Peer to Peer
PEPA	Performance Evaluation Process Algebra
POOSL	Parallel Object-Oriented Specification Language
PRISM	Probabilistic Symbolic Model Checker
PSM	Performance Specific Model
ProSPEX	Protocol Software Performance Engineering using XMI
PUMA	Performance by Unified Model Analysis
QoS	Quality of Service
QVT	Query View Transformation
Q-WSDL	QoS enabled Web Service Definition Language
RRB	Recurrence Relation Based
RSA	Rational Software Architect
rt_EFSM	real-time Extended Finite State Machine
SAN	Storage Area Network
SAN	Stochastic Activity Network
SHARPE	Symbolic Hierarchical Automated Reliability / Performance Evaluator
SHE	Software/Hardware Engineering
SLA	Service Level Agreement
SPE	Software Performance Engineering
SPL	Software Product Line
SPN	Stochastic Petri Net
SPNP	Stochastic Petri Net Packages
SPT	Schedulability, Performance, and Time
SysML	Systems Modeling Language
SRN	Stochastic Reward Net
STM	State Machine
TLA	Temporal Logic of Actions
TLC	Temporal Logic Checker
UML	Unified Modeling Language
XMI	XML Metadata Interchange
XML	Extensible Markup Language

Part I

Thesis Introduction and Overview

CHAPTER 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 thesis). 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 mapped onto a physical, resource-constrained infrastructure.

Quantitative analysis is realized by conducting a performance and performability 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- and performability-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 system performance and performability.

1.1 Problem description

Being able to assess the performance and performability 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 this system is expected to be executed on, including the resource constraints

- Performance and performability attributes for the system, including acceptable thresholds (e.g., given as QoS parameters in a 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 and performability evaluation to produce pertinent results in accordance with the real system behavior

To develop a framework that addresses these challenges, we need to focus on the following:

- **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 and performability of a distributed system

- Physical resource constraints to capture the effect of different **deployment strategies**

- **Non-functional properties** that reflect the performance and performability attributes of the system

- **Performance and performability evaluation approach** to uncover meaningful evaluation results when the real system does not exist

Functional behavior: We use the collaborative method to specify the system functional behavior and the coordination among the components of the system. We envision an approach in which collaborations, that is, the local behaviors of participating components, as well as the necessary interactions related to a certain distributed function or task, are the major specification units. In particular, we want to model collaboration in the form of encapsulated building blocks in a self-contained way that can easily be composed with each other [1]. 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. In particular, capturing the properties of the system into collaborative building blocks will allow system developers to reuse those building blocks. 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 with the inner complexity.

- Collaborative building blocks might be combined into more comprehensive services. This means that new services can be developed through 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.

Deployment strategies: An efficient allocation of the service components over the physical infrastructure (execution environment) is crucial to achieving good performance and performability. 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 [2]. The resulting deployment mapping has a large influence on the QoS that will be provided by the system.

In this thesis, the functional behavior of the system and the representation of the system physical infrastructure are modeled by the UML, which is a universally used modeling specification language that is widely accepted by the scientific community [3]. The thesis does not develop optimal deployment strategies but rather gives a framework for the performance and performability of a given deployment that can be specified and assessed.

Non-functional properties: In this thesis, the resource constraints and non-functional properties are included in the UML models by the use of UML profiles, specifically the following:

- *UML profile for MARTE: Modeling and Analysis of Real-Time Embedded System*

- *UML profile for Modeling Quality of Service and Fault Tolerance Characteristics and Mechanisms*

These UML profiles provide stereotypes and tagged values that are used for quantitative prediction of system performance and performability taking into consideration both hardware and software characteristics [4] [9]. The profiles and their use will be explained in the next chapter.

Performance and performability evaluation approach: Performance is the *ability of a system to provide the resources needed to deliver its services*, whereas performability is the *performance of a system where the resources needed to deliver its service might fail* [122]. Several approaches exist for conducting performance and performability evaluations of systems [5]:

- *Analytical approach*: This method is fast and easy to use when mathematical models can be formulated and solved, but the method suffers from too restrictive modeling assumptions that make the results invalid and not applicable for assessment of the quality of the service. In addition, it sometimes is difficult to validate the results produced by this approach.

- *Measurements on the prototype or real system*: The results produced by this method are convincing and reliable as the method is not realized by any simplification or assumption. However, as the measurement using this method is based on a prototype or real system, it is only possible to apply this method when a prototype or real system exists.

- *Simulation-based approach*: The method can be applied with an arbitrary level of detail to produce realistic results. It provides flexibility in modeling in case of any changes to the system. However, the approach is very demanding with respect to computational time. Another challenge is to decide on how much detail should be included, which is relevant for the evaluation of the system.

Among all the approaches, the analytical approaches are attractive because of the existence of well-established formulas and the availability of analysis tools for conducting quantitative system evaluation. This, in turn, makes the analytical approach easy to implement and evaluate and provides an accurate evaluation results for system performance and performability when the assumptions reflects the real system and its services. Moreover, 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 approach is restrictive modeling assumptions that sometimes hinder capturing the real behavior of the system. This challenge can be met by the use of the assessment model at the end, which can be simulated 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 an erroneous result during the early assessment, which can result in the incorrect development and implementation of the whole system. This incorrect development requires changes in the real system after being built, which incurs waste of effort and cost. 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.

Considering the above-mentioned factors, the general structure of the performance and performability modeling framework for a distributed system is illustrated in Figure 1.1. The figure shows the framework where the service is evaluated by an analytic approach. The same framework can be applied with simulations, but in this thesis, the focus is on analytic models[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 specification
- Information regarding system execution environment
- Non-functional parameters for quantitative analysis.

The model representations of the functional behavior, physical platform, and non-functional properties are combined to form annotated models using the UML profiles described above. Then, a given deployment of the service components onto the currently available physical resource (assumed static throughout the evaluation) is added, using annotated models. Finally, this deployment specification is automatically translated

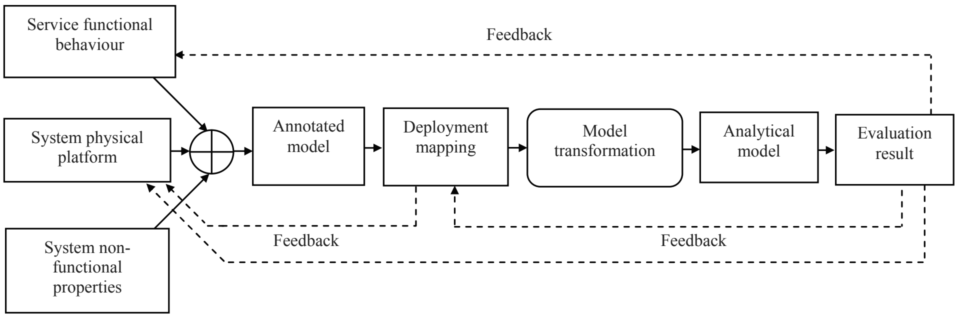


Figure 1.1 General structure of the performance and performability framework

into an analytic model, where the performance and performability of the services can be evaluated. If necessary, the evaluation results can be fed into the system design model to identify the performance anti-patterns that might cause performance problems. When

[1] The SRN models used in this thesis can be solved both by analytic approaches and by simulations.

using the framework given in Figure 1.1, several feedback loops will exist, depending on the objective of the case study. The feedbacks are as follows:

- System evaluation results can be utilized to identify any discrepancy in the model that demonstrates system functional behavior (feedback from “Evaluation result” to “Service functional behavior”).

- The deployment mapping might reveal that there is no feasible solution, forcing the alteration of the physical infrastructure (feedback from “Deployment mapping” to “System physical platform”).

- Different deployment strategies can be attempted on the same physical platform with the same service components, and then, the automated translation to analytic model and corresponding assessment is conducted for each deployment (feedback from “Evaluation result” to “Deployment mapping”).

- Performance and performability parameters sensitivity can also be checked for different resource constraints in the physical infrastructure (feedback from “Evaluation result” to “System physical platform”).

1.2 Research objective and questions

The objective of this research is to generate an extensive modeling framework that provides an automatic transformation process from UML models to analytical models that can evaluate the performance and performability of the services and systems. The transformation must be scalable and efficient in such a way that it can hide the intricate system details.

The framework is given in Figure 1.1. The main focus of this PhD work is on the framework, mostly without the feedback loops as indicated in Figure 1.2. Although the case studies included in the thesis demonstrate the applicability of the framework, a more extensive study of the applicability of the framework is regarded as future work.

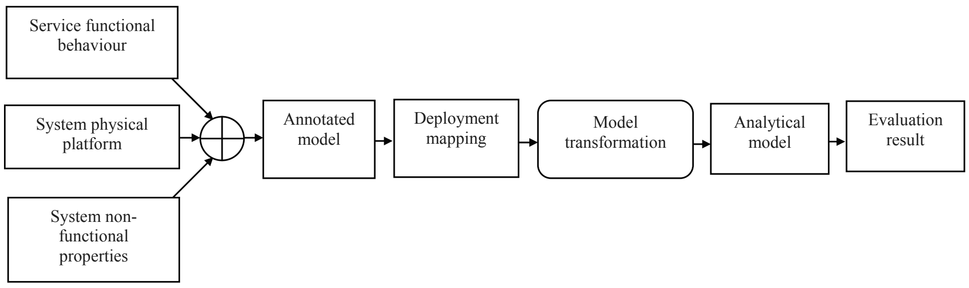


Figure 1.2 High level overview of the modeling framework used in this thesis

Research questions related to the objective are as follows:

- What is the method that will allow us to provide a rapid way to specify the functional behavior of a distributed system that can easily be combined with a model of physical infrastructure to represent deployment strategies?

- How can the deployment mapping of software components be specified considering the QoS requirements such that the performance of a service or a system over a particular physical infrastructure with resource constraints can be assessed?

- How do we incorporate non-functional properties into UML models that reflect the performance and performability attributes of the system?

- How can building blocks from the functional behavior models be translated to building blocks in performance and performability models?

- How do we conduct the automated model transformation in a scalable way to accomplish performance and performability evaluation of the system?

- How do we ensure that we obtain the complete set of model transformation rules?

- How can the correctness of the UML model specifications and model transformations be ensured?

1.3 Research method

To accomplish the research objective and obtain the research results, it is necessary to use one or more scientific techniques. These will provide methods using existing research and allowing the acquisition of new knowledge with this whole work. Our preferred approach is literature studies and scenario-driven methodology in particular to identify the criteria and valuation methods for the performance and performability framework (see Figure 1.3). We construct the framework according to the criteria that are the most imperative based on the literature study to provide improvements and to test the framework on realistic scenarios.

To establish the criteria and building assumptions to fulfill the requirements of our research, we conduct a knowledge gathering phase through literature study of existing works and approaches and also by considering real case scenarios. Subsequently, the design of the performance and performability modeling framework will be defined. The expected outcome by following our methodology is an architecture model that will be used in performance and performability modeling for distributed systems. The design of the framework will be accomplished in accordance with the research objective, which will be tested against realistic scenarios.

Later, the performance and performability framework will be implemented, and validation and testing will be conducted to satisfy the research objective and goal. The testing phase is devoted to analysis of the developed modeling framework. The results of this analysis will be compared against the performance and performability requirements, and a correction action is eventually taken (*Re-Engineering*) with the supervision of the modeler. This research methodology is followed to ensure that the objectives are met and the expected outcome will be reached at the end.

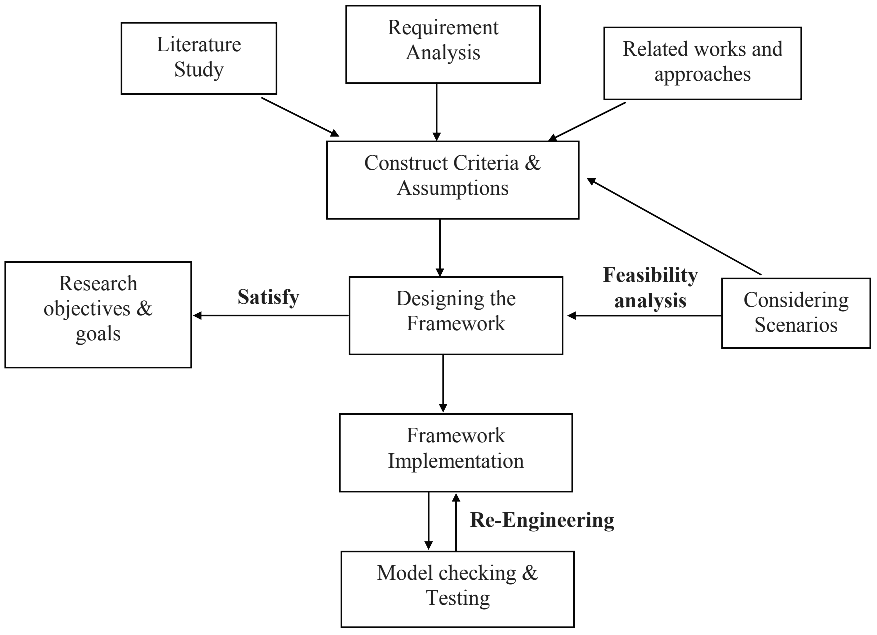


Figure 1.3 Research method

1.4 Contribution

Considering the above research questions and objective, the contributions of this thesis are mentioned below and are achieved through 9 publications presented in Part II:

- The reusable specification of collaborative building blocks is utilized and the specification is formalized to generate Markov and Petri net models through our performance and performability modeling framework (focused on Paper 1, Paper 3, Paper 4, Paper 5, Paper 6, and Paper 7) [23].

- The reusable specification of collaborative building blocks is utilized and formalized to generate the Markov and Petri net models for multiple collaborative sessions and instances that occur at the same time (focused on Paper 2 and Paper 3) [23].

- The deployment mapping of the software components is specified by considering QoS requirements so that the performance of a system on a particular physical topology can be assessed (focused on Paper 4, Paper 5, and Paper 6).

- An approach is introduced to incorporate non-functional properties into UML models that reflect the performance and performability attributes of the system (focused on all the Papers included in Part II).

- A scalable approach is introduced for automatically conducting the model transformation process by using the reusable model transformation rules (Paper 6, Paper 7, Paper 8, and Paper 9).

- Our performance and performability modeling framework that provides tool support of the framework, is implemented (Paper 8 and Paper 9).

- The feasibility and consistency of our approach is described with the help of artificial and real case studies (focused on all the Papers included in Part II).

During our research, we worked on several case studies and examples that are presented in the papers of Part II.

- Paper 1 and Paper 2 present an example that utilizes a system description where users that are equipped with smart phones want to receive current location weather information using their hand-held devices.

- Paper 3 presents an example that utilizes a system description where several users that are equipped with smart phones want to receive current location weather information using their hand-held devices after performing authorization checks based on user identity.

- In Paper 4, Paper 5, Paper 6, and Paper 9, we consider a scenario adopted from Efe dealing with the heuristically clustering of modules and the assignment of clusters to nodes [6]. This scenario, even though artificial and potentially lacking tangibility from a designer’s point of view, is sufficiently complex to demonstrate the applicability of our framework.

- Paper 7 and Paper 8 consider an example dealing with heuristically clustering of modules and assignment of clusters to nodes and is adopted from [6].

- Paper 9 also introduces a real case study, the 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 calls. The orders are processed by the call centre, which then sends out tour requests to the taxis.

1.5 Structure of the thesis

Part I: Introduction and Overview

Part I continues in Chapter 2 where a research approach is described that focuses on the key points of the work, including the detailed description of the developed performance and performability modeling framework. The summary of the contribution is illustrated in Chapter 3 concentrating on how the included articles are interrelated and providing guidelines for reading. Chapter 4 reviews the related works with a following discussion in Chapter 5. Part I of the thesis ends with Chapter 6, where some of the on-going works and future directions have been outlined.

Part II: Included Publications

Part II contains 6 peer-reviewed conference and 3 journal articles (one is submitted) that are presented in an order that builds up the core of the research approach and expands successively through several examples.

Part III: Appendix

Appendix A contains a table that includes a list of related approaches.

CHAPTER 2

Performance & performability modeling framework

In this chapter, we describe our modeling framework in detail with focuses on the specification, semantics, reasoning, and algorithms that form the base of our performance and performability modeling approach. In Figure 1.2, the overview of our approach for distributed system performance and performability modeling is illustrated. Sections 2.1 - 2.6 illustrate the steps of the framework by highlighting the gradual development process of the modeling framework. Section 2.7 highlights the formalization of the input models, whereas in Section 2.8, the tool support of our framework is given, and finally, Section 2.9 is devoted to discussing some critical issues.

2.1 Service functional behavior

We adopt a model-driven approach, where the functional behavior of service is specified using UML, which is widely used and accepted by the software engineering community. UML provides a set of diagrams that facilitate illustration of system behavior from different viewpoints and from different detail levels. The papers of part II describe the UML specification style with illustrated examples. Therefore, we will present the key points and features of the UML specification style in this section.

UML collaboration is utilized as the main specification unit of our work 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. An example collaboration diagram is shown in Figure 2.2, where c_i and c_j are the collaboration roles (illustrated in the dashed rectangle), and the interactions among them are defined as $k_{i,j}$ (demonstrated in the oval).

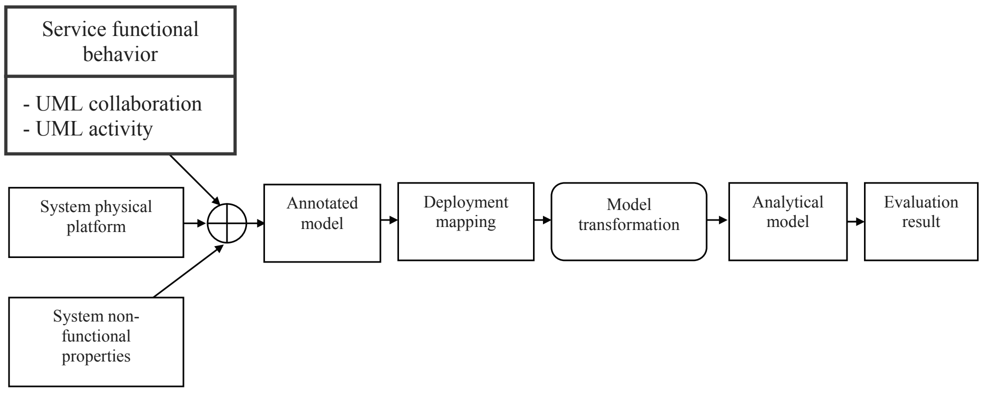


Figure 2.1 Service functional behavior using UML collaboration and activity diagram

The collaboration diagram example is the basic example that is used throughout our work. 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.

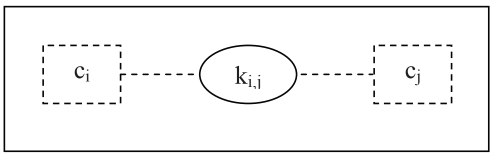
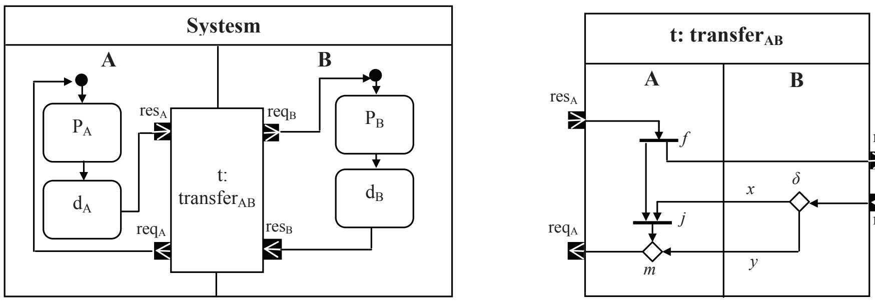


Figure 2.2 Basic collaboration diagram example

As the service specification later will be transformed into a performance and performability 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. To remove this shortcoming, we use UML activity diagrams, which define the internal behavior of the collaboration as well as the detailed behavior of how different events of collaboration roles are coupled. Figure 2.3 illustrates both the detailed behavior of the collaboration roles and the internal behavior of the collaboration. The specifications for collaborations in our work are given as coherent, self-contained building blocks. 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 every collaboration, the activity declares a corresponding call behavior action referring to the activities of the employed building block, which is illustrated in Figure 2.3(b). The activity $transfer_{ij}$ (*where* $ij = AB$) describes the behavior of the corresponding collaboration. It has one activity partition for each collaboration role: *A* and *B*. Activities base their semantics on token flow [1]. The activity starts by placing a token when there is a response (indicated by the streaming pin res_A or res_B) to transfer by either participant A or B. After completion of the processing by the collaboration role A, the token passes through the fork node f, where the flow is divided into two branches. One branch is directly forwarded to the streaming pin req_B as a request, which is sent to the collaboration role B. Another flow is directed to the join node j. After completion of the processing by the collaboration role B, the token passes through the decision node δ, where only one branch, either x or y, will be activated. If the flow marked with x activates, it will then pass through the join node j. If both the incoming flows of the join node j arrive, the join node j will be activated. If the flow marked with y activates, it will then pass through the merge node m. The outgoing flow of the merge node will be activated when either of the incoming flows arrives.

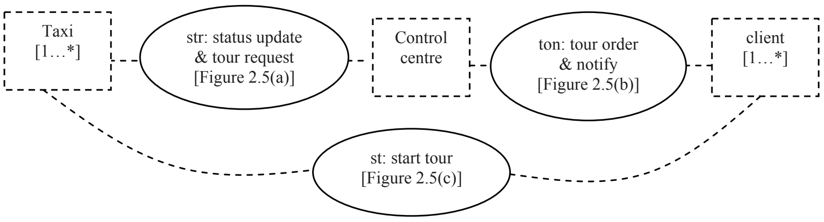


(a) (b)

Figure 2.3(a) Detailed behavior of the collaboration roles
(b) Internal behavior of the collaboration

For delineating the detailed behavior of 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 [12]. 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. For example, the detailed behavior of collaboration is given in Figure 2.3(a).

Case example: Taxi control system: service functional behavior: *Here, the service specification style using UML collaboration and activity is demonstrated using a real case scenario. A real case scenario has been considered, the 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. Figure 2.4 illustrates the scenario as UML collaboration. Participants in the service are represented by the collaboration roles taxi, control centre, 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****.*



Taxi [1…*]
str: status update & tour request [Figure 2.5(a)]
Control centre
ton: tour order & notify [Figure 2.5(b)]
client [1…*]
st: start tour [Figure 2.5(c)]

Figure 2.4 Collaborations and components in the taxi control system (references in bracket to the figures that contain more details)

The internal behavior of the collaboration ***status update & tour request****,* ***tour order & notify****, and* ***start tour*** *are demonstrated using UML activity, which are shown in Figure 2.5. The specifications for the collaborations are given as coherent, self-contained building blocks and have one activity partition for each collaboration role.*

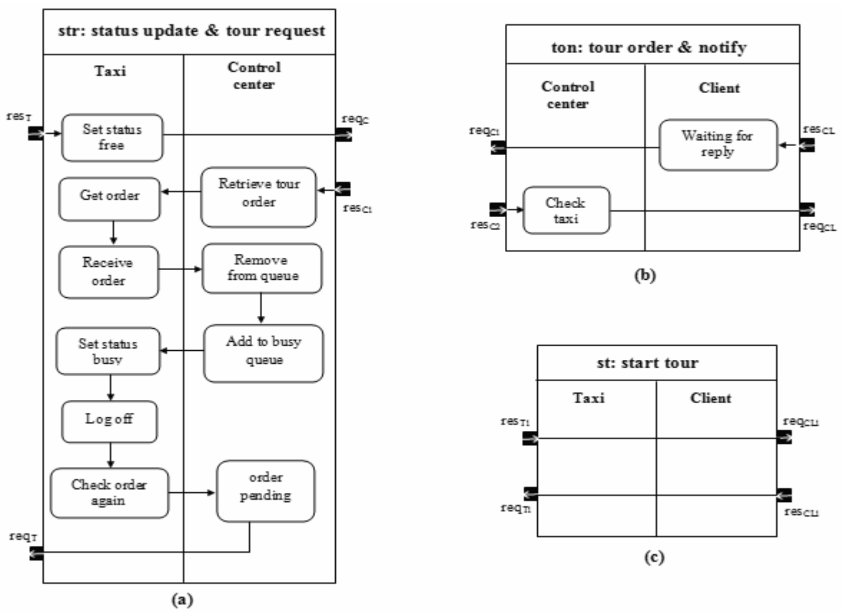


Figure 2.5 Internal behavior of the collaboration using UML activity

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 pins, streaming pins, and terminating pins. The pins shown in Figure 2.5 (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. The detailed behavior of the taxi control system is shown in Figure 2.6, which is mainly composed of the collaborations demonstrated in Figure 2.5. 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

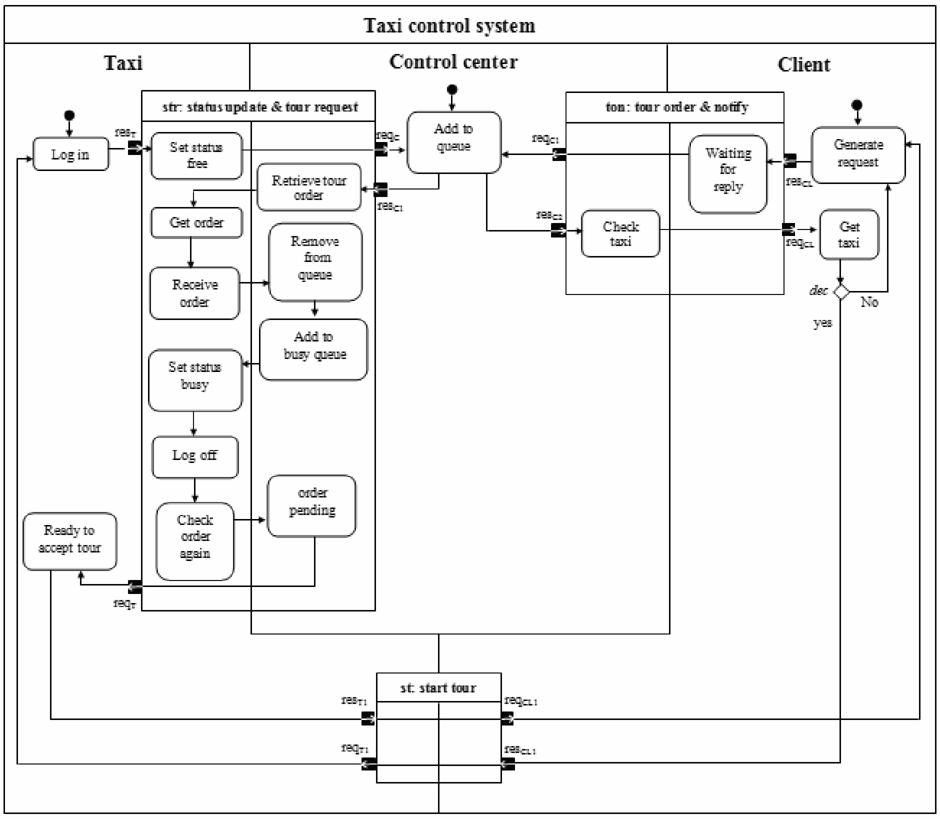


Figure 2.6 Detailed illustration of the service behavior using UML activity

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.

2.2 System physical platform

Specification of the system physical platform is incorporated into our modeling framework by the use of UML deployment and UML STM diagrams.

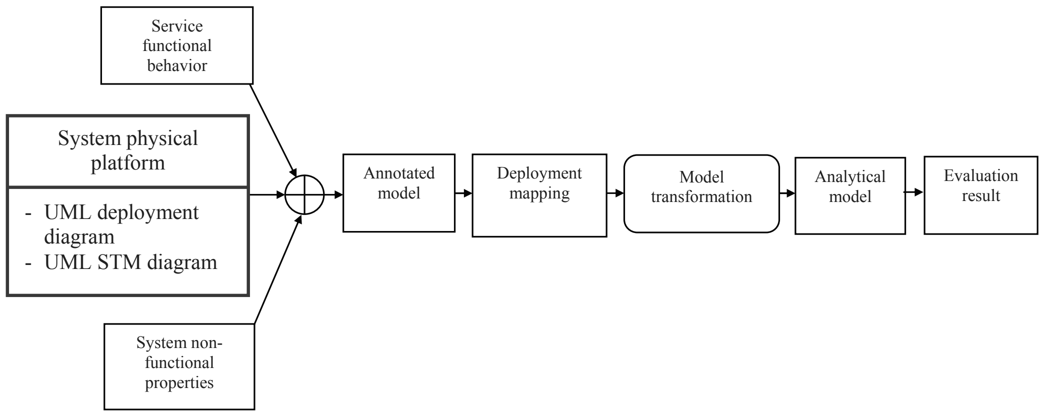


Figure 2.7 System physical platform using UML deployment diagram

2.2.1 UML deployment diagram

The static view of the system physical platform is illustrated using 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 connection between physical components, and the assignment of software artifacts to those identified physical components [3]. 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.

Case example: Taxi control system: system physical platform: *An example of the UML deployment diagram for the taxi control system is illustrated in Figure 2.8, where the system components taxi and the user mobile device are connected with the control centre via a wireless communication channel.*

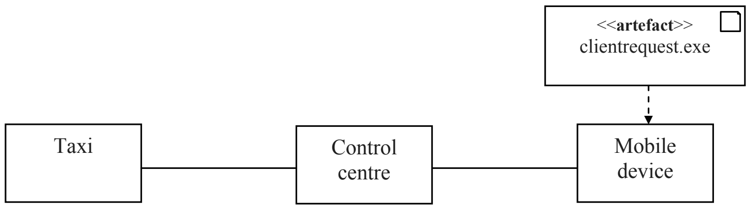


Figure 2.8 Physical platform of taxi control system using UML deployment diagram

2.2.2 UML STM diagram

The dynamic view of the system components that are deployed in the system execution environment is illustrated using a UML STM diagram. In our modeling framework, the UML STM diagram is used to highlight the dependability properties such as failure and recovery events for the software and hardware components for which changes in the states of the system components happen. The dependability properties of a system address the representation of changes in the states of the system components being modeled, which are generally due to faults, and how such changes affect the availability of the system. In an STM, a state is depicted as a rectangle, and a transition from one state to another state is represented by an arrow.

Example: dependability behavior of a hardware component: *An example STM of a hardware process is illustrated in Figure 2.9, which sketches the states and transitions from one state to another. A hardware process starts in an initial state, represented by the closed circle, and enters a Running state. If the server process fails during the operation, the process enters the Failed state. The process moves to the detected state when the failure is detected by a monitoring mechanism. The process subsequently enters*

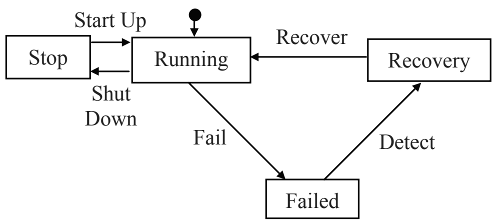


Figure 2.9 STM diagram of hardware process

the Recovery state and returns to the Running state after the recovery process finishes. The server starts the operation when the start-up command is invoked, and then the process enters the Running state.

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, tag definitions, and constraints 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 [4]. Stereotypes permit us to map

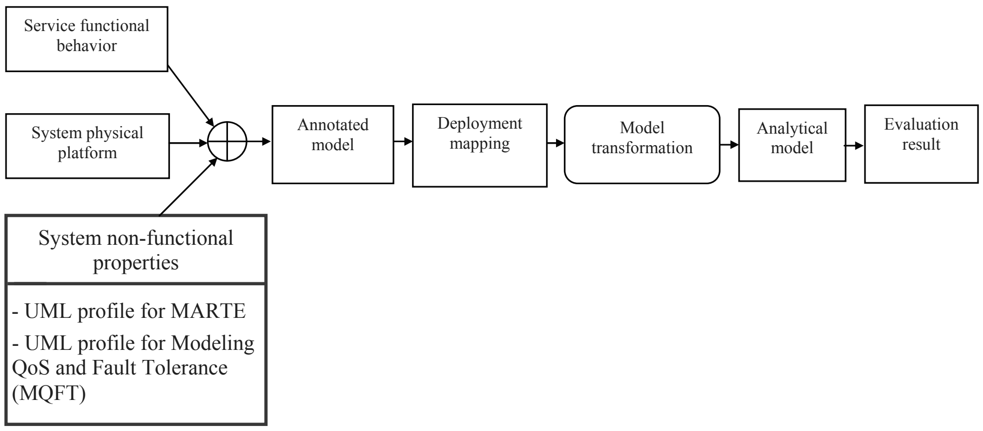


Figure 2.10 System non-functional properties

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 above-mentioned profile. Tagged values are a type of value slot associated with the attributes of specific UML stereotypes [4]. The significance for using UML profiles is as follows:

- As much as possible, modelers should not be hindered in the way they use UML to represent their systems just to be able to do model analysis. That is, rather than enforcing a specific approach or modeling style for real-time systems, the profile should allow modelers to choose the style and modeling constructs that they feel are the best fit for their needs of the moment.

- Profiles provide 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.

- 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.

- The profiles must support all the current mainstream real-time technologies, design paradigms, and model analysis techniques. However, profiles should also be fully open to new developments in all of these areas.

We use two UML profiles throughout the whole work for describing the non-functional properties of the system:

- *UML profile for MARTE – Modeling and Analysis of Real-Time Embedded Systems*

- *UML profile for Modeling Quality of Service and Fault Tolerance Characteristics and Mechanisms Specification*

The *UML profile for MARTE* is intended to replace the existing *UML Profile for Schedulability, Performance, and Time* [21]. MARTE defines foundations for model-

Table 2.1 Stereotypes and tagged values of MARTE [4]

UML profile for MARTE: Modeling and Analysis of Real Time Embedded Systems	
Stereotypes with definitions	
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.
Tagged Values with definitions	
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.

based descriptions of real-time and embedded systems. These core concepts are then refined for both modeling and analyzing concerns. 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. In this context, the intent is not to define new techniques for analyzing real-time and embedded systems but to support them. Hence, it provides facilities to annotate models with information required to perform specific analysis. The stereotypes and tagged values according to the UML profile for *MARTE: Modeling and Analysis of Real Time Embedded Systems* used in our work are defined in Table 2.1 [4].

The specification *UML profile for Modeling Quality of Service and Fault Tolerance Characteristics and Mechanisms Specification* defines a set of UML extensions to represent the Quality of Service and Fault-Tolerance concepts. The profile provides the ability to associate the requirements and properties to UML model elements. The general profile for fault-tolerance includes notations to model risk assessments, paying special attention to the description of hazards, risks, and risk treatments. The stereotypes and tagged values according to the *UML profile for Modeling Quality of Service and Fault Tolerance Characteristics and Mechanisms Specification* used in our work are defined in Table 2.2 [9].

Table 2.2 Stereotypes and tagged values of UML profile for modeling QoS and fault tolerance [9]

UML profile for Modeling Quality of Service and Fault Tolerance Characteristics and Mechanisms	
Stereotypes with definitions	
QoSDimension	*QoSDimension* provides support for the quantification of QoS characteristics and attributes.
Tagged Values with definitions	
mean-time-to-repair	*mean-time-to-repair* defines the time required for repairing a system physical resource.
mean-time-between-failures	*mean-time-between-failures* defines the time between the consecutive failures of system physical components.

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 and also captures the dependability-related behavior of the system components. However, one shortcoming of UML is not having the capability to incorporate non-functional parameters, which is vital for conducting the quantitative analysis from 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 two specification styles, which provide several stereotypes and tagged values (see Section 2.3), throughout the whole work for incorporating performance- and dependability-related parameters into the service specification model defined by the UML. The two specification styles are as follows:

- *UML profile for MARTE – Modeling and Analysis of Real-Time Embedded Systems* to annotate the UML collaboration, activity, and deployment diagram

- *UML profile for Modeling Quality of Service and Fault Tolerance Characteristics and Mechanisms Specification* to annotate the UML STM diagram

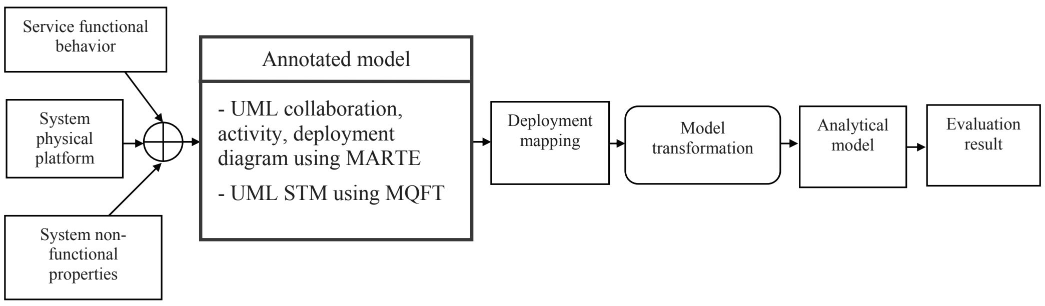


Figure 2.11 Annotation of UML models

Case example: Taxi control system: annotated model: *To illustrate the annotation of the UML model, an example of a UML activity diagram for the taxi control system is illustrated in Figure 2.12. The scenario is defined as follows: after being deployed in*

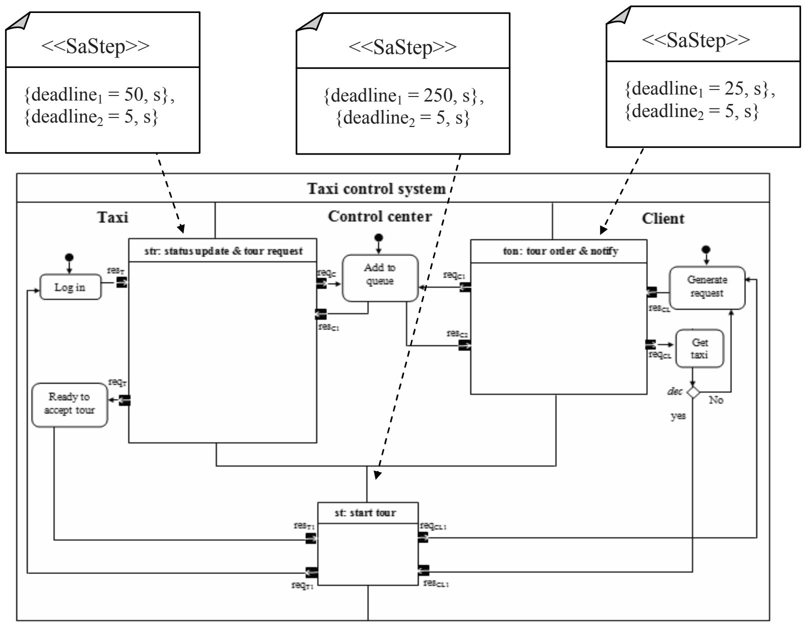


Figure 2.12 Annotation of the UML activity diagram for a taxi control system

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.

Another example of annotation of with a UML deployment diagram of the taxi control system according to the MARTE profile is demonstrated in Figure 2.13. 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 one, and the tagged value capacity indicates that the maximum number of permissible concurrent users handled by the control centre is 20.

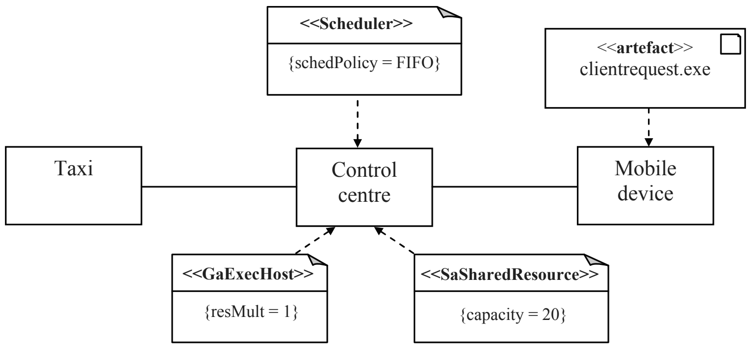
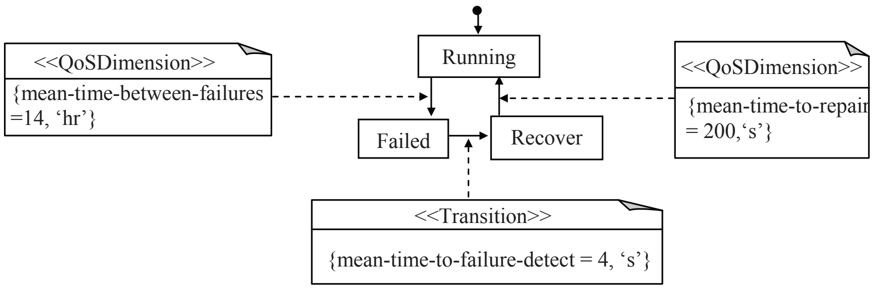


Figure 2.13 Annotation of UML deployment diagram for taxi control system

Example: annotated UML STM model: *An example of an annotated UML STM diagram of the software process according to the UML profile for Modeling Quality of Service and Fault Tolerance Characteristics and Mechanisms Specification is shown in Figure 2.14. We use the stereotypes <<QoSDimension>>, <<Transition>> and tagged values*



Running
<<QoSDimension>>
{mean-time-between-failures =14, 'hr'}
<<QoSDimension>>
{mean-time-to-repair = 200,'s'}
Failed
Recover
<<Transition>>
{mean-time-to-failure-detect = 4, 's'}

Figure 2.14 Annotation of UML STM diagram

mean-time-between-failures, mean-time-to-failure-detect and mean-time-to-repair, which are already described in Section 2.3. The initial node (•) indicates the starting of the operation of the software process. Then, the process enters the Running state. Running is the only available state in the STM. If the software process fails during the operation, the process enters the Failed state. The duration between consecutive failures is 14 hours, which is indicated by the tagged value mean-time-between-failures. When the failure is detected by the external monitoring service, the software process enters Recovery state and the repair operation will be started. The time required for detecting a failure is 4 seconds, which is specified by the tagged value mean-time-to-failure-detect. When the failure of the process is recovered, the software process returns to the Running state. The time necessary for repairing the failure is 200 seconds, which is tagged by mean-time-to-repair.

2.5 Deployment mapping

The allocation of software components to the available physical resources of the distributed system is defined as deployment mapping, which has a considerable impact on the desired QoS provided by the system. The term desired QoS is interpreted as the delivery of a service in accordance with its specification [2]. For large and multifaceted distributed systems, several combinations of deployment mapping exist that provide the same functionality, but the combinations show differences when satisfying the QoS provided by the systems. There are many QoS requirements that need to be considered for providing better deployment mapping. Focusing on one or more requirements to provide a better deployment mapping might affect the other requirements, which, in turn, produces a poorer solution. The solution becomes more difficult to derive when the questions of efficiency, dynamisms, and scalability are involved.

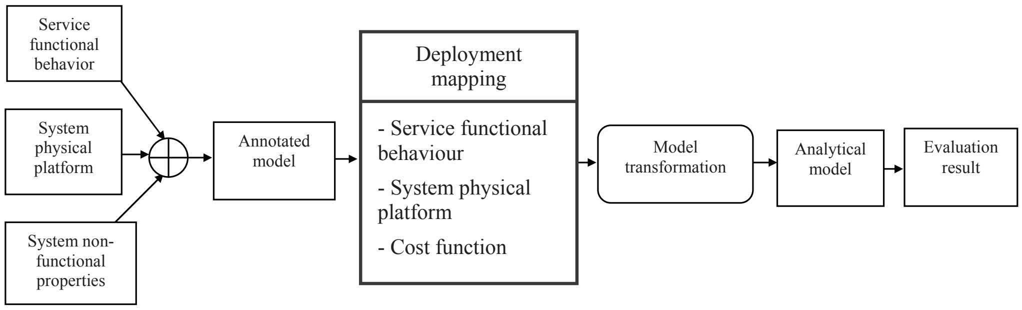
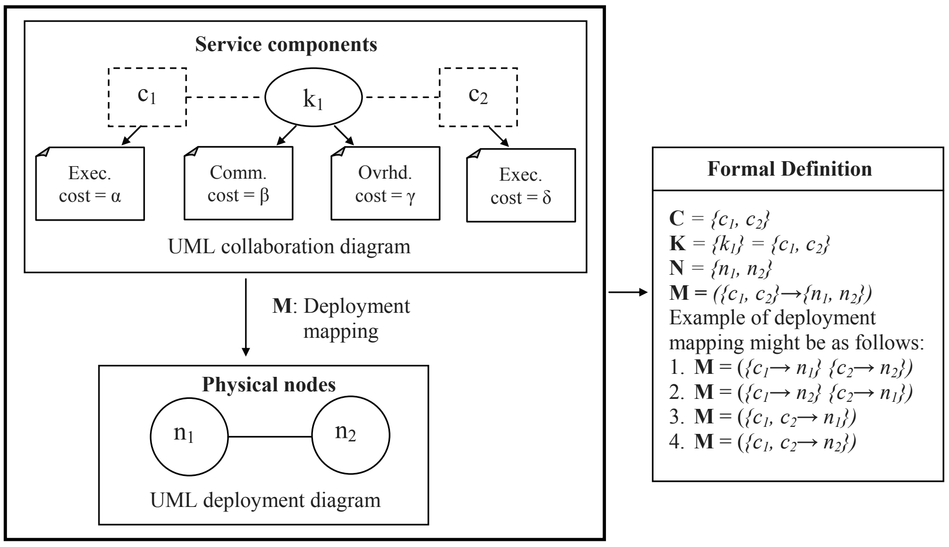


Figure 2.15 Deployment mapping of the system

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. Deployment mapping **M** is defined as [$\mathbf{M}=(C \rightarrow N)$], between a number of service components 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 settings, 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. For example, an existing collaboration between two service components is illustrated in Figure 2.16. These components are ready for deployment onto the physical nodes. The example also shows two software components (that have to be deployed) and collaboration between them, where each has a corresponding cost value. The components have costs related to their execution, e.g., memory or CPU sharing needed at the host, factored into a single value, whereas the collaborations have costs that inform about the communication needed between the components as well as the overhead costs to conduct the communication. Communication costs are composite values incorporating the volume of interaction between components, i.e., they are characterized by the amount of message interchanged and the average message length. In Figure 2.16, the target network consists of two nodes, **N**= $\{n_1, n_2\}$. There are two components to be deployed **C**= $\{c_1, c_2\}$ and one collaboration **K**= $\{k_1\}$ between them. The execution platform of services is possibly a highly distributed hardware environment consisting of physical nodes that are heterogeneous in capabilities, amount of resources, and in access rights. This network of possible execution hosts is considered to be a hybrid environment, in which services can be deployed in various clusters depending on the present conditions and usage patterns. Execution in such a dynamic and hybrid environment might be influenced by a plethora of various parameters making the search for an efficient deployment difficult. Regarding the network, it is assumed that all nodes are identical with respect to the capacity and that the network is fully interconnected. Furthermore, each participating node contains an execution runtime that encapsulates the functionalities of installation and execution of components.



Formal Definition

C = $\{c_1, c_2\}$
K = $\{k_1\} = \{c_1, c_2\}$
N = $\{n_1, n_2\}$
M = $(\{c_1, c_2\} \rightarrow \{n_1, n_2\})$
Example of deployment mapping might be as follows:
1. **M** = $(\{c_1 \rightarrow n_1\}\ \{c_2 \rightarrow n_2\})$
2. **M** = $(\{c_1 \rightarrow n_2\}\ \{c_2 \rightarrow n_1\})$
3. **M** = $(\{c_1, c_2 \rightarrow n_1\})$
4. **M** = $(\{c_1, c_2 \rightarrow n_2\})$

Figure 2.16 Deployment mapping with formal definition

Assessment of deployment mapping using cost functions: In this thesis, 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 designing cost functions, which are functions that express the utility of the deployment mapping of a service. 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 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 to conduct the communication, which are known as overhead costs.
3. Some of the service components can be restricted in the deployment mapping to specific physical nodes, which are called bound components.

Here, cost is the reward/metric of a certain resource constellation. The resource constellation is result of stochastic behavior that follows negative exponential distribution (such as request arrival, service time). It is more like a condition for the modeling approach. For example, in our system, the stream of processing requests coming for any given individual service components makes up a very small part of the total stream of processing requests to the system. It is also reasonable that the processing requests for any service component are independent of one another. So according to Palm-Khintchine theorem [123], such arguments provide a theoretical justification for modeling the stream of processing requests to a system, as a negative exponential distribution.

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 [19]. 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 thesis. 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 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**). This 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. 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 [20]. 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:

- The execution cost of each service component can be labeled f_{c_i} .
- The communication cost between the service components is labeled f_{k_j} .
- The cost for executing the background process for conducting the communication between the service components 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* 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, and thus, the total offered execution cost for the given service can be calculated as $\sum_{i=1}^{|E|} f_{c_i}$. Hence, the average load $\boldsymbol{T_a}$ becomes [2]:

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

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 [2]:

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

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**.

Let collaboration $k_1 = (c_1, c_2)$ (Figure 2.16), 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** = $(\{c_1, c_2 \rightarrow n_1\})$ or $(\{c_1, c_2 \rightarrow n_2\})$ in Figure 2.16).
- The cost is f_{k_j} if the service components are otherwise co-located (i.e., the collaboration is remote) (if **M** = $(\{c_1 \rightarrow n_1\}\ \{c_2 \rightarrow n_2\})$ or $(\{c_1 \rightarrow n_2\}\ \{c_2 \rightarrow n_1\})$ in Figure 2.16).

- 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 k_j is remote, the set of mapping **M** is used. Given the indicator function, the overall communication cost of service, $F_K(\mathbf{M})$, is the sum [2]:

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

Given a mapping $\mathbf{M} = \{m_n\}$ (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 $F(\mathbf{M})$ becomes [2] (where $I_j = 1$, if k_j external or 0 if k_j internal to a node):

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

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

Example: deployment mapping: *To specify the deployment mapping using the cost functions, we consider an example as a service of collaboration of E = 10 components or collaboration role (labeled $C_1 \ldots C_{10}$) to be deployed and K = 14 collaborations between them, as illustrated in Figure 2.17. We consider three types of requirements in this specification. In addition to the execution cost, the communication costs and the overhead cost, we have a restriction on components C_2, C_7, C_9 regarding their location. They must be bound to physical nodes $\boldsymbol{n_2}$, $\boldsymbol{n_1}$, $\boldsymbol{n_3}$ respectively.*

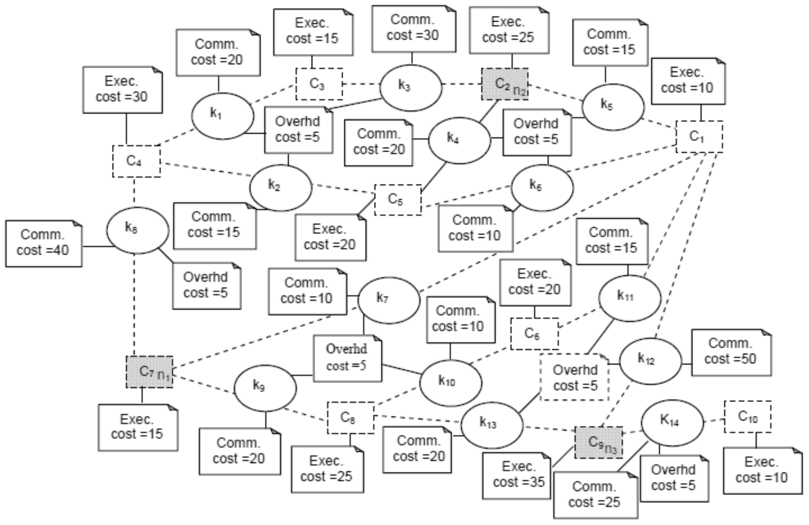


Figure 2.17 Collaborations and components in the example

Costs such as execution, communication, and overhead are estimated by guessing in the application scenario and are utilized for reasoning about the deployment logic. The development of a method that could be applied for deriving the costs automatically and in a realistic manner would be useful and is the part of our future work (see Chapter 6).

In this example, the target environment consists only of N = 3 identical, interconnected physical nodes with a single provided property, namely processing power, and infinite communication capacities (Figure 2.18).

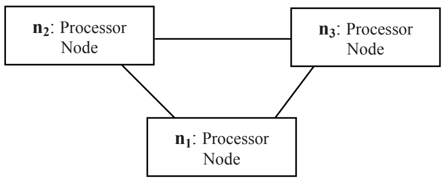


Figure 2.18 Target network of hosts

The optimal deployment mapping can be observed in Table 2.3. The lowest possible deployment cost, according to equation (4) is 17 + 100 + 70 = 187.

Table 2.3 Optimal deployment mapping in the example scenario

Node	Components	$\hat{l}_n$	$\lvert \hat{l}_n - T \rvert$	Internal collaborations
n_1	c_4, c_7, c_8	70	2	k_8, k_9
n_2	c_2, c_3, c_5	60	8	k_3, k_4
n_3	c_1, c_6, c_9, c_{10}	75	7	k_{11}, k_{12}, k_{14}
			17	100
$\sum$ cost			117	

Case example: Taxi control system: deployment mapping: *Moreover, the following Figure 2.19 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.*

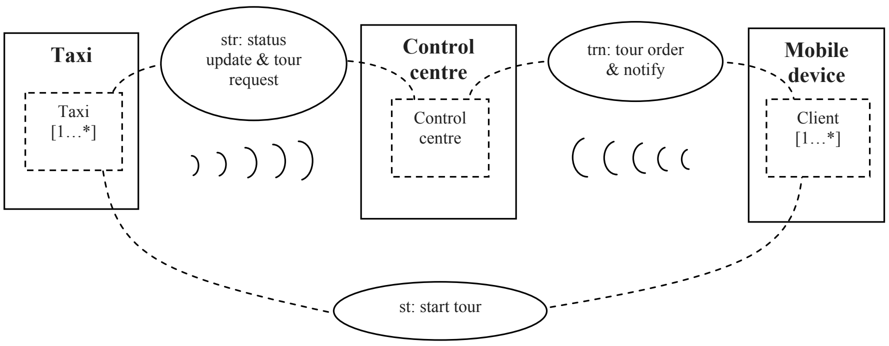


Figure 2.19 Deployment mapping of a taxi control system

2.6 Model transformation

To conduct the early assessment of performance and performability modeling, the service design defined by the UML specification is transformed into analytical models such as the Markov model and Petri net [23]. To keep the service specification model and analytical model consistent with each other, the process of model transformation is driven

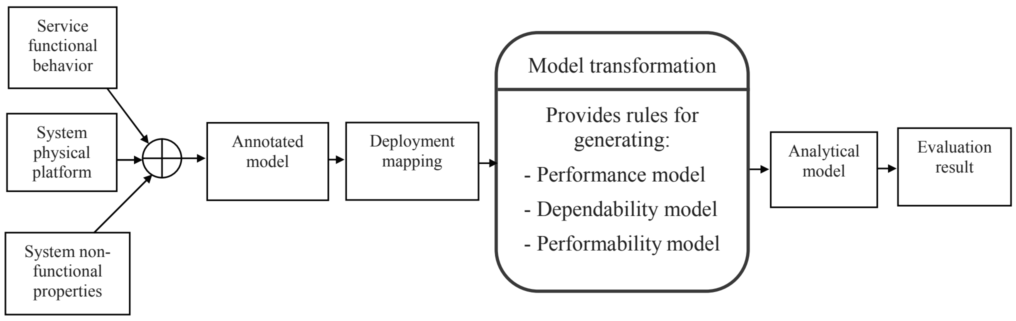


Figure 2.20 Model transformation

by the model transformation rules, which provide an efficient, scalable, and automated approach to conducting model transformation for large, complex, and multifaceted distributed systems. Though we consider the generation of a Markov model from the UML specification style in Papers 1 and 2, our ultimate goal is to generate Petri nets, specifically SRN models, from the UML specification style [22]. The reason is that Petri nets have very similar semantics to the UML activity diagrams and state machine diagrams that have been used as the main specification unit in our modeling framework, and, in particular, SRN provides several extremely useful features to model performance- and performability-related behavior well. The model transformation rules have been utilized to generate three types of analytical models in our modeling framework: performance, dependability, and performability. This section contains the description regarding the below items:

- Introduction of the Petri net model
- Performance model generation
- Dependability model generation
- Performability model generation

Before introducing the model transformation rules for generating SRN models, we would like to provide a brief introduction about Petri nets.

2.6.1 Introduction of the Petri net model

A Petri net is a directed graph with two disjoint-type nodes, such as places and transitions [23]. A directed arc connecting a place to a transition is called an input arc, and the arc connecting a transition to a place is called output arc of the transition. A positive integer called multiplicity can be associated with each arc. The places connected to a transition by input arcs are called the input places of this transition, and those connected by means of output arcs are called its output places. Each place may contain zero or more tokens in a marking. A marking represents the state of the model at a particular instant. This concept is central to Petri nets. The notation # (p, μ) is used to indicate the number of tokens in place p in marking μ. If the marking is clear from the context, the notation #p is used. A transition is enabled when each of its input places has at least as many tokens as the multiplicity of the corresponding input arc. A transition may fire when it is enabled, and on firing, a number of tokens equal to the multiplicity of the input arc are removed from each of the input places, and a number of tokens equal to the multiplicity of the output arc deposited in each of its output places [23].

SPNs are obtained by associating stochastic and timing information to Petri nets [23]. This is accomplished by attaching firing time to each transition (shown in Figure 2.21(a), where transition $T_{arrival}$ is associated with firing time, which is exponentially distributed with λ), representing the time that must elapse from the instant that the transition is enabled until the instant is actually fires. In GSPNs, transitions are allowed to be either timed (exponentially distributed firing time ($T_{arrival}$ in Figure 2.21(b))) or immediate (zero firing time (T_{quick} in Fig 2.21(b))) [23]. Another other extension of GSPNs includes inhibitor arcs. An inhibitor arc has small hollow circles instead of arrows at its terminating ends. A transition with an inhibitor arc cannot fire if the number of tokens that the input place of the inhibitor arc contains is equal to or more tokens than the multiplicity of the arc.

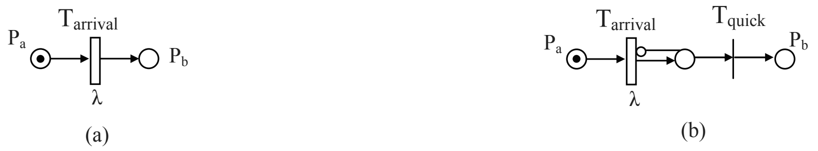


Figure 2.21(a) A simple SPN model (b) A simple GSPN model

SRNs are based on GSPNs and extend them further by introducing prominent extensions such as guard function, reward function, and marking dependent firing rate [22]. A guard function is assigned to a transition. It specifies the condition to enable or disable the transition and can use the entire state of the net rather than just the number of tokens in places [22]. It can be expressed 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) and a logical "OR" for inhibitor arcs (any inhibitor condition is sufficient to disable the transition). Reward function defines the reward rate for each tangible marking of Petri net based on which various quantitative measures can be performed at the net level. A marking-dependent firing rate allows using the number of token in a chosen place, multiplying the basic rate of the transition. An SRN model has the following elements: finite set of the place (drawn as circle), finite set of the transitions defined as either timed transitions (drawn as thick transparent bar) or immediate transitions (drawn as thin black bar), set of the arc connecting place and transition, multiplicity associated with the arcs, and marking that denotes the number of token in each place. The SRN model is described formally by the 6-tuple {***Φ, T, A, TT, Ml, m₀***} in the following manner [4]:

Φ *= Finite set of the place*
T *= Finite set of the transition*
A $\subseteq$ *{Φ × T}* $\cup$ *{T × Φ} 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* $\boldsymbol{m_0}$.

By considering the semantic definition of the SRN model, we provide the model transformation rules used in this thesis to generate performance, dependability, and performability models.

2.6.2 Performance model generation

Rules for generating performance SRN model can be divided into four categories:

- Rule 1: Deployment mapping of a collaboration role
- Rule 2: Deployment mapping of single collaboration (Bidirectional)
- Rule 3: Deployment mapping of single collaboration (Unidirectional)
- Rule 4: Deployment mapping of multiple collaborations (Unidirectional)

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 2.22 (where P_i = Processing of i^{th} collaboration role and d_i = Processing performed of the i^{th} collaboration role). Mainly, rule 1 has been utilized to model the load of a physical node. For each physical node, there must be an upper bound of 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 Figure 2.22. Initially, place PP_n contains ***q*** (where integer **q** > 0) tokens, which define the upper bound of the execution of the process in parallel with a physical node *n,* and the timed transition

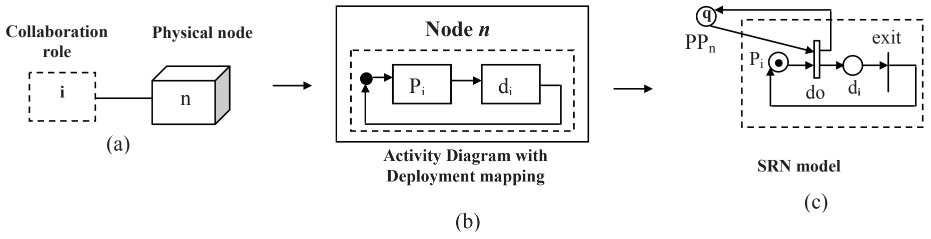


Figure 2.22 Model transformation rule 1

Rule 1: Deployment mapping of a collaboration role: *When the collaboration role of a building block deploys onto a physical node, the equivalent SRN model is represented by the 6-tuple in following manner:*

Φ $= \{P_i, d_i, PP_n\}$
T $= \{do, exit\}$
A $= \{\{(P_i \times do) \cup (do \times d_i)\}, \{(PP_n \times do) \cup (do \times PP_n)\}, \{(d_i \times exit) \cup (exit \times P_i)\}\}$
TT $= (do \rightarrow Timed, exit \rightarrow Immediate)$
Ml $= \{(P_i \times do) \rightarrow 1, (do \times d_i) \rightarrow 1, (PP_n \times do) \rightarrow 1, (do \times PP_n) \rightarrow 1 (d_i \times exit) \rightarrow 1, (exit \times P_i) \rightarrow 1\}$
$\boldsymbol{m_o} = \{(P_i \rightarrow 1\}, (d_i \rightarrow 0), (PP_n \rightarrow q)\}$

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.

Rule 2: Deployment mapping of single collaboration (Bidirectional): Rule 2 addresses the generation of an SRN model of a single collaboration, which is illustrated in Figure 2.23. 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 $\mathbf{t_{ij}}$ 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 $\mathbf{t_{ij}}$ in the SRN model is realized by both the overhead cost and communication cost.

Rule 2: Deployment mapping of single collaboration (Bidirectional): *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:*

Φ $= \{P_i, d_i, P_j, d_j\ PP_n\}$
T $= \{do_i, do_j, t_{ij}\}$
A $= \{(P_i \times do_i) \cup (do_i \times d_i), (PP_n \times do_i) \cup (do_i \times PP_n), (d_i \times t_{ij}) \cup (t_{ij} \times P_i), (P_j \times do_j) \cup (do_j \times d_j), (PP_n \times do_j) \cup (do_j \times PP_n), (d_j \times t_{ij}) \cup (t_{ij} \times P_j)\}$
TT $= \{(do_i, do_j, t_{ij}) \rightarrow Timed\}$
Ml $= \{((P_i \times do_i), (do_i \times d_i), (PP_n \times do_i), (do_i \times PP_n), (d_i \times t_{ij}), (t_{ij} \times P_i), (P_j \times do_j), (do_j \times d_j), (PP_n \times do_j), (do_j \times PP_n), (d_j \times t_{ij}), (t_{ij} \times P_j)) \rightarrow 1\}$
$\boldsymbol{m_o} = \{(P_i \rightarrow 1), (d_i \rightarrow 0), (P_j \rightarrow 1)\ (d_j \rightarrow 0), (PP_n \rightarrow q)\}$

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

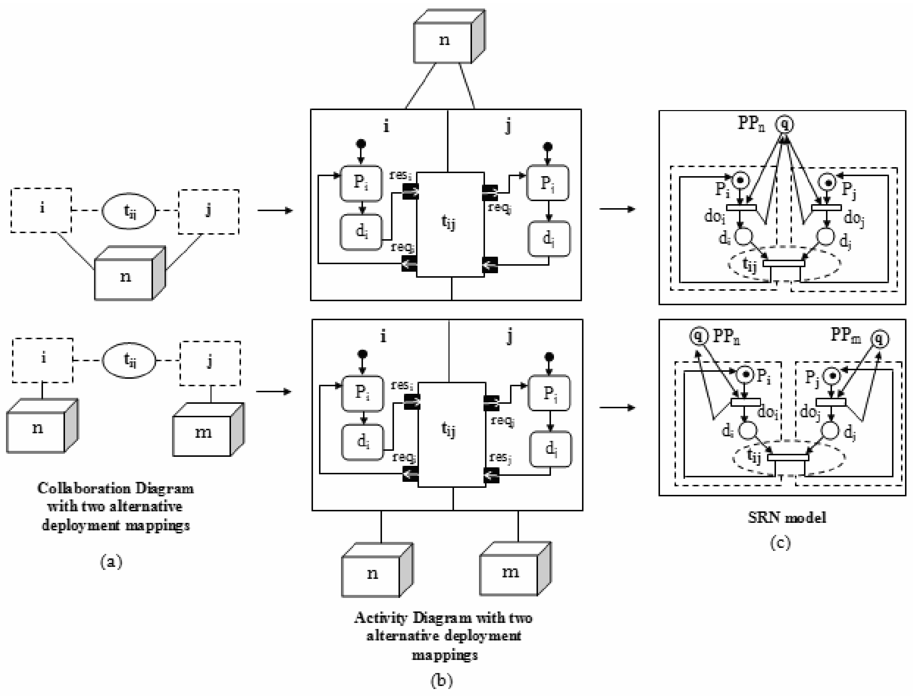


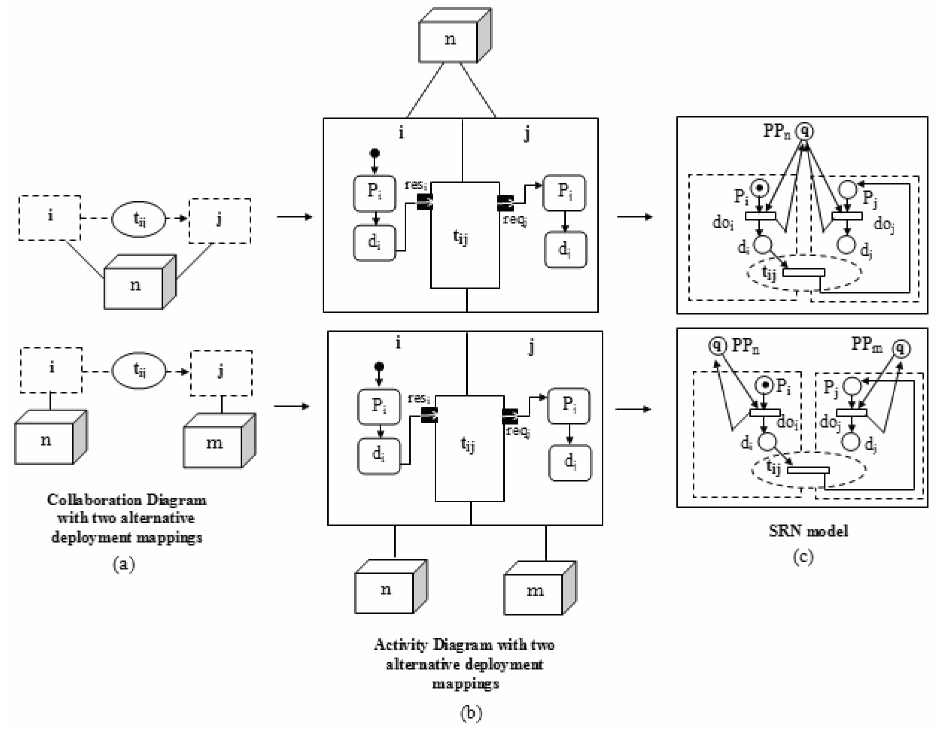
Figure 2.23 Model transformation rule 2

Rule 3: Deployment mapping of single collaboration (Unidirectional): Rule 3 addresses the generation of a SRN model of single collaboration, which is illustrated in Figure 2.24. Here, the collaboration connects only two collaboration roles in a unidirectional way, and the 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 $\mathbf{t_{ij}}$ 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 $\mathbf{t_{ij}}$ in the SRN model is realized by both the overhead cost and communication cost.

Rule 3: Deployment mapping of single collaboration (Unidirectional): *The SRN model of a collaboration, where the collaboration connects only two collaboration roles in a unidirectional manner and the roles are deployed on the same physical node can be represented by the 6-tuple in the following manner:*

$\boldsymbol{\Phi} = \{P_{i,}\ d_i,\ P_{j,}\ d_j\ PP_n\}$
$\boldsymbol{T} = \{do_i,\ do_j, t_{ij}\}$
$\boldsymbol{A} = \{(P_i \times do_i) \cup (do_i \times d_i),\ (PP_n \times do_i) \cup (do_i \times PP_n),\ (d_i \times t_{ij}) \cup (t_{ij} \times P_j),\ (P_j \times do_j) \cup$
$\quad (do_j \times d_j),\ (PP_n \times do_j) \cup (do_j \times PP_n)\}$
$\boldsymbol{TT} = \{(do_i,\ do_j,\ t_{ij}) \rightarrow Timed\}$
$\boldsymbol{Ml} = \{((P_i \times do_i),\ (do_i \times d_i),\ (PP_n \times do_i),\ (do_i \times PP_n),\ (d_i \times t_{ij}),\ (t_{ij} \times P_j),\ (P_j \times do_j),\ (do_j \times d_j),$
$\quad (PP_n \times do_j),\ (do_j \times PP_n)) \rightarrow 1\}$
$\boldsymbol{m_o} = \{(P_i \rightarrow 1),\ (d_i \rightarrow 0),\ (P_j \rightarrow 0)\ (d_j \rightarrow 0),\ (PP_n \rightarrow q)\}$

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



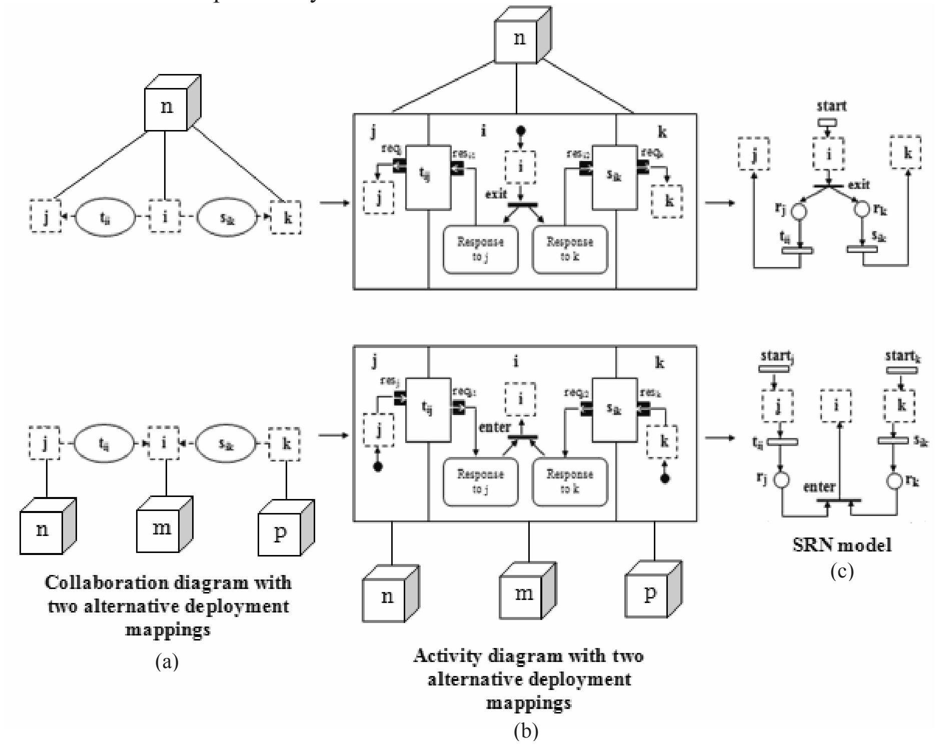
Collaboration Diagram with two alternative deployment mappings
(a)
Activity Diagram with two alternative deployment mappings
(b)
SRN model
(c)

Figure 2.24 Model transformation rule 3

Rule 4: Deployment mapping of multiple collaborations (Unidirectional): Rule 4 is described generally with deployment mapping of multiple collaborations. For a composite structure, if a collaboration role *i* connects with **n** collaboration roles by **n** collaborations like a star graph (where integer n > 1), where each collaboration connects only two collaboration roles, then the following is true:

- *Only one instance of collaboration role i exists during its state transition, and the single instance of collaboration role i connects with all other collaboration roles by timed transitions to generate the SRN model.*
- *The rates of transition* $\boldsymbol{t_{ij}}$ *and* $\boldsymbol{s_{ik}}$ *are determined in the same manner as rule 2, based on the deployment location of the collaboration roles.*

The SRN model of rule 4 is shown in Figure 2.25. In the first diagram (Figure 2.25), if component *i* contains its own token, an equivalent SRN model of the collaboration role *i* will be formed similar to rule 2. The same applies to the components *j* and *k* in the second diagram (Figure 2.25). The collaboration roles *i*, *j*, and *k* can be expanded as like rule 2 shown in Figure 2.23. Model transformation rule 4 can be presented by the 6-tuple in the same manner as the previously mentioned rules.



Collaboration diagram with two alternative deployment mappings
(a)
Activity diagram with two alternative deployment mappings
(b)
SRN model
(c)

Figure 2.25 Model transformation rule 4

Additional rules: Furthermore, we will present the transformation rules to generate SRN models for some of the components of the activity diagram that might have been presented in collaborative building blocks:

Decision node: The decision node in the UML activity diagram activates one of the outgoing flows, which are realized by the immediate or timed transition in the SRN model according to the performance annotation requirement. The activation of outgoing

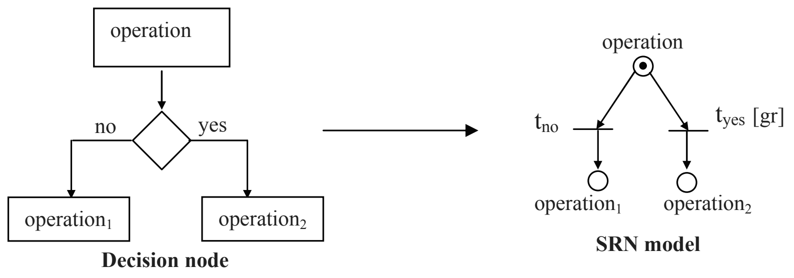


Figure 2.26 Model transformation of a decision node

flow in the UML activity diagram 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 to ensure that the decision is not made randomly. 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 Figure 2.26 is attached with a guard function **[gr]** in the equivalent SRN model, which will capture the same condition to be activated as the outgoing flow, which is 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.

Merge node: In the case of merge nodes, the outgoing flow is activated when either of the incoming flow arrives. This event is captured by using the immediate transition or timed transition in the SRN model according to the performance annotation requirement, which is shown in Figure 2.27.

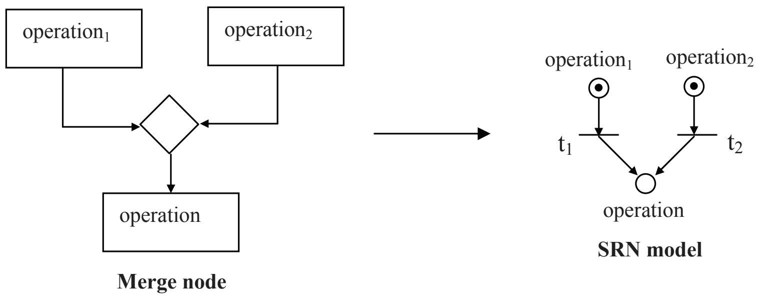


Figure 2.27 Model transformation of a merge node

Timer node: The timer node in the UML activity diagram is represented by the timed transition in the corresponding SRN model, which is illustrated in Figure 2.28.

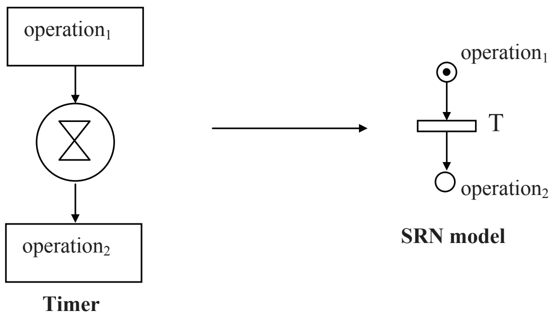


Figure 2.28 Model transformation of a timer node

Join node: In the case of join nodes, when all incoming flows arrive, the outgoing flow starts, which is realized by the immediate or timed transition in the SRN model according to the performance annotation requirement.

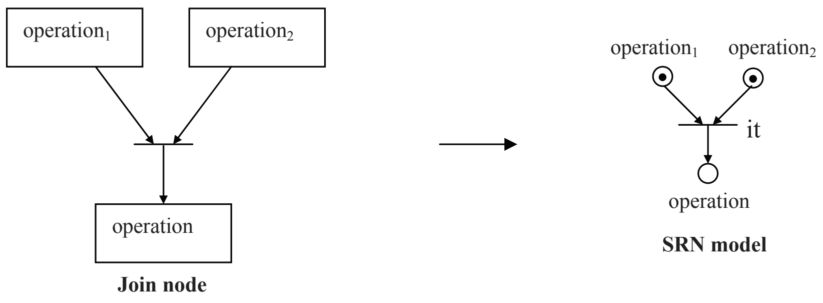


Figure 2.29 Model transformation of a join node

Fork node: The fork node is realized by the split of the incoming flow into several outgoing flows, which is represented by the immediate or timed transition in the SRN model according to the performance annotation requirement.

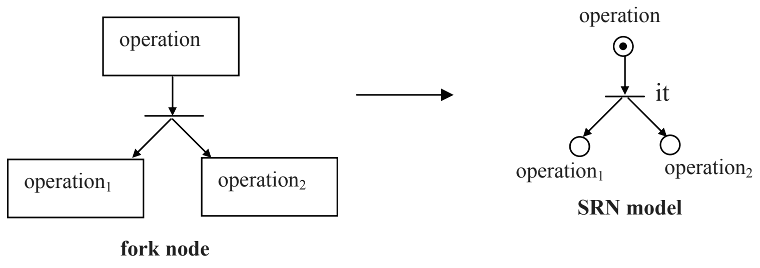


Figure 2.30 Model transformation of fork node

Case example: Taxi control system: generating performance SRN model using model transformation rules: *Here, we will illustrate how the model transformation rules have been utilized to generate a 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 Figure 2.31. Considering the deployment mapping of the taxi control system (Figure 2.19), UML models with annotations (Figure 2.12, 2.13), and the model transformation rules (Section 2.6.2 and Figure 2.22 – 2.30), 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 to model transformation rule 2 (Figure 2.23). The generated places and transitions 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 communication time and overhead time (see Figure 2.12), as the collaboration roles taxi and control centre*

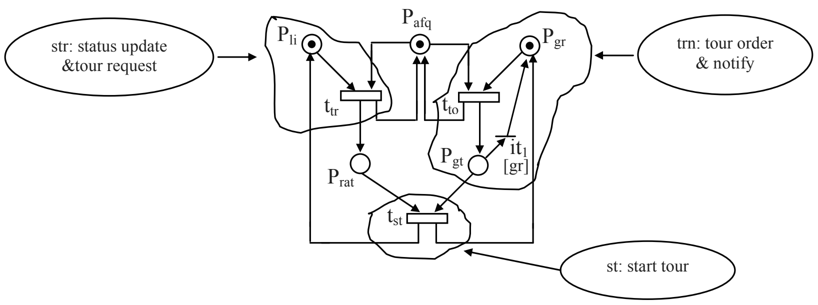


Figure 2.31 Performance SRN model of taxi control system

*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 (Figure 2.23). The generated places and transitions 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 both communication time and overhead time (see Figure 2.12), 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 (Figure 2.26). 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 in the Table 2.4, 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 (Figure 2.23), where the timed transition t_{st} is realized by both communication time and overhead time.*

Table 2.4 Guard function definition

function	definition
gr	If (#(P_{gt}) > n) 1 else 0

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 [23]) for various numbers of client and taxi combinations. The graph presented in Figure 2.32 shows the mean response time for various numbers of client and taxi combinations in the system by solving the SRN model (Here, the rate of each timed transition in the SRN model is considered 0.033). The top curve shows the mean response time for serving a client request when there is only 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 when there is only one taxi *available in the system. The response time for serving a client request is not only depends on the available taxis in the system but also number of client request 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 times 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 is 20. It is not logical that the control centre would be able to handle unlimited number of clients. Therefore, we consider a cutoff point of 20 clients for all of the curves. However, it is possible to solve the SRN model for a large number of client and taxi combinations.*

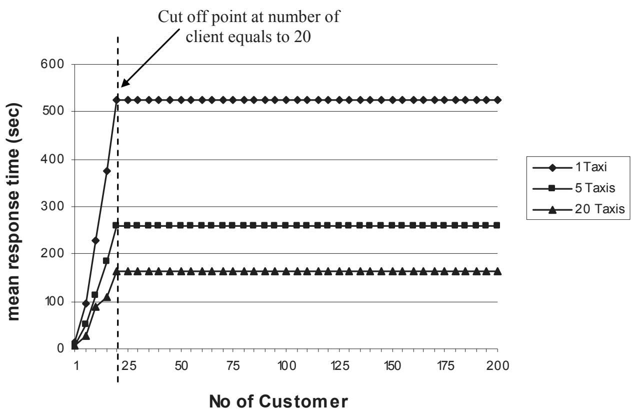


Figure 2.32 Numerical results of the analytical model (SRN) of the taxi control system

2.6.3 Dependability model generation

The system is defined by its logic, configuration data, and state space. The system receives input, i.e., a service request, and delivers a result, i.e., a service. A system fails when it does not perform the expected actions. A failure may be a physical defect, weakness or shortcoming of hardware components, or an error in the software components. Physical defects are the "classical" faults, i.e., wear of components, random device defects due to manufacturing imperfections, physical degradation of the material of the components as a result of electric overstress, mechanical stress, or shock. Physical faults are internal faults with respect to system boundaries, i.e., they are causes of the succeeding errors within the system. Physical faults are permanent; as soon as a physical fault occurs, the fault will remain until it is detected and repaired. Physical faults occur over the operational phase of the system's life cycle both during use and inactivity (in store, serving as a back-up or simply powered off). Design faults are another type of fault that affects the logic of the system. This class of faults spans from basic design flaws to trivial implementation failures in the circuitry and software of a system. A lack of proper timing and synchronization are other examples of design faults. In this context, design faults also encompass specification faults, e.g., inconsistent system specifications. Design faults are found in both hardware and software. Software faults are a subgroup of design faults embedded in the software. Some software faults cause failures that are almost impossible to reproduce. For instance, a slight change in the timing of the input values or in the overall system state may change the “course of proceedings” so much that a failure will not be reproduced. Some of the software faults will cause a failure if the conditions are roughly the same. These failures are known as reproducible failures. Software faults are sometimes difficult to localize. Thus, failure detection sometimes takes longer than recovery. In our work, UML STM is used to focus the abstract view of dependability behavior (failure and recovery) of the software and hardware components. STM can be translated into SRN model components by converting each state into a place and each state transition into a timed transition, which is reflected in the transformation Rule 5 and Rule 6. The rules for generating a dependability SRN model can be divided into two categories:

- Rule 5: software component
- Rule 6: hardware component

Rule 5: software component: Rule 5 addresses the generation of a STM model of software component dependability behavior, which is shown in Fig 2.33.

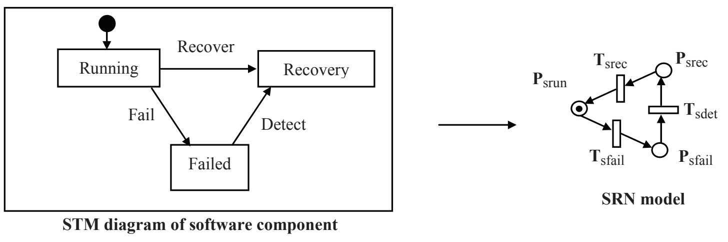


Figure 2.33 Model Transformation rule 5

Rule 5 (software component): *The SRN model of the STM of software component for addressing the dependability behavior can be presented by the 6-tuple in the following way:*

$\boldsymbol{\Phi} = \{P_{srun}, P_{srec}, P_{sfail}\}$
$\boldsymbol{T} = \{T_{sfail}, T_{sdet}, T_{srec}\}$
$\boldsymbol{A} = \{(P_{srun} \times T_{sfail}) \cup (T_{sfail} \times P_{sfail}), (P_{sfail} \times T_{sdet}) \cup (T_{sdet} \times P_{srec}), (P_{srec} \times T_{srec}) \cup (T_{srec} \times P_{srun})\}$
$\boldsymbol{TT} = \{(T_{sfail}, T_{sdet}, T_{srec}) \rightarrow Timed\}$
$\boldsymbol{Ml} = \{((P_{srun} \times T_{sfail}) \cup (T_{sfail} \times P_{sfail}), (P_{sfail} \times T_{sdet}) \cup (T_{sdet} \times P_{srec}), (P_{srec} \times T_{srec}) \cup (T_{srec} \times P_{srun})) \rightarrow 1\}$
$\boldsymbol{m_o} = \{(P_{srun} \rightarrow 1), (P_{srec} \rightarrow 0), (P_{sfail} \rightarrow 0)\}$

Rule 6: hardware component: Rule 6 addresses the generation of a STM model of hardware component dependability behavior, which is shown in Fig 2.34.

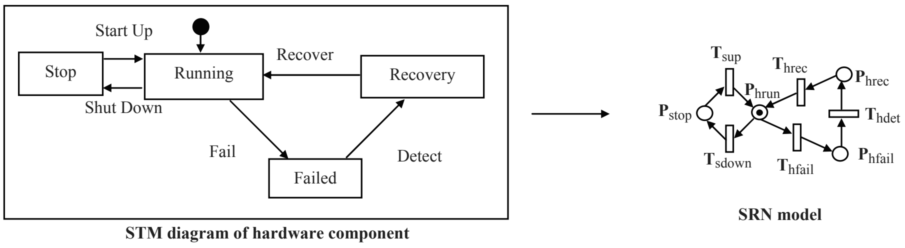


Figure 2.34 Model transformation rule 6

Rule 6 (Hardware component): *The SRN model of the STM of hardware component for addressing the dependability behavior can be presented by the 6-tuple in the following way:*

$\boldsymbol{\Phi} = \{P_{hrun,}\ P_{hrec},\ P_{hfail},\ P_{stop}\}$

$\boldsymbol{T} = \{T_{hfail}, T_{hdet}, T_{hrec}, T_{sup}\ , T_{sdown}\}$

$\boldsymbol{A} = \{(P_{hrun} \times T_{hfail}) \cup (T_{hfail} \times P_{hfail}), (P_{hail} \times T_{hdet}) \cup (T_{hdet} \times P_{hrec}), (P_{hrec} \times T_{hrec}) \cup (T_{hrec} \times P_{hrun}), (P_{hrun} \times T_{sdown}) \cup (T_{sdown} \times P_{stop}), (P_{stop} \times T_{sup}) \cup (T_{sup} \times P_{hrun}))\}$

$\boldsymbol{TT} = \{(T_{sfail}, T_{sdet}, T_{srec}, T_{up}\ , T_{down}) \rightarrow Timed\}$

$\boldsymbol{Ml} = \{((P_{hrun} \times T_{hfail}) \cup (T_{hfail} \times P_{hail}), (P_{hail} \times T_{hdet}) \cup (T_{hdet} \times P_{hrec}), (P_{hrec} \times T_{hrec}) \cup (T_{hrec} \times P_{hrun}),(P_{hrun} \times T_{sdown}) \cup (T_{sdown} \times P_{stop}), (P_{stop} \times T_{sup}) \cup (T_{sup} \times P_{hrun}))\rightarrow 1\}$

$\boldsymbol{m_o} = \{(P_{srun}\rightarrow 1), (P_{srec} \rightarrow 0), (P_{sfail} \rightarrow 0)\}$

2.6.4 Performability model generation

The execution of performance SRN is dependent on the execution of dependability SRN. A transition in the dependability SRN may induce a state change in the performance SRN. Moreover, state transitions in the dependability SRN model for the software process are connected to state transitions in the dependability SRN model for the associated hardware component where the software process deploy. These dependencies in the SRN models are handled using guard functions to generate a performability model [5] in two steps:

1. Synchronize the software and hardware dependability SRN models using guard functions
2. Synchronize the performance SRN model and dependability SRN model using guard functions

Synchronize the behavior of software and hardware dependability SRN models using guard functions: Referring to Section 2.6.3 (dependability model generation), two dependability SRN models are generated in our framework: A SRN model for the software component and a SRN model for the hardware component. State transitions in the SRN model of a software component are linked to the state transitions in the SRN model of the associated hardware component. A software component may stop working because of its own malfunction or failure. Again if the associated hardware component fails, the deployed software component on that hardware automatically stops working. To maintain the dependency between the state transitions in the software component with the associated hardware component, the SRN model of the software process is expanded by incorporating the following (Figure 2.35(c)):

- One additional place P_{hf}
- Three immediate transitions t_{hf}, t_{hfl}, t_{hfr}
- One timed transition T_{recv}

The expanded SRN model (Figure 2.35(c)) is associated with four additional arcs, including:

- $(P_{sfail} \times t_{hfl}) \cup (t_{hfl} \times P_{hf})$
- $(P_{srec} \times t_{hfr}) \cup (t_{hfr} \times P_{hf})$
- $(P_{srun} \times t_{hf}) \cup (t_{hf} \times P_{hf})$
- $(P_{hf} \times T_{recv}) \cup (T_{recv} \times P_{srun})$

The immediate transition, t_{hf} (Figure 2.35(c)), will be enabled only when a token is in place, P_{srun} (Figure 2.35(b)) and no token is in place, P_{hrun} (Figure 2.35(a)), this means that a failure of the hardware node will stop the operation of software process deployed on that node. The enabling of the immediate transitions t_{hfl} and t_{hfr} can be described in the

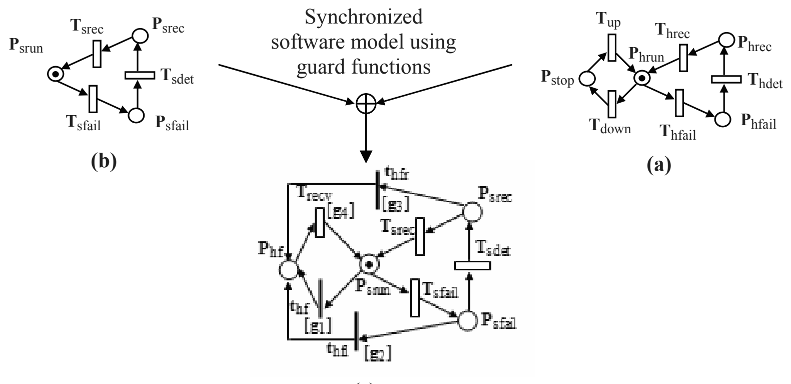


Figure 2.35 (a) Hardware SRN model (b) Software SRN model (c) Synchronized Software SRN model with hardware SRN model

same way. The timed transition, T_{recv}, will be enabled only when there is a token in both the places P_{hf} (Fig. 2.35(c)) and P_{hrun} (Fig. 2.35(a)). If there is a token in both the places P_{hf} and P_{hrun}, the software node will again begin to work after being recovered from a failure of the associated hardware component. T_{recv} is a timed transition, as it requires some time to start the operation of the software process after being recovered from the failure of the associated hardware node, where the software deploys. Four guard functions, $\boldsymbol{g_1}$, $\boldsymbol{g_2}$, $\boldsymbol{g_3}$, and $\boldsymbol{g_4}$ allow four additional transitions, t_{hf}, t_{hfl}, t_{hfr}, and T_{recv} of the software process to work consistently with the changes of states of the associated hardware node. The guard function definitions are given in Table 2.5.

Table 2.5 Guard functions definitions

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

Synchronize the performance SRN model and dependability SRN model using guard functions: The computer system receives an input, i.e., service request, and delivers a result, i.e., service. A computer system consists of several hardware and software components that can be used to complete a service request. The failure of either the software or hardware, which are responsible for achieving a certain service, eventually stops delivering service to the end user. To maintain the dependency between the processing of service requests and the running of software and hardware components, the performance SRN model is expanded by incorporating a guard function to produce performability model, which is demonstrated in Figure 2.36. For example, the performance SRN model (Figure 2.36(b)) is expanded by incorporating one additional

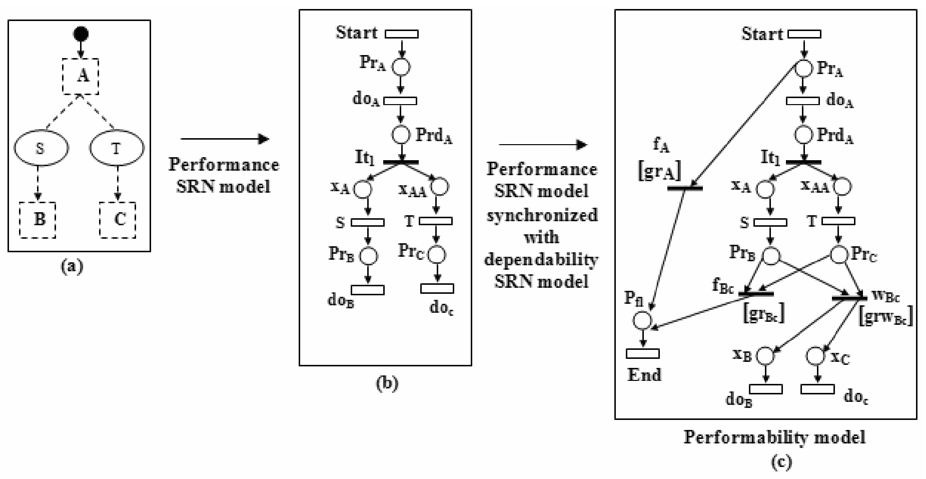


Figure 2.36 Synchronization of the performance SRN model with the dependability SRN model using guard functions

place, P_{fl}, and one immediate transition, f_A, shown in Figure 2.36(c). After being deployed when a service component A begins to execute, a check will be performed to verify whether both the software and hardware components (Figure 2.35(a), (c)) are running. If both the software and hardware components work, the timed transition, do_A, will fire, which represents the continuation of the processing of the service component A. However, if software respective hardware components (Figure 2.35(a), (c)) fail, the immediate transition, f_A, will be fired, which represents the cessation of the processing of service component A. The guard function, gr_A (defined in Table 2.6), allows the immediate transition, f_A, to work consistently with the change of states of the software and hardware components.

In our discussion, we consider that the failure of the execution of one service component because of the failure of software respective hardware eventually stops providing service to the users while maintaining the dependency of the parallel execution of service components with the running of software and hardware components. For example, service components B and C are executing in parallel (Figure 2.36(b)). To synchronize the processing of service components with the running of the software and hardware components, the performance SRN model of the parallel execution of service components B and C are expanded to produce performability model by incorporating one additional place, P_{fl}, and two immediate transitions, f_{BC} and w_{BC}, shown in Figure 2.36(c). If software respective hardware components (Figure 2.35(a), (c)) fail, the immediate transition, f_{BC}, will be fired, which symbolizes the cessation of the execution of both service components B and C instead of stopping the execution of either component B or C, which eventually postpones the execution of services. Postponing the processing of either service component B or C will result in the inconsistent execution of the SRN model and produce erroneous results. If both the software and hardware components work as intended (Figure 2.35(a), (c)), the timed transition w_{BC} will be fired, which ensures the continuation of the execution of parallel processes B and C. The guard functions gr_{BC} and grw_{BC} allow the immediate transitions f_{BC} and w_{BC} to work consistently with the change of the states of the software and hardware components. The guard function definitions are given in Table 2.6. The examples regarding the generation of the performability model are illustrated in Paper 7 and Paper 8. The algorithms for the transformation of UML models to SRN models utilizing the above stated model transformation rules have been described in Paper 8 and Paper 9.

Table 2.6 Guard functions definitions

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

The above Sections (2.1 - 2.6) mainly describe the steps of our performance and performability framework. The below Sections (2.7 - 2.9) will describe other aspects of our modeling framework.

2.7 Formalizing the UML specification style using cTLA

The UML specification style introduced in this thesis 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 formally present the modeling specifications 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 cTLA style, which is defined as cTLA/c and the semantics of state machine diagram realized by the cTLA formula [13]. 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 collaborations, activities, deployment, and the state machine diagram.

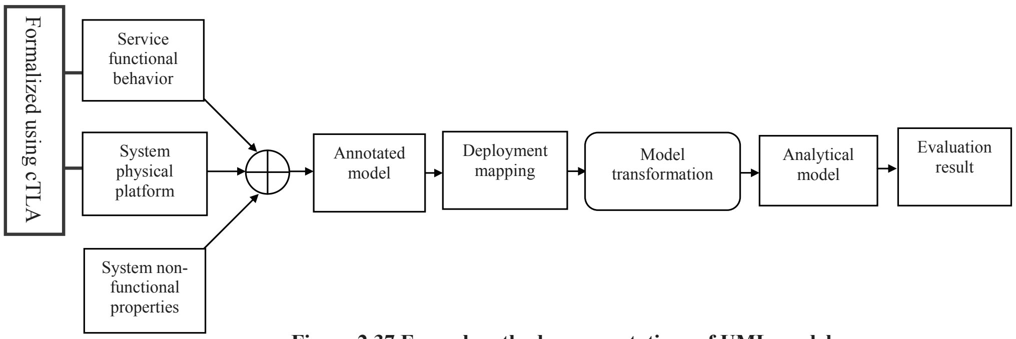


Figure 2.37 Formal method representations of UML models

2.7.1 cTLA/c for collaborative service specification

cTLA/c is a formal basis for defining the collaborative service specification using UML collaboration and activity [13]. 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 Paper 9, 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. A 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 [1]. 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 because 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 [13]. We describe some activity element types as separate cTLA processes in Paper 9, which help to understand the semantics of the activity elements. Moreover, the production rules of cTLA actions for UML activities have been presented in Paper 9 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, which is also described in Paper 9.

2.7.2 cTLA to specify UML state machine diagrams

Our modeling framework used UML state machine diagrams to capture failure and recovery events of the system components. TLA is a linear-time temporal logic that models 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 . .] [14] [15]. Thus, the TLA formalism can define the state machine formally produced by our framework, which ultimately also models considerably long sequences of states, s_i, starting with an initial state, s_0. cTLA originated from TLA to offer more easily comprehensible formalisms and propose a more supple composition of specifications [8]. A 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 with the help of variables, actions, and events. The detailed formalism is defined in Paper 8.

In addition, the formalization of a UML model using a cTLA process thus helps to describe the mapping process to show the correspondence with the analytical model. UML activities are based on Petri Nets and as such describe a state transition system. As an analytical model, our framework produces a SPN and SRN (extension of Petri Nets). The cTLA can define the state transitions formally produced by our framework, which ultimately also model considerably long sequences of states, s_i, starting with an initial state, s_0. Thus, we consider a mapping approach that directly shows the mapping of the cTLA-specified UML model into a SRN model. This correspondence in turn ensures that the mapping of the cTLA-specified UML model and Petri nets fit together and the UML correctly transforms to the analytical model. The detailed mapping process demonstrating the correspondence between the cTLA-specified UML and SRN is described in Paper 9.

2.8 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. The description of the tool-based support of our modeling framework is given in Papers 8 and 9. The tool support of our modeling framework is illustrated in Figure 2.38. We have used two tools:

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

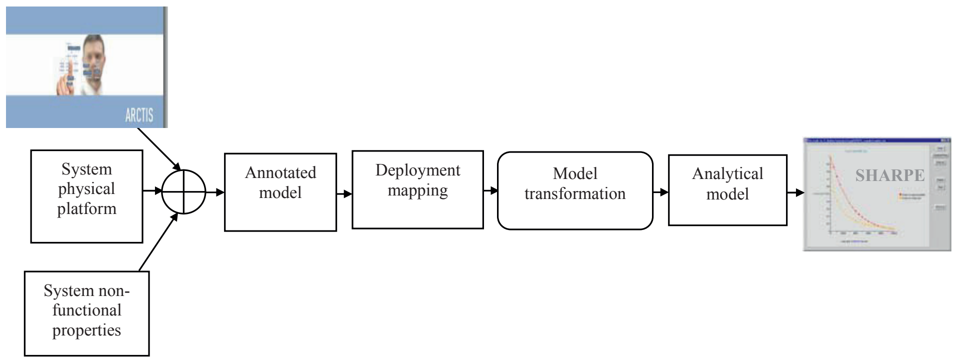


Figure 2.38 Tool support of our modeling framework

The two tools are integrated in our modeling framework to evaluate the performance and performability of the distributed system. The above 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 and performability evaluation of that service. This functionality ultimately helps to accomplish our research objective.

2.8.1 Arctis

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 [10]. 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.

Model verification using Arctis: Model verification is an integral part while deriving the tool-based support of the developed framework. This verification is also necessary to precisely and correctly represent the model specifications. Because temporal logic is applied to define the UML model specification, we can use the model checker TLC to verify the specification [17]. The external model checker TLC is invoked by Arctis from the command line, invisible to the user. To analyze building blocks and complete systems, the Arctis editor constantly checks the model for a number of syntactic constraints. For a more thorough analysis of the behavior, Arctis employs the model checker TLC based on the Temporal Logic of Actions (TLA). Figure 2.39 outlines this process: 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. 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

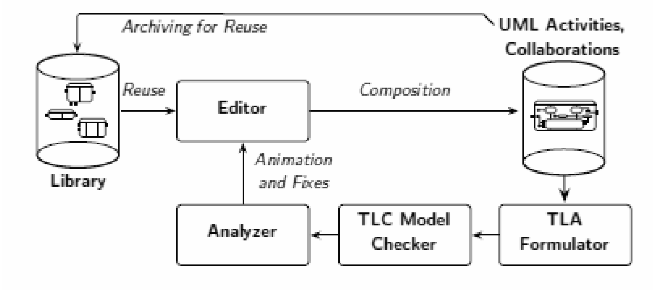


Figure 2.39 Model checking with TLC [18]

maintained by activities in general. Examples of these properties include the correct use of building blocks within the activity such that the activity itself satisfies a certain externally visible behavior; each call behavior action representing an instance of a building block, where a call behavior action declares a corresponding pin; each building block that has one activity partition for each collaboration role; state sequences that are connected through appropriate pins during the composition of building block activity, etc. When TLC detects that a theorem is violated, it produces an error trace displaying the state sequence that leads to the violation. If the model specification does not violate any theorems, the verification ends successfully. Otherwise, an error will be reported by the TLC in textual format, and a number of diagnoses will be considered based on the error trace. The detailed model verification process has been defined in [18].

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, UML STM model, and incorporate non-functional parameters to annotate UML model for Arctis are under development.

2.8.2 SHARPE

SHARPE (Symbolic Hierarchical Automated Reliability/Performance Evaluator) is a tool that accepts specifications of mathematical models and requests for model analysis [11]. 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 SPN or SRN model as follows:

```
spn/srn  name {(param list)}
* section 1: places and initial numbers of tokens
<place name expression>
end
* section 2: timed transition names, types, and rates
{
<transition_name ind expression {guard expression} {priority expression}>
<transition_name placedep place_name expression {guard expression}{priority expression}>
<transition_name gendep expression {guard expression} {priority expression}>
}
end
*  section 3: immediate transition names, types, and rates
{
<transition_name ind expression {guard expression} { priority expression}>
<transition_name placedep place_name expression {guard expression}{priority expression}
<transition_name gendep expression {guard expression} {priority expression}>
}
end

* section 4: place-to-transition arcs and multiplicity
{ <place_name transition_name expression> }
end
* section 5: transition-to-place arcs and multiplicity
{<transition_name place_name expression>}
end
*section 6: inhibitor arcs and multiplicity
{<place_name transition_name expression>}
end
* section 7: deriving results
{
<built in function >
<user defined function>
}
end
where param list is:
name, name, ..., name
```

name, *trans name,* and *place name* are all symbols; *expression* is a mathematical expression that could contain function calls; *ind* means that the transition's firing rate is not dependent on the current marking of the net; *placedep* means that the transition's firing rate depends on the number of tokens in the specific place mentioned and the expression assigned to it; and *gendep* means that the firing rate depends on the marking-dependent function referenced in the corresponding *expression*. Each line in the first section specifies a place name and the initial number of tokens in the place.

Each line in the 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. If the transition is marking-dependent, the effective rate of the transition depends on (is multiplied by) the number of tokens present in the place.

Each line in the 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. If the transition is dependent, the effective weight of the transition depends on (is multiplied by) the number of tokens present in the place. The transition weight determines the probability that the transition is chosen if it is one of multiple immediate transitions leaving a place.

The lines in the fourth section specify 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.

The lines in section five specify the arcs from transitions to places. The multiplicity indicates the number of tokens that are deposited in the place when the transition is fired.

The lines in section six specify inhibitor arcs from places to transitions. The multiplicity indicates how many tokens must be in place to inhibit the transition from firing.

The lines in section seven specify the built in functions of SHARPE or user-defined functions that can be used to derive result.

Some of the built-in functions for model evaluation are given below:

- tput (system name, transition name): throughput for a GSPN or SRN transition in steady state
- util (system name, transition name): utilization of a GSPN or SRN transition in steady state
- etok (system name, place name): average number of token in the place in steady state
- prempty (system name, place name): probability that the place is empty in steady state

Although the Arctis and SHARPE tools have been utilized to describe service definitions using a UML collaboration and activity as well as model evaluations, the following steps of our modeling framework have been implemented as parts of the PhD thesis work (focused on Paper 8 and Paper 9):

- System physical platform using UML deployment diagram
- Describing system components dependability behavior using UML STM
- Deployment mapping
- Generation of annotated UML model
- Automated Model transformation
- Model validation for XML

The steps of the performance and performability modeling framework, such as UML deployment diagram and deployment mapping (Section 2.2 and 2.5), UML STM diagram generation (Section 2.2), and performance and dependability parameters incorporation into UML models (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. This validation ensures that the XML document is syntactically correct. Hence, erroneous data or typos in the XML document will be fixed during the XML validation and inform the user to correct the corresponding data. To ensure this claim, we have defined several constraints and checks in the schema files. The detailed description of XML validation is given in Paper 9.

2.9 Scalability, generalization and extensibility aspects

The contributions of this thesis work were presented in detail in the previous sections. This section highlights some of the important factors concerning our modeling approach.

2.9.1 Scalability

The examples we considered as case studies and presented in the papers were chosen to cover real scenarios and also included the well-known problem of assigning clusters to nodes as artificial case studies. The examples were compact enough in some respects to be completely presented within the space constraints of an 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 papers, 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 efficient deployment mapping. For the model transformation, we have described generalized rules and algorithms (see Papers 8, 9, and Section 2.6) to also handle complexity and scalability. However, tackling state explosions for model evaluations of large Petri nets using SHARPE is challenging. Below, we explain an aggregate method (see Paper 9 for more description) to easily solve and evaluate large problems by tackling the challenge of state explosion. To describe the aggregate method, we will revisit the taxi control system example mentioned in Section 2.1. However, the UML collaboration diagram of the taxi

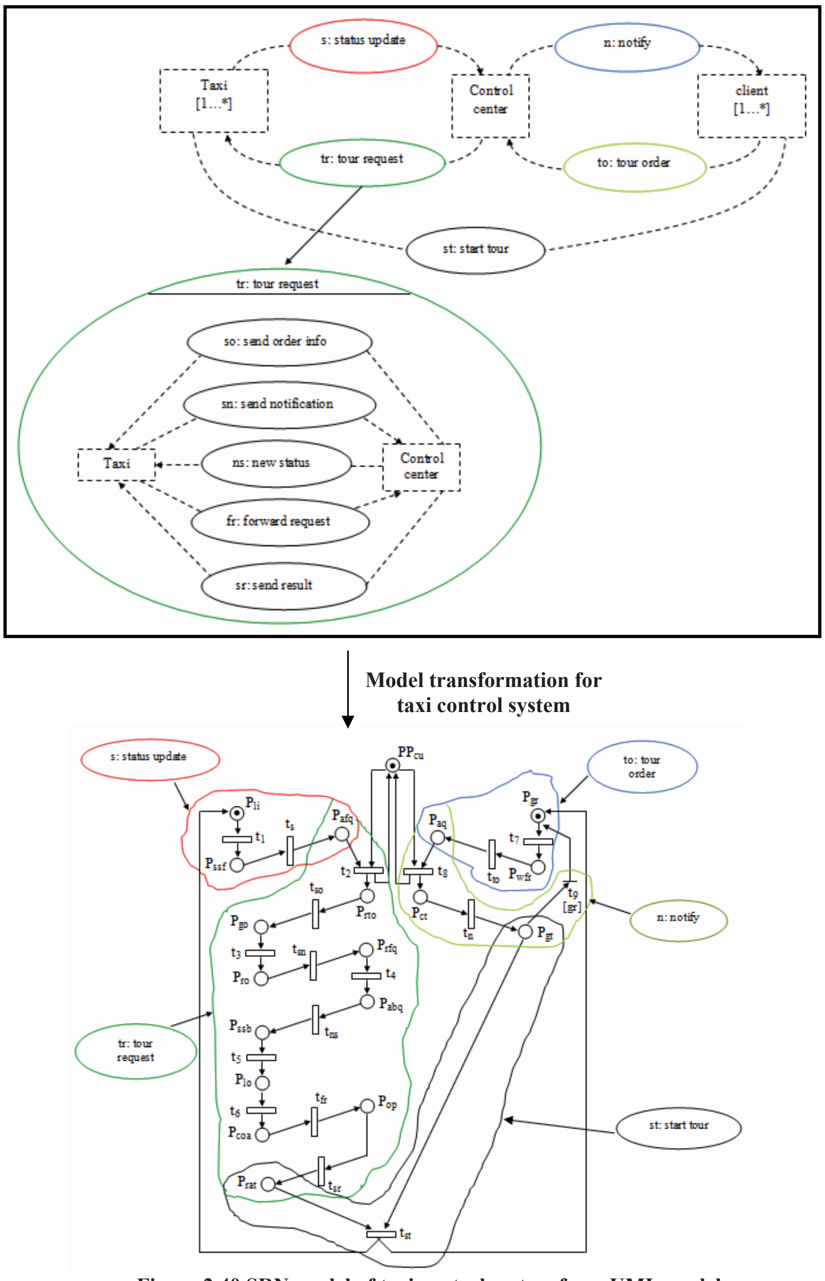


Figure 2.40 SRN model of taxi control system from UML model

control system is described more broadly here, which is shown in Figure 2.40. The participants in the service are represented by the collaboration roles *taxi*, *control center*, and *client*. The *control center* has a default multiplicity of one; while many taxis and customers can be in the system, which are denoted by the multiplicity [1,…,*]. Between the roles, collaborations denote the occurrence of a behavior: *taxi* and *control center* are interacting with the collaborations *status update* and *tour request* (where *tour request* is a composite collaboration), the *control center* cooperates with the *client* by means of the collaborations *notify* and *tour order,* while the interaction between *taxi* and *client* is

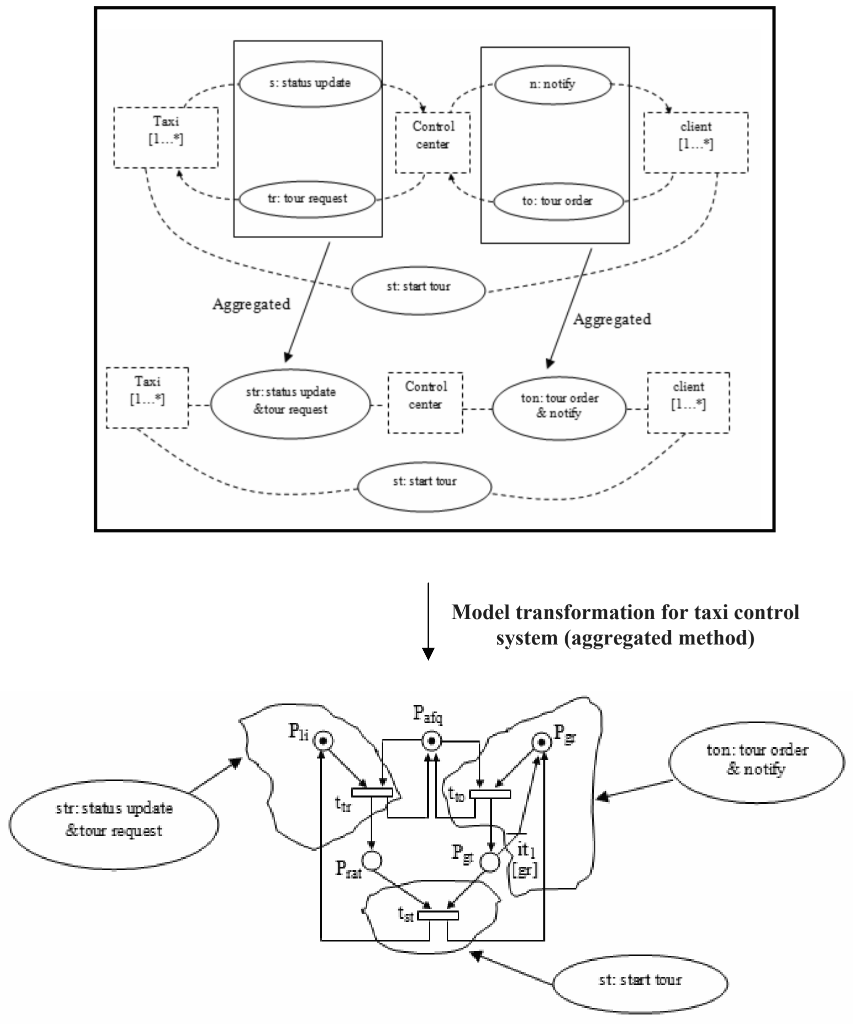


Figure 2.41 SRN model of Taxi control system from UML model (aggregated version)

realized by the collaboration *start tour*. Considering the collaboration diagram (Figure 2.40), the equivalent SRN model is shown in Figure 2.40, which is generated using model transformation rule 1 (Figure 2.22), rule 3 (Figure 2.24), and the transformation rule for a decision node (Figure 2.26). Here, we skip other steps of the model transformation process, as they were already described in Section 2.1 to Section 2.6.

The taxi control system usually handles a large number of client requests and taxis. Considering this factor, the SRN model for a taxi control system mentioned in Figure 2.40 is very demanding with respect to the execution time and also quickly suffers from scalability and state explosion problem as the numbers of client requests and arrival of taxis increase in the system. To tackle these problems, we have redesigned the UML model to generate an aggregated version of the above SRN model (Figure 2.40) so that the model can easily be scaled for a large number of client requests and taxis. The aggregated version of the UML collaboration diagram and the equivalent SRN model for the taxi control system are shown in Figure 2.41, which was already described in Section 2.1 to Section 2.6. Between the collaboration roles, collaborations denote the occurrence of a behavior: *taxi* and *control center* interact with the collaboration *status update & tour request* (collaborations *status update* and *tour request* in Figure 2.40 are presented aggregately in Figure 2.41); the *control center* cooperates with the *client* by means of the collaboration *tour order & notify* (collaborations *tour order* and *notify* in Figure 2.40 are presented aggregately in Figure 2.41)*,* while the interaction between *taxi* and *client* is realized by the collaboration *start tour*. Considering the aggregated collaboration diagram (Figure 2.41), the equivalent SRN model is shown in Figure 2.41, which is generated using model transformation rule 1 (Figure 2.22), rule 2 (Figure 2.23), and the transformation rule for decision nodes (Figure 2.26).

We use SHARPE [11] to execute the obtained SRN models for the taxi control system and calculate the mean response time for various numbers of client requests and taxis. The large SRN model in Figure 2.40 and the aggregated version of the large SRN model in Figure 2.41 produce similar results. However, the large SRN model (Figure 2.40) can only be solved for very small numbers of client requests and taxis, whereas the aggregated version (Figure 2.41) can be solved for large numbers of client requests and taxis. Paper 9 focuses more on the scalability issue.

2.9.2 Generalization and extensibility

The modeling approaches presented in this work are general enough to illustrate a wide variety of application scenarios (presented in the papers of part II) to show the applicability of our developed framework. 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 model transformation rules are defined to ensure the generality and completeness such that the model can be transformed for various application domains of distributed systems. The model transformation also supports the transformation of arbitrary UML models, such as combinations of collaborations, activity, deployment, and state machine diagrams. Moreover, the model transformation algorithms (see Paper 8 and Paper 9) are designed based on the generalized model transformation rules to automate the model transformation. Moreover, our framework is supported by the tool suites Arctis and SHARPE, which can be generally used for various model types. The Arctis wizard can define UML collaborations and activity diagrams. The development of a missing plug-ins for Arctis to define UML deployment diagram and UML STM diagram are ongoing. Arctis will then be used to support various UML models as inputs for the model transformation process. Although SHARPE is used to evaluate Markov and Petri nets in our modeling framework, it can easily solve various analytical models [11].

The extensibility of our modeling approach can be defined in several directions, such as considering the internal behavior of collaborative building blocks in run time by exchanging existing service components or making new components available via some discovery mechanisms, 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, compiling a complete profile that will help annotate our UML models in accordance with the performance and performability evaluation, considering a method that could be applied to automatically and realistically derive the costs, generating other output models using our framework that can be solved by the SHARPE to evaluate the performance and performability result, 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.

CHAPTER 3

Summary of the papers

The main contributions of this thesis have been published in international peer reviewed conference proceedings and journals. The papers in Part II of this thesis are organized in a chronological order that reflect the gradual development of this research work. The papers are included in this thesis as originally published (except Paper 9, which has been submitted to a journal for reviewing), but the formatting has been modified according to the style of this thesis report. However, some variations in the notation used in the papers might present because of the gradual development of the methods of this research work.

Generally, the papers have been categorized into four areas:

A – Developing a performance modeling framework, which considers reusable software components to generate performance models from the system design specification

B – Considering execution cost, overhead cost, and communication cost while deriving cost functions to assess deployment mapping and consider optimized model transformation rules while generating performance models

C – Developing a performability modeling framework, which considers reusable software components to capture the system functional behavior, cost functions for deployment mapping, and optimized model transformation rules

D – Tool support for the performance and performability modeling framework, including model validation

The order and relationship among the papers are shown in Figure 3.1.

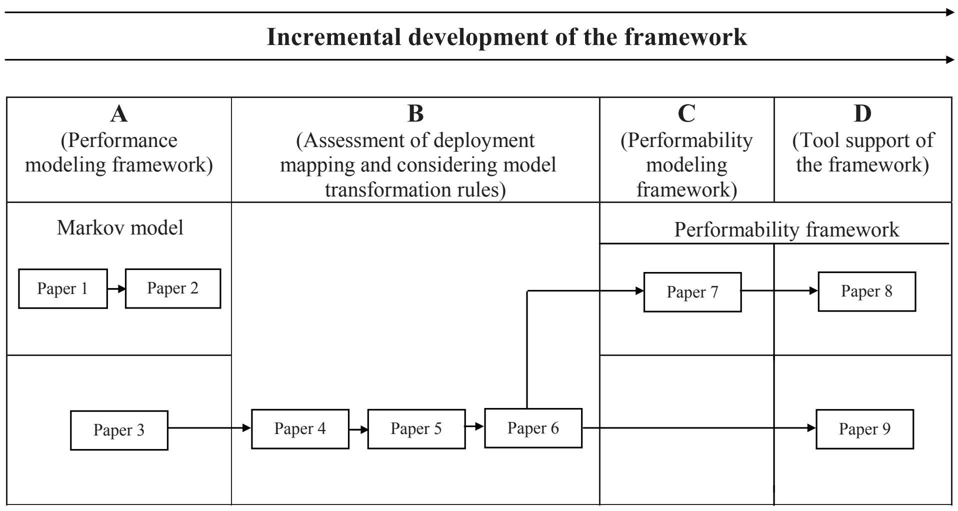

Incremental development of the framework
A
(Performance modeling framework)
B
(Assessment of deployment mapping and considering model transformation rules)
C
(Performability modeling framework)
D
(Tool support of the framework)
Markov model
Performability framework
Paper 1
Paper 2
Paper 3
Paper 4
Paper 5
Paper 6
Paper 7
Paper 8
Paper 9

Figure 3.1 Relationship and order of the included publications

The focus of Papers 1 – 3 is to develop a performance modeling framework that utilizes reusable software components to capture the dynamics of distributed systems. Another important aspect is to investigate the deviation in the system performance because of considering alternative system execution architectures (category A). Papers 4 - 6 develop cost functions to consider deployment mapping while satisfying the non-functional properties of the system. Optimized model transformation rules to automatically and scalably transform the models are also emphasized (category B). These six papers present the research conducted together with the only advisor of the author, Poul E. Heegaard. Paper 7 focuses on the development of the performability modeling framework while utilizing reusable software components to capture the dynamics of distributed systems (category C). Fumio Machida and Kishor S Trivedi also contribute to this research conducted by the author. Papers 8 and 9 delineate the tool support of the performability and performance modeling frameworks and formalize the UML specifications. Defining the model validation to correctly transform the model is another important focus of these papers (category D). Although we considered generating Markov models from the UML specification style in Papers 1 and 2, we ultimately aimed to generate Petri nets. Specifically, we intended to develop SRN models from the UML specification style because the semantics of Petri nets are similar to those of the UML activity diagram and state machine diagrams, which are used as the main specification units of our modeling framework. Moreover, the SRN provides several tremendous features to accurately model performance- and performability-related behavior [22]. The papers presented in part II mainly describe the automated generation of Petri net models from the UML specification style, except for Paper 1 and Paper 2. A brief summary of the included papers is given below:

Paper 1

Translation from UML to Markov model: A performance modeling framework

This paper focuses on the early assessment of the distributed system performance. This assessment is achieved by developing a framework that proposed a translation process from a high-level UML notation to a CTMC model and solved the model for related performance metrics. UML models capture the functional behavior of the distributed system, and the system is quantitatively evaluated by solving the CTMC. The framework utilizes several UML models, such as collaboration, activity, and the deployment diagram. The system dynamics are captured by the collaborative building block, where the UML collaboration and activity diagram are complementarily used to each other. The collaboration- and activity-oriented approach provides an opportunity to reuse the collaborative building block that specifies the functional behavior of the system where a collaboration diagram is used to define the structure of the building blocks. An activity diagram is applied to delineate the internal behavior of the collaboration and overall behavior of the system. The architecture of the system execution and deployment mapping of the service components over the execution environment are outlined with the help of a UML deployment diagram. This deployment mapping shows how the service is delivered by utilizing the joint behavior of the system components, which may be physically distributed. UML models are annotated to incorporate non-functional parameters required during the performance evaluation of the system according to the *UML profile for Schedulability, Performance, and Time*. The translation process from a UML to a CTMC model is specified with the help of a state-marking approach. The SHARPE tool is used to evaluate the performance by solving the CTMC model.

Unique contributions of Paper 1:

- The framework that transforms the UML specification style into CTMC to evaluate the performance of a distributed system is described in a stepwise manner.
- The system dynamics are captured by collaborative building blocks, where the UML collaboration diagram is used to define the structure of the building block. An activity diagram is applied to delineate the internal behavior of the collaboration and overall behavior of the system.
- The physical platform of the system is demonstrated using the UML deployment diagram.
- The deployment mapping of the service components over the execution environment shows the delivery method of the service by the joint behavior of the system components, which may be physically distributed.

- The UML models are annotated to incorporate non-functional parameters according to the *UML profile for Schedulability, Performance, and Time.*

- The deployment mapping with annotation is used to transform the model into a CTMC model using a state-marking approach.

- A real case study is considered to show the applicability of the framework.

Paper 2

Translation from UML to Markov model: A performance modeling framework for managing behavior of multiple collaborative sessions and instances

This paper describes the performance modeling framework that presents the UML specification style to capture the functional behavior of a system while utilizing reusable software components to explicitly coordinate multiple collaborative sessions that occur simultaneously. Another contribution of this paper is to consider design alternatives of the architectures of system execution to stipulate the deployment mapping of software artifacts on the physical nodes. The UML models are annotated to incorporate non-functional parameters according to the *UML profile for Schedulability, Performance, and Time*. The design alternatives of the execution environment with annotation are considered while conducting the model transformation to generate a CTMC model from the UML model. The CTMC models are subsequently evaluated to show the performance effects of the distributed systems due to the consideration of different system execution architectures for identical system functional behavior. A new and complex case study is introduced to show the applicability of the performance modeling framework.

Unique contributions of Paper 2:

- The system dynamics are captured by the collaborative building blocks, which focus on the coordination among multiple collaborative sessions that occur simultaneously (Paper 1 included the standard description).

- The paper considers design alternatives of the system execution architecture using UML deployment diagram.

- The deployment mapping is conducted for design alternatives of the system execution architectures to show how the service of multiple collaborative sessions is delivered by the joint behavior of the system components, which may be physically distributed.

- The deployment mapping for the service of multiple collaborative sessions with model annotation is utilized to transform the model into a CTMC model using a state-marking approach.

- A real case study is considered to demonstrate the applicability of a modeling framework that focuses on the performance effects of the distributed systems due to the consideration of different system execution architectures for identical system functional behaviors.

Paper 3

Translation from UML to SPN model: A performance modeling framework for managing behavior of multiple collaborative sessions and instances

This paper focuses on the performance evaluation of the distributed system at the early stage of the system development process that generates a SPN model from the system design specifications. This assessment is achieved by the developed framework is tailored to propose a translation process from a high-level UML notation to a SPN and solves the model for related performance metrics. This paper emphasizes the multiple collaborating instances that occur simultaneously while capturing system functional behavior and also the design alternatives of the system execution architecture to illustrate deployment mapping of the system. UML models are annotated to incorporate non-functional parameters required during performance evaluation of the system according to the *UML profile for Schedulability, Performance, and Time*. In addition, the model transformation process is described to generate a SPN model from the UML specification style for multiple collaborating instances and design alternatives of system execution architectures. The applicability of the performance modeling framework is illustrated with the help of a new real case scenario.

Unique contributions of Paper 3:

- Stepwise description of the framework that transforms the UML specification style into a SPN model to evaluate the performance of a distributed system performance (This paper introduces a different analytical model compared with Paper 1 and Paper 2)

- The system dynamics are captured by the collaborative building blocks, which focused on the coordination among multiple collaborative sessions that occur simultaneously (Paper 1 included the standard description).

- The paper considers design alternatives of the system execution architecture using UML deployment diagram.

- The deployment of design alternatives of the system execution architectures are mapped to show how the service of multiple collaborative sessions is delivered by the joint behavior of the system components, which may be physically distributed.

- A detailed model transformation approach is described that utilizes the deployment mapping for the service of multiple collaborative sessions with model annotations to transform into SPN model

- A real case study is considered to show the applicability of the framework that focused on the performance effects of the distributed systems due to the consideration of different system execution architectures for identical system functional behaviors.

Paper 4

A performance modeling framework incorporating cost efficient deployment of collaborating components

This paper continues the development of our modeling framework to provide further general explanations that aims to generate a SPN model from the system design specifications. System design specifications are modeled by the reusable software components as collaborative building blocks. The specification is given in such a general way that can be easily tailored to any specific case scenario of distributed systems. To assess the deployment mapping of software components to the available physical resources to satisfy non-functional requirements, cost functions are introduced that express the utility of the deployment mapping of a distributed service. In addition, model transformation rules are presented, which ensure the scalability and automation in model transformations to generate a SPN model from UML specifications. A general formula is derived to produce performance results by solving the SPN model. A traditional and well-established task assignment problem is acclimatized to the service-engineering context in this paper. The original problem is tailored to show the applicability of our performance modeling framework.

Unique contributions of Paper 4:

- The illustration of the service functional behavior using UML specification is given in such a general way that can be easily tailored to any specific case scenario of distributed systems (The illustrations are more generalized than in previous papers).

- Cost functions that express the utility of the deployment mapping of a distributed service are introduced to assess the deployment mapping of software components to the available physical resources that satisfy non-functional requirements (Here, deployment decisions were made differently than in previous papers).

- Very generalized model transformation rules are presented, which ensure the scalability and automation of the model transformation to generate a SPN model from UML specifications.

- A formula is derived to produce performance results by solving the SPN model.

- A traditional and well-established task assignment problem is tailored to show the applicability of our performance modeling framework (A different case study is considered than in previous papers)

Paper 5

A performance modeling framework incorporating cost efficient deployment of multiple collaborating instances

This paper introduces several contributions that carry on the incremental development of the framework while describing our performance modeling framework. The UML specification style for multiple collaborating instances is initiated in a very general and broad way to easily illustrate systems for different application domains using this specification style. Furthermore, the UML models are annotated to incorporate non-functional parameters according to the *UML profile for MARTE: Modeling and Analysis of real-time embedded systems* for performing quantitative evaluation. This new profile is intended to replace the existing *UML Profile for Schedulability, Performance, and Time* to provide a common way to model both the hardware and software aspects of real-time embedded systems. This paper also optimizes the model transformation rules so that models can be easily transformed for a wide range of application domains in a scalable and automated way. As a performance model, the SRN (extension of the SPN model) model is generated to take advantage of several prominent and important properties, such as the marking dependent arc multiplicity that can change the structure of the net, marking dependent firing rate, and reward rate defined at the net level. Another important focus of this paper is to describe methods for the parallel processing of a networked node while utilizing the limited processing power of that node.

Unique contributions of Paper 5:

- The stepwise description of a framework that transforms the UML specification style for multiple collaborating instances into a SRN model (extension of SPN model) for distributed system performance evaluation (This paper introduces a different analytical model compared with the previous four papers)

- UML models are annotated to incorporate non-functional parameters according to the *UML profile for MARTE: Modeling and Analysis of real-time embedded systems* for quantitative evaluation (Different UML profiles are considered than in previous papers)

- A new parameter is introduced in the cost function to assess the specific deployment mapping of service components on the physical infrastructure.

- A generalized set of model transformation rules to generate the SRN model from the UML specification style is illustrated.

- As a performance model, the SRN (extension of the SPN model) model is generated to take advantage of several prominent and important properties, such as the marking dependent arc multiplicity, marking dependent firing rate, and reward rate defined at the net level

- The paper also focuses on addressing the parallel processing of a physical node while utilizing its limited processing power when generating a SRN model.

Paper 6

Derivation of Stochastic Reward Net (SRN) from UML specifications considering cost-efficient deployment management of collaborating service components

This paper introduces a new idea that focuses on another aspect of our performance modeling framework. In this paper, a service-engineering approach is specified with respect to a unidirectional graph while capturing the system functional behavior using the UML model. This new UML specification style is utilized to generate the SRN performance model. A new set of model transformation rules are introduced to support a scalable and automated model transformation process to generate the SRN model from the UML specification style. A general formula is derived to produce performance results by solving the SRN model. A well-established task assignment problem is considered to show the applicability of our performance modeling framework that utilizes the new UML specification style.

Unique contributions of Paper 6:

- A stepwise description of the framework that transforms the UML specification style into the SRN model, by specifying a service engineering approach using the UML with respect to a unidirectional graph (a different way of explaining service functional behavior than in the previous five papers).

- A new set of model transformation rules is introduced that utilizes the new UML specification style.

- A formula is derived to produce performance results by solving the SRN model.

- A well-established task assignment problem is considered to show the applicability of our performance modeling framework utilizing the new UML specification style.

Paper 7

From UML to SRN: A performability modeling framework considering deployment of service components

The following paper continues the incremental development of the modeling framework that concentrates on the performability modeling and evaluation of the distributed system. This paper considers the behavioral change of the system components due to failure and repair events. It also reveals how this behavioral change affected the system performance. The functionalities of the performability modeling framework are divided into two views:

the performance modeling view and dependability modeling view. The performance modeling view focuses on capturing the system's dynamics through a UML collaboration and activity-oriented approach. The performance information is incorporated into the UML diagram according to the *UML profile for MARTE.* The SRN model is then generated automatically from the system functional specification by utilizing the model transformation rules. The resultant SRN model is called the performance SRN. The dependability modeling view is responsible for capturing any changes in the system states using UML STM because of failure and recovery events of the system components. A dependability parameter is incorporated into the STM model according to the *UML profile for Modeling Quality of Service and Fault Tolerance Characteristics and Mechanisms Specification*. The SRN model is subsequently generated automatically from the STM model by utilizing the model transformation rules. The resultant SRN model is called the dependability SRN. After generating the performance and dependability SRN models, the model synchronization is used as the glue between the performance SRN and dependability SRN to generate the performability SRN model. The synchronization task guides the performance SRN to synchronies with the dependability SRN by identifying the transitions in the dependability SRN. The performance and dependability SRN are synchronized using guard functions. Once the performance SRN model is synchronized with the dependability SRN model, a merged SRN model known as the performability SRN model is obtained, and various performability measures can be evaluated. The applicability of our framework is demonstrated in the context of performability modeling and evaluation of a distributed system.

Unique contributions of Paper 7:

- A stepwise description of a framework that transforms the UML specification style into a SRN model to evaluate the performability of a distributed system (Previously mentioned papers consider the performance evaluation of a distributed system at an early stage)

- The functionalities of the performability modeling framework are divided into two views: the performance modeling view (generates a performance SRN model from UML specifications) and the dependability modeling view (generates a dependability SRN model from UML specifications)

- Performance information is incorporated into the UML diagram according to the *UML profile for MARTE*, and a dependability parameter is incorporated according to the *UML profile for Modeling Quality of Service and Fault Tolerance Characteristics and Mechanisms Specification*.

- New model transformation rules are defined to generate dependability SRN models.

- A model synchronization process is specified, in which the performance SRN model was synchronized with the dependability SRN model using guard functions to generate the performability SRN model.

- The applicability of our framework is demonstrated in the context of performability modeling and the evaluation of a distributed system.

Paper 8

A performability modeling framework considering deployment of service components

This paper focuses on the tool support of the performability modeling framework. The UML models are formally presented using cTLA to understand the precise semantics and correctly model the transformation to generate a performability SRN model. The formal semantics of UML models thus help to very efficiently implement the model and provide the tool support of our framework. The partial input models for model transformation in our framework are generated using the Arctis tool, which is integrated as plug-ins into the eclipse IDE. The other input models of performance and dependability views are generated as XML. The SHARPE tool is used on the evaluation side of our performability modeling framework. The detailed algorithms for the automatic model transformation are specified to generate a SRN performability model from the UML specification style. A new case study is introduced to show the applicability of the performability modeling framework.

Unique contributions of Paper 8:

- The UML models utilized in the performability modeling framework are formalized using the cTLA method to understand the precise semantics and correctly transform the model (Paper 7 includes the standard description).

- A complex case is considered while capturing the dependability behavior of system components.

- The detailed algorithms for the automatic model transformation from UML to SRN model are specified to evaluate the performability.

- The performability modeling framework is supported by the Arctis and SHARPE tools.

- A new and complex case study is introduced to show the applicability of the performability modeling framework.

Paper 9

Software performance evaluation utilizing UML specification and SRN model and their formal representation

This paper provides an extensive illustration of the development of the performance modeling framework that defines the formal method representation of UML models, mapping between the UML model and SRN model, as well as a detailed description of the tool support of the framework. The formalization of the UML collaboration model, production rules to delineate the detailed behavior of collaborations with the help of UML activity, and formalization of the UML deployment diagram using cTLA are the main focus of this paper. This paper also mentions the mapping process that shows the correspondence between the UML diagram and SRN model by utilizing their formal representation. Detailed model transformation algorithms to automatically generate the SRN model from the UML model are defined. Furthermore, the tool support of the performance modeling framework including the model validation, is described in this work. A new real case study, the Taxi Control System, is introduced to demonstrate the stepwise execution of the performance modeling framework.

Unique contributions of Paper 9:

- This paper provides a broad illustration of the development of the performance modeling framework with a formal method representation of UML models using the cTLA process.
- The cTLA production rules are illustrated to delineate the detailed behavior of collaborations with the help of the UML activity.
- A mapping process is exposed that shows the correspondence between the UML diagram and SRN model by utilizing their formal representations.
- Detailed model transformation algorithms are defined to automatically generate the SRN model from the UML model.
- The performance modeling framework, including model validation activities, are supported by tools.
- A new real case study, the Taxi Control System, is introduced to demonstrate the stepwise execution of the performance modeling framework.

CHAPTER 4

Related works

This chapter highlights a review of related studies that are aligned with the performance and performability evaluation of a distributed system as well as associated with some crucial parts of the solutions that have been considered in this work as research outcomes. To introduce related studies, several categories based on our research approach are defined below. Each category begins by with illustrating the relationship of the topic with the research conducted in this thesis. This chapter is a summary with an overview of related works that reflects the position of this thesis work.

UML model as main specification unit while generating analytical model

The UML collaboration was used as the main specification unit in this work to capture the functional behavior of the system through our modeling framework. The novelty of our approach compared to existing approaches lies in the utilization of collaborations to specify services by encapsulating the local behavior of service components and their interactions in self-contained reusable building blocks instead of separately considering the behavior of components. However, we would like to refer to one significant work, which is presented in [12], that provides the real motivation behind using UML collaborations as the main specification unit of our work. Although UML collaborations have been used in a service-engineering context [12], our approach extends the concept so that it can be used as an input model to capture the functional behavior of a system while generating analytical models. Moreover, several UML collaboration approaches were considered to derive the functional behavior of the software while generating analytical models: Kähkipuro developed a performance modeling framework to generate a queuing network with simultaneous resource possessions from the high-level UML notations; they used a UML collaboration diagram to visualize these networks [24]. King and Pooly utilized UML collaboration and state-chart diagrams to systematically generate GSPN models, which could be solved to find their throughput and other performance measures. This mapping was demonstrated using the example of communication via the alternating bit protocol, and the resulting GSPN was solved using the SPNP package [25]. Saldhana and Shatz described a methodology to develop a Petri net model of a system by deriving a form of the OPN called the "OPMs from UML state-chart diagrams" and connecting these using UML collaboration diagrams. They used a collaboration diagram as an interaction diagram that emphasizes the structural organization of the objects that send and receive messages [26]. Abdullatif and Pooly presented a method of providing computer support to extract performance models from a performance annotated UML diagram. In this approach, a UML collaboration diagram was efficiently utilized as part of an interaction diagram to demonstrate how a group of participants (objects and actors) in a system collaborate in some behavior [27]. Pooly presented an approach to demonstrate how simple UML designs could be systematically transformed into a process algebra model and thus be used to provide performance estimate by suitable annotation. UML collaboration diagrams describe the external interaction of objects, and state-charts describe instances of the internal behavior of classes, particularly in response to external stimuli [28]. Wet and Kitzinger demonstrated a methodology and tool called the proSPEX (protocol Software Performance Engineering using XMI) to analyze the design and performance of communication protocols specified with UML collaboration, class, and state-chart diagrams [29]. Jasmine and Vashanta proposed UML-based performance models to assess the design in a reuse-based software development scenario, where a collaboration diagram was used to describe the interaction among software components [30]. Verdikt et al. presented a MDA model transformation algorithm and tool to transform a high-level PSM into a low-level PSM by including the structural changes and the overhead of using CORBA as middleware. A UML collaboration diagram was used to contain the architectural pattern used in the system [31]. Gomma and Menasce proposed a method to performance engineer components based on distributed software systems, where collaboration diagrams were used to depict the dynamic interactions between the components and connector objects, i.e., instances of the classes depicted on the class diagrams [32].

Activities for drawing the detailed behavior of elementary collaborations

As mentioned previously, our modeling framework uses UML activities to describe the behavior of the collaborations and delineate the detailed behavior of the system, where activities can be understood as token flows, similar to Petri nets. However, the activity diagram has long been used to delineate the detailed functional behavior of distributed system, much like in our approach. Lopez-Grao *et al.* proposed a conversion method from an annotated UML AD to a stochastic petrinet model [33]. They developed a tool that addresses every model element from activity diagrams and ensures an automatic translation from ADs into GSPNs strictly following the process presented in their paper. They based their interpretation of the AD on their suitability to internal flow process modeling as expressed in [3]. Therefore, these models are relevant to describe activities performed by the systems. Campos and Merseguer considered the quantitative analysis of the behavior of software systems. They attempted to integrate the usual object oriented methodology with a performance modeling formalism, namely SPN, in a very pragmatic approach, which was supported with the UML language and widespread CASE tools. They utilized an activity diagram to represent the internal control flow of processes and described the behavior of a model element of the systems [34]. Bocclarelli and D'Amborgio proposed a method that exploits Q-WSDL to annotate reliability data onto a BPEL-based UML model of the composite service. The UML model was used to predict and describe the reliability of the composite web service. Specifically, the abstract model consisted of an activity diagram that described the abstract workflow to facilitate service discovery and retrieve a set of concrete services matching each abstract service interface [35]. Distefano, Scarpa, and Puliafito presented an evaluation methodology to validate the performance of a UML model and represented the software architecture. UML specifications were collected in an intermediate model, called the PCM. The intermediate model was translated into a subsequently evaluated performance model. The workflow among the service components was delineated using an activity diagram [36]. Woodside, Petriu, and Merseguer described a tool architecture called PUMA, which provides a unified interface between different types of design information through UML and different types of performance models. They utilized an activity diagram to demonstrate the entire behavior of the system [37]. D'Amborgio introduced a framework that was applied to the transformation of UML-type source models into target LQN-type models. The proposed approach was founded on precepts introduced by model-driven development MDA and utilized the set of related standards (MOF, QVT, and XMI). The activity diagram meta-model was used to add LQN tasks and specify their details in terms of entries, activities, and related calls [38]. Paci et al. represented a method that aimed to automate a process to measure computing performance starting from a UML-specified model. The proposed approach was based on open and well-known standards: UML to model the software, the profile for schedulability, the performance, the time specification to annotate the performance, and XMI to interchange metadata. The scenario contained in the use case was detailed in the activity diagram in this following work [39]. The UML activity diagram was utilized to realize the system dynamics in [40], where the authors demonstrated a tool that was a compositional approach to translate several UML diagrams into an analyzable Petri net model. Korherr et al. proposed a methodology to extend the UML 2 activity diagram with business process goals and performance measures and mapped it to BPEL. An activity diagram was used to model the business process and describe the control flows in software [41]. Tribastone and Gilmore explored the use of the stochastic process algebra PEPA as one such engine, providing a procedure to systematically map activity diagrams onto PEPA models. In this approach, the activity diagram was used as a behavioral element that models the coordination of both sequential and concurrent lower-level behaviors to carry out a computational step [42]. Yosr, Andrei, and Mourad described a mapping procedure of SysML activity diagrams to their corresponding DTMC and used a PRISM model checker to assess and evaluate the performance characteristics. SysML activity was mainly used here to highlight the inputs, outputs, sequences, and conditions to coordinate the behaviors in the system. Particularly, these behaviors might require time to execute and terminate [43]. Bharati and Kulanthaivel considered the integration of a performance and specification model while developing a tool to quantitatively evaluate software architectures at the design phase of the software life cycle. The activity diagram was considered to provide the complete detail of the execution system. Thus, any behavioral diagram given by the user was finally reduced to activity diagrams [44]. Petriu proposed a method to automate the derivation of LQN performance models from UML design models to fill the cognitive gap between the software development domain and the performance analysis domain. The activity diagrams were automatically derived by the graph transformations from a set of interaction (sequence) diagrams that described the system behavior and were partitioned in swim-lanes corresponding to different software components responsible for various activities [45]. Balsamo and Marzolla proposed an approach based on queuing network models to predict the performance of software systems at the software architecture level specified by UML. Starting from an annotated UML use case, activity, and deployment diagrams, a performance model was derived based on multi-chain and multi-class queuing networks. Activity diagrams were used in this work to describe the content of each use case in more detail; in particular, they described the computation performed on the system [46]. Matameni et al. presented a method to obtain performance parameters from a GSPN translated from a UML activity diagram to analyze the stochastic behavior of the system. An activity diagram was used here to show the activity and the event that caused the object to be in the particular state. The activity was triggered by one or more events, and it might result in one or more events that may trigger other activities or processes [47]. Bakshi et al. proposed a method to transform activity diagrams created in fuzzy UML into a fuzzy Petri to formally evaluate and verify the performance, rather than exact a visual analysis. Here, the activity diagram played an important role in the design stage of the software because of its momentous efficiency; it also helped to better define the operation [48]. El-Desouky et al. proposed a framework that applied the model-driven principles in the context of performance engineering, which transformed UML software models into LQN performance models. Activity diagrams were used to represent scenarios and illustrate the cooperation between several objects. Activity diagrams provided more direct ways of modeling concurrent forks and joints as well as hierarchal scenarios [49]. D'Amborgio et al. introduced a model-driven QoS management framework that provided both a standard (UML-based) notation to describe QoS-aware collaborative P2P service-based applications and a method for adaptive QoS management based on the automated building of performance models. In this approach, the activity diagram provided the behavior specification of the application by describing the flow of activities carried out in a given execution scenario [50]. Kreische proposed a method to model a business process using the UML in a way that facilitated the computation of the results, like throughput and resource utilization. The described activity diagrams are well suited to model the business process because the business task can be described by the activities and the relationships between them using object flows [51]. Antonio et al. proposed a method that used the MARTE profile to derive the performance requirements of each action in a UML activity diagram from the requirements of the containing activity and some local annotations [52]. Arpinen et al. presented an efficient method to capture the abstract performance model of streaming data real-time embedded systems, where UML 2 was used to model the performance and serve as a front-end for a tool framework that enabled simulation-based performance evaluation and design-space exploration. UML 2 activity diagrams were selected here as the view for application workload models to present the control and data flow between functional elements of the application [53].

Reusability of software components

The reusability of collaborative building blocks is one of our important design issues. Here, the local behavior of the software component is not only reused, but the interaction among the components is also captured and reused in an encapsulated and self-contained way. One approach presented by Frank Alexander Kraemer and Peter Herrmann in [12] shows how to design reusable software components in a self-contained way in a service-engineering context, which provided the main inspiration for capturing the functional behavior of a system with the help of reusable software components while generating analytical models using our framework to evaluate the performance and performability. However, in most cases reusability is achieved for separate software components, which makes the system development process inefficient and slow. For example, Moorsel et al. discussed the software reusability strategies for performance and reliability modeling tools. Special emphasis was placed on web-embedded tools and the potential interaction between such tools. They presented an application programming interface for system analysis tools, which allowed for the quick embedding of existing tools in the web and generally simplified programming analysis tools by structured reuse [54]. Woodside provided an approach for component-based modeling, which matched the capabilities of component-based software engineering and generative programming. Here, a component library will be specific to a domain, like web services, or to the elements of a single product line [55]. Kappler et al. proposed a reverse engineering approach to derive performance models from implemented software components. They focused on one specific step of the reverse engineering approach, namely the static analysis of Java code to derive abstract behavioral performance models of component services. In this approach, software architects could reuse the resulting performance models in different architecture models [56]. Wang et al. proposed a method to model the performance of integrated embedded control software design, where they assumed that the functional model is to be constructed by integrating existing reusable software components [57]. Kulanthaivel et al. developed a performance evaluator for component based software architectures, in which the software architecture model was formed using reusable software components [58]. Tawhid et al. proposed a method that aimed to automatically derive a product performance model from a UML SPL model, where SPL contains all the possible artifacts contained in all the products [59]. Bui et al. presented a component-based infrastructure to model the performance and power of parallel scientific applications [60]. Hardung et al. reused software components in distributed embedded automotive systems [61]. Gooma et al. investigated the design and performance modeling of component interconnection patterns, which defined and encapsulated the way the client and server components communicate with each other. They also aimed to eventually specify both the architecture and performance of a large component-based distributed system in terms of new components, as well as predefined components and inter-component communication patterns that are stored in a reuse library [32]. Schmidt et al. presented a technique to automatically compose reusable software components for mobile devices. They described an automated variant selection engine based on a CLPFD solver that could dynamically derive a valid configuration of reusable software components suitable for a target device's capabilities and resource constraints [62].

Annotation of UML model

To quantitatively analyze the UML model, it must be annotated because UML specifications only accurately provide the modeling facility of the functional behavior of the system. In fact, UML profiles have long been used to annotate UML models. Much like in our study, several approaches have already considered UML profiles to annotate UML models. Marzolla proposed to annotate the UML diagrams using a subset of annotations defined in the UML Profile for Schedulability, Performance, and Time specification to quantitatively evaluate the performance [5]. Gilmore et al. proposed to annotate the UML diagram with the profiles for MARTE that were used to specify the timing behavior of the actions and denote the output variables of concern to the modeler [42]. The approach in [67] proposed a technique to analyze the performance effects of a given aspect on the overall system performance after determining the composition of the aspect model with the primary model of a system. The performance analysis of UML models was enabled by the UML Performance Profile for Schedulability, Performance, and Time, which defines a set of quantitative performance annotations to be added to a UML model. In [46], UML-ψ transformed annotated UML diagrams into a simulation model, implemented the model using process-oriented simulation, and evaluated the performance model. In this approach, the UML model must be annotated according to a subset of the Profile for Schedulability, Performance, and Time Specification. In [76], the authors described a plug-in for the Rhapsody tool, which demonstrated how UML models with SPT annotations could be analyzed using the Times tool, which is a tool for modeling and schedulability analysis, and code generation for timed systems. In [77], the authors presented a new performance modeling approach for designing embedded real-time systems using UML 2, where the existing UML meta-model had been extended by defining stereotypes to include the message latency and execution time in UML state-charts. The UML Performance Profile for Schedulability, Performance, and Time was applied in [78] to define a performance engineering methodology for the performance analysis of models designed using UML sequence diagrams. The approach used in [59] included the PUMA transformation approach of annotated UML models with MARTE annotations, where the variability expressed in the software product line model was analyzed and bound to a specific product, and the generic performance annotations were bound to concrete values for the product. The focus of the study performed by [79] was on the analysis of performance effects of different security solutions modeled as aspects in UML. For performance analysis, the authors used techniques that were previously developed in the PUMA project, which provided input for the UML models annotated with the standard UML Profile for Schedulability, Performance, and Time. The authors in [80] introduced a new SPE tool that fit in the OMG framework and implemented most of the features. The tool allowed designing UML diagrams annotated according to the UML Performance Profile for Schedulability, Performance, and Time, and automatically generates a performance model in terms of GSPN. In [81], the author examined the problem within the UML context to show how performance anti-patterns could be defined and detected in UML models by mean of OCL where UML model was annotated with the MARTE profile. In [36], the author presented an evaluation methodology to validate the performance of a UML model, representing software architecture. The proposed approach was based on open and well-known standards: UML for software modeling and the OMG Profile for Schedulability, Performance, and Time specification for the performance annotations into UML models. Booy et al. proposed a Method for constructing performance annotation model based on architecture design of information systems utilizing UML activity diagram and colored petri net where UML model was annotated using Profile for Schedulability, Performance, and Time [82]. Shen et al. proposed a graph-grammar based method for transforming automatically a UML model annotated with performance information according to Profile for Schedulability, Performance, and Time into a LQN performance model [83]. The work in [84] proposed and implemented a method for transforming a UML 2.0 model with performance annotations into an equivalent CSM. The input to the transformation algorithm was a UML 2.0 model generated by a UML tool, either as an internal data structure, or as an XML file according to the XMI standard. The software specifications models in UML 2.0 were annotated using the UML Profile for Schedulability, Performance, and Time or its successor UML MARTE. Another modeling approach, known as TUT-Profile, for UML 2.0 together with System-on-Chip architecture exploration tools provided an explicit control of real-time constraints at UML level and the transformation of the original UML model using back-annotated results of SoC architecture exploration [85]. Using a stochastic modeling approach, based on the UML, and enriched with annotations that conform to the UML profile for Schedulability, performance, and time, the authors proposed a method for assessing QoS in fault-tolerant distributed systems [86]. In [87], author proposed using the MARTE profile to derive the performance requirements of each action in an UML activity diagram from the requirements of the containing activity and some local annotations. In [31], the author presented an MDA model transformation algorithm and tool for transforming a high level performance specific model to a low-level performance specific model where the performance specific model was a UML model annotated with performance information using the UML performance profile. In [65], the LQN model structure was generated from the high-level software architecture showing the architectural patterns used in the system, and from deployment diagrams indicating the allocation of software components to hardware devices. The LQN model parameters were obtained from detailed models of key performance scenarios, represented as UML interaction or activity diagrams annotated with performance information according to the proposed UML performance profile. The design of the tool AgroSPE followed the architecture proposed by OMG in the UML Profile for Schedulability, Performance, and Time specification described in [88]. IMPACT is a performance plug-in that makes use of the modeling capabilities of the Papyrus tool and the relational QVT model transformation implementation of mediniQVT to produce an LQN performance model of a MARTE annotated UML design [89].

UML deployment diagram and deployment decision-making

One of the primary focuses of this thesis was to examine the UML deployment diagram to reveal the physical layout of the distributed system and to assess the deployment mapping of software components that was closer to the optimal solution. The target of the deployment decision-making was to achieve a reduced service turnaround time by maximizing the utilization of resources while minimizing any communication between the processing nodes, thus offering a high system throughput while taking into account the expected execution and inter-node communication requirements of the service components on the given hardware architectures. This fact has been completely ignored by relevant approaches, where the deployment diagram was only accountable for identifying the physical layout and assignment of the software artifacts to the physical components without any indication of the manner to assess the solution. Marzolla described a simulation-based performance model generation method, where the UML deployment diagrams were used to describe the physical environment in which the software system executes [5]. Merseguer proposed a model-based performance evaluation of web service, where the deployment diagram was utilized to model the deployment of the software components in the hardware platform [63]. In addition, Pooly highlighted a technique used to generate an extended queuing network model from the UML specification, where the MM was a basic model representing the components consisting of the system and its relationship to the hardware platform, and the building of the MM was dependent on the UML deployment diagram [64]. Petriu proposed a method to derive performance models from the UML models using graph transformation, where the UML deployment was used to delineate the physical configuration of the focused system [65]. Kulanthaivel et al. developed a performance evaluator for component-based software architectures, where the deployment diagram was used on the input side of the evaluator to determine the interconnections between the processing nodes [58]. Balsamo derived a performance model for component-based software engineering, where the deployment diagram modeled the available resources and its characteristics [66]. Shen et al. developed a performance analysis method of UML models using an aspect-oriented modeling technique, where the deployment diagram highlighted the deployment of high-level software components for hardware devices [67]. Tawhid et al. proposed a method of automatic derivation of a product performance model from a UML SPL model, where the SPL deployment diagram contained all the potential artifacts in all of the products [59]. Silva et al. introduced a new methodology that employed an architectural framework that could be used to automatically generate simulation models on the basis of the UML model diagrams, which were created by requirement engineers and software system architects. The deployment diagrams employed here were used to map the software components used in the sequence diagram model and the node types [68]. Mania et al. proposed a methodology that automatically constructed analytical models and initiates potential performance improvements for the systems under study, where the deployment diagram presented the configuration of a set of run-time processing nodes and the components running on each node [69]. Huang et al. proposed a framework to automatically integrate the middleware component interactions and their performance attributes to the application UML model, where the authors changed the original UML deployment diagram to add a stub component to the client node and middleware service components to the server node [70]. Merseguer et al. utilized a deployment diagram to demonstrate the physical structure of the system, which was necessary to describe the resources, where to allocate the modules of the architecture, and their connections via a network [71]. Scarpa et al. discussed the implementation of the software performance engineering development process, where the deployment diagram described the deployment of the components on the elaboration infrastructure [72]. Klapiscak et al. developed a model-driven approach for the construction, composition, and analysis of services on sensor networks, where the deployment diagram was used to map the design of a set of fielded software and hardware assets of the sensor network [73]. Emmerich et al. described a model driven performance analysis of the enterprise information system, where the deployment diagram demonstrated the relationships between the application components, architecture components (such as containers and database), infrastructure (CPUs, network links) and the application clients [74]. Tsadimas et al. proposed using the UML to model all aspects of distributed system configuration process by extending and integrating different diagram types, where UML deployment diagrams are commonly used to represent network architectures [75].

Formal method representation of UML models

The importance behind this task was to understand the semantics of the UML models that were previously informally defined. This formalization provides accuracy in the model transformation with the help of incremental model checking. Formalization of UML models in this thesis was based on the temporal logic of action. Available approaches might differ from our main concepts. A previous study [13] established the semantic foundation of the collaborative building blocks and how they were constructed using TLA. The specification style cTLA/c for collaborations had also been outlined. Elementary collaborations were mapped to simple cTLA processes, and composite collaborations were expressed using corresponding compositional cTLA processes. The authors formulated a set of production rules that described how the graph of an activity consisting of activity nodes and edges was transformed into cTLA actions, and provided an example. To create entire systems, collaborations were constructed, which, gave the semantics in cTLA/c, could be formalized as cTLA process compositions [13]. In [90], the authors presented an overview of their aspect-oriented formal design analysis framework and how it could be used to design and analyze performance properties. In [91], the authors aimed to use the UML in the performance modeling process to introduce the benefits of performance analysis with process algebras without the complexities and conceptual challenges that were normally associated with formal description techniques. In [64], the author introduced a methodology that included steps starting from gathering the performance data needed to build the model to the algorithms used to convert the design model into an EQN performance model. In [92], the author described the annotation of UML class diagrams with fragments of the Object-Z specification language. IBM proposed another approach, where the OCL language [93] was used as a standard formal specification language to formalize the UML diagrams. In [94], the author proposed the SHE method, which enabled the generation of formal executable models on the basis of the expressive modeling language POOSL. Kähkipuro developed a performance modeling framework to generate a queuing network with simultaneous resource possessions from high level UML notations, such that the model could be solved for the relevant performance metrics [24]. Lopez-Grao *et al.* proposed a conversion method from an annotated UML activity diagram into a SPN model. They developed a tool that interacted with every model element from the activity diagrams and ensured an automatic translation from ADs into GSPNs [33]. In [95], formal methods were introduced into the real-time embedded software testing field and a real-time extended finite state machine. A reactive system-oriented testing method based on both state charts and temporal logic was demonstrated in [96]. In [97], the author presented an approach for the automatic generation of a performance evaluation model based on a queuing network model from a software architecture specification described through a message sequence chart. In [98], the author proposed a step towards formal semantics for the interaction diagrams of UML by defining a partial order between messages and actions and to generate a Petri net that defined the semantics of this diagram. The paper in [99] established the basis for the development of a formal semantics for UML statechart diagrams based on Kripke structures. In [100], the author discussed a complete formalization of UML state machine semantics. The formalism was given in terms of operational semantics, which was used for code generation, verification, and simulation for the state machine diagram. The approach used in [101] proposed a revised semantic interpretation of the UML statechart diagrams. In particular, hierarchical state machines may be properly encapsulated to enable independent verification and compositional testing in this work. The aim of this paper [102] was to describe an approach to integrate the information in sequence diagrams and to check it for consistency and completeness using additional information. In [103], the authors defined formal execution semantics for UML activity diagrams that was appropriate for workflow modeling and focused on the requirements level by assuming that the software state changes did not take time. The paper [65] proposed a graph grammar based on transformation from UML design models into LQN performance models. The paper in [104] defined how Activity Graphs can enable process semantics in the CSP language. Gehrke et al. [105] provided semantics by translating an activity diagram into a Petri net. The paper showed how activity graphs can provide process semantics in the CSP language. In [106], the author provided rigorous semantics of the UML activity diagrams for the description of dynamical system behavior. In [107], the author defined formal execution semantics for UML activity diagrams that were suitable for workflow modeling, where the goal was to support execution of workflow models and analysis of the functional requirements that these models satisfied. In [108], the authors provided a formal foundation of the distributed workflow executions, where the statechart formalism was adapted to the need of the workflow model to establish a basis for the correctness reasoning and run time support for complex and large-scale workflow applications. The approach used in [109] introduced the language of state-charts that presented only a brief discussion of how its semantics could be defined. In fact, the works presented in [13] and [24] mainly affected and motivated our work of formalizing UML models and mapping to analytical models to show the correspondence between the two, which helped to provide a model that confirms the facility and tool support of our framework.

Performance modeling framework

Several approaches have been considered to generate a performance modeling framework from system design specification. However, we developed a complete and comprehensive framework for performance modeling of a distributed system focusing on the key characteristics mentioned in Chapter 2, where some ideas presented in this work are analogous to earlier attempts to derive the concept and framework for a distributed system performance evaluation, such as Kähkipuro, who developed a performance modeling framework to generate queuing network using simultaneous resource possessions from high level UML notations, such that the model could be solved for the relevant performance metrics [24]. Lopez-Grao *et al.* proposed a conversion method from an annotated UML activity diagram into stochastic petrinet model [33]. Trowitzsch and Zimmermann proposed the modeling of technical systems and their behavior by means of UML, and a transformation into a SPN was established for the resulting models [110]. Abdullatif and Pooly also presented a method to provide computer support for extracting Markov chains from a performance annotated UML sequence diagram [27]. Zimmermann and Hommel presented a SPN model of communication failure and recovery behavior of future European train control systems with performance evaluations, demonstrating the significant effect of packet delays and losses in the reliable operation of high-speed trains [111]. Distefano *et al.* proposed a potential solution to address software performance engineering that evolved via system specification using an augmented UML notation, creation of an intermediate performance context model, and generation of an equivalent SPN model whose analytical solution provided the required performance measures [36]. D'Ambrogio proposed a framework to transform source software models into target performance models using meta-modeling techniques to define the abstract syntax of models, interrelationships between model elements, and the model transformation rules [38]**.** In [112], StoCharts had been proposed as a UML statechart extension for the performance and dependability evaluation, and had been applied in the context of train radio reliability assessment to show the principal tractability of real cases. In [113], the author proposed extensions to UML state diagrams and activity diagrams to enable the association of events with exponentially distributed and deterministic delays. The paper in [114] proposed a Meta modeling procedure devoted in providing a reference model for use by decision makers in the performance evaluation of ITN, where the case study UML model was translated into simulation software and the performance measures were obtained by the simulation results. In [115], the author presented a simulation framework that could be used to generate simulation programs directly from UML notations. Moreover, a tool had been generated to demonstrate the feasibility of using this framework to perform such a transformation automatically.

Performability modeling framework

The extension of our performance-modeling framework was initiated to cover the performability evaluation of the distributed system. The performability evaluation was performed by generating analytical models from the system design specification using reusable software components and the dependability behavior of system components. Some early approaches have been considered to capture the essential requirements to generate our performability modeling framework, such as work performed by Sato et al., who developed a set of Markov models to compute the performance and reliability of web services and detecting bottlenecks [116]. Another initiative focused on model-based analysis of performability of mobile software systems via a general methodology that used design artifacts expressed in a UML-based notation. Inferred performability models were generated based on the SAN notation [117]. Subsequent efforts proposed a methodology for the modeling, verification, and performance evaluation of communication components of the distributed application building software, which translated UML 2.0 specifications into executable simulation models [118]. Gonczy et al.

also mentioned a method for high-level UML models of service configurations captured by a UML profile dedicated to the service design; and then the performability models were derived using automated model transformations for the PEPA toolkit to assess the cost of the fault tolerance techniques in terms of performance [119]. Moorsel and Haverkort constructed a framework, the so-called performability evaluation framework, in which the quantitative evaluation of both types of systems could be discussed. The author presented a general view, a systems view, and a modeling view on the performability evaluation, resulting in a framework, which naturally fit the known measure definitions, modeling methods, and solution techniques. The paper rather described some general views of performability evaluation without any focus on the applicability of their framework [120]. Dalibor et al. presented a performance and dependability evaluation of fault-tolerant multiprocessors, where two specific architectures were analyzed taking into account system functionality, actual workloads, and the failures of system components as well as the inter-component dependencies. Object-oriented software design and process-oriented simulation techniques were used for model construction allowing sophisticated performance and dependability analysis of massive parallel systems [121].

Summary of related works that reflect the position of this thesis work

Approaches presented in this chapter are summarized in Table 4.1, where the comparison between our approach (2nd row in Table 4.1) and other approaches is mentioned based on the key characteristics (1st row in Table 4.1 and also referring to the Chapter 2), resulting in the main focus of our work. The comparisons are summarized in Table 4.1. The detailed results are provided in Appendix A. Most of the existing approaches use the UML model to define system or service functional behavior, with very few exceptions. However, a few ([our approach, [12], 30, 32, 54, 55, 56, 57, 58, 59, 60, 61, 62]) approaches take the advantage of using the reusable software components to describe the system functional behavior, where reusability is achieved only with respect to the local behavior of the software components, although none (except our approach, [12]) of the approaches describe or show how software components ensure reusability to delineate system functional behavior with any concrete example. For annotation of the UML model, we use the standard SPT and MARTE methods defined by OMG that have been well established by the scientific community, which are also used in many current approaches (shown in Table 4.1). Some of these approaches ([5, 58, 59, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75]) use the UML deployment diagram to describe the system physical infrastructure, although none of these approaches discuss (except our approach) any methods on how to specify and assess the deployment mapping of software components on a physical infrastructure for a large distributed system. Some of the existing approaches (shown in Table 4.1) confirm the formal representation of UML models, which also exhibit very important characteristics to the UML, but does not convey the formal representation of the model. However, very few of these approaches propose (shown in Table 4.1) UML model validation to demonstrate the correctness of the model generation while providing the tool support of the framework. Model validation is important to accurately perform the model transformation. We mainly concentrate on generating analytical models specifically Markov and Petri nets (SPN, SRN) for the

Table 4.1 Summary of the existing approaches and comparison with our approach

References	System/Service specification using UML	Annotation of UML using SPT/MARTE	Specifying method for deployment mapping	Formalization of UML model	UML model validation	Markov/PN Performance/ performability model generation	Tool support
This thesis work	√	√	√	√	√	√	√
[33, 34, 40]	√	√		√		√	√
[36, 37, 39, 42, 80, 84, 86, 88, 110]	√	√				√	√
[38, 58]	√	√		√			√
[94, 100]	√			√	√		√
[5, 29, 35, 44, 46, 52, 53, 72, 76, 78, 79, 83, 89, 119]	√	√					√
[25, 43, 51, 54, 113, 117]	√					√	√
[27, 47, 59, 63, 82]	√	√				√	
[55, 64, 91, 96, 112]	√			√			√
[95, 103, 108, 109]	√			√	√		
[48, 98, 105]	√			√		√	
[49, 68]	√				√		√
[90]	√	√		√			
[24, 45, 92, 93, 97, 99, 101, 102, 104, 106, 107]	√			√			
[31, 50, 65, 66, 67, 70, 71, 74, 77, 81]	√	√					
[56, 60, 61, 62, 85, 115, 118]	√						√
[26, 28, 111]	√					√	
[87]		√					√
[30, 32, 41, 57, 69, 73, 114]	√						
[116]						√	
[120]							

performance and performability evaluation using our modeling framework because of the advantages (mentioned in Chapter 1 and Section 2.6) provided by these models. Many of the existing approaches are very similar with our approach and others differ (detailed list in Appendix A). There are very few approaches that provide a performability evaluation using a complete framework, such as that presented in ([54, 116, 117, 118, 119, 120, 121]). Tool support is an integral part of the performance and performability modeling framework, which provides easy to handle, automation, and correct processing for model transformation and generation. We use Arctis and SHARPE tools, which are mature and well-established tools [10], [11]. Some of the existing approaches provide tool support of their frameworks (although not all of the tools are mature and complete (detailed list in Appendix A)).

CHAPTER 5

Concluding remarks

Studies performed in this thesis began with the development of a performance and performability framework realizing the importance of the rapid and expressive development of system functional behavior. Afterwards, establishment of an efficient deployment configuration of software components was achieved by satisfying the non-functional properties of the system. Subsequently, a scalable and automated method of model transformation was attained to generate analytical models for performance and performability evaluation of large and multifaceted distributed systems. Incremental model checking is provided to confirm that the method of model development and model transformation is correct and efficient.

Utilities and results of this conducted research are presented in the included papers and are summarized in the following list:

I. Designing and applying reusable collaborative building blocks to delineate system functional behavior, which is utilized as an input model for performance and performability evaluation of the distributed system. The definition of the collaborative building block is given as an encapsulated and self-contained method to capture the local behavior of service components and its interaction. The way in which we define the collaborative building block gives the opportunity of reusability to draw the system functional behavior for particular application domains using existing building blocks.

II. Incorporating non-functional properties that reflect the performance and performability attributes of the system.

III. Applying cost functions that sufficiently solve the complex problem of deployment mapping with respect to given requirements for large and multifaceted distributed systems and provide a solution in a distributed manner, which is close to the optimal one.

IV. Developing model transformation rules and algorithms to generate performance and performability models in a scalable and automated manner from the system design specification while considering optimal deployment mapping.

V. Providing tool support with incremental and automated model checking facility for both performance and performability evaluation.

Papers 1 – 9 contribute to the given 5 areas as shown in Table 5.1.

Table 5.1 Contributions of the included publications

Paper	**Contribution**				
	I. Applying reusable collaborative building block for capturing system functional behavior	II. Incorporating non-functional parameters	III. Deployment decision making	IV. Automated and scalable model transformation	V. Tool support of the framework
1	√	√			
2	√	√			
3	√	√			
4	√	√	√	√	
5	√	√	√	√	
6	√	√	√	√	
7	√	√	√	√	
8	√	√	√	√	√
9	√	√	√	√	√

To review our results, we recapitulate the research questions described in Section 1.2:

- ***What is the method that will allow us to provide a rapid way to specify the functional behavior of a distributed system that can easily be combined with a model of physical infrastructure to represent deployment strategies?***

  The main approach was to use the UML collaboration to provide an abstract and structural view of the service delivered by the distributed system, where the collaboration was represented as functional, comprehensive, and self-contained building blocks. The behavior of the collaboration was demonstrated using a UML activity-oriented approach. This UML specification was presented in all 9 papers in terms of specific scenarios and later on, in a much more generalized manner. Papers specified in part II describe the manner in which the collaborations were utilized to capture the system functional behavior by combining all of the sub-functionalities provided by different participants of the system. This indicates that the service provided by the system consists of the sub-services and their behaviors can be demonstrated using activity in a self-contained form. Thus, the sub-service functionality is reflected in the system overall functional behavior in a correct manner. Moreover, the composition of sub-services functionalities defines the more complex service behavior very efficiently. The developed collaborative building block was archived in a library for later use, which provided a rapid and efficient method to apply building blocks for the developing service of a particular application domain instead of starting the development process from scratch. Papers 1 - 9 also provide a detailed description of how the collaborative building blocks are reused to build services for a particular application domain with the help of real and complex cases.

- ***How can the deployment mapping of software components be specified considering the QoS requirements such that the performance of a service or a system over a particular physical infrastructure with resource constraints can be assessed?***

  The distributed and dynamic nature of distributed systems make it difficult and challenging to assess the efficient deployment mapping of service components on the execution environment, which satisfies the non-functional requirements. This problem of assessing the deployment mapping of software components over the physical infrastructure has been recognized as an optimization problem. The solution for this problem is described in Papers 4 – 9, which defines the cost functions, where the first cost function was derived based on the communication and execution cost. Subsequently, the cost function refined the targeting load balance among the physical nodes in the system based on the overhead cost, communication cost, and execution cost. The scenarios presented in these papers are real case scenarios and a modified version of a well-known task assignment problem, with a known optimum. Thus, the derived cost functions have been validated by checking the solution using different scenarios.

- ***How do we incorporate non-functional properties into UML models that reflect the performance and performability attributes of the system?***

  Modeling of the system functional behavior is realized by UML models because UML is a widely used modeling language that is accepted universally and is commonly known by the scientific community [3]. When referring to the quantitative evaluation of the distributed system modeled by UML, the UML model incorporates performance and performability related parameters to perform the quantitative analysis. To fulfill the requirements, it is essential to extend the UML model to associate performance and the performability related parameters. As a result, we annotated the UML model using the *UML Profile for MARTE: Modeling and Analysis of Real-Time Embedded System* and *UML profile for*

Modeling Quality of Service and Fault Tolerance Characteristics and Mechanisms. These profiles provided stereotypes and tagged values that were required for the quantitative prediction and assessment of systems considering both the hardware and software characteristics [4] [9]. Papers 1 - 9 provide a detailed description of how to incorporate non-functional properties into UML models that reflects the performance and performability attributes of the system.

- ***How can building blocks from the functional behavior models be translated to building blocks in performance and performability models?***

  System functional behavior is modeled using UML collaborations in the form of encapsulated building blocks in a self-contained manner that capture the local behavior of the participating components as well as the necessary interactions among them. The collaborative building block is the basic specification unit in our work that combines with each other to provide the system overall behavior. Because the system behavior is expressed as a composition of collaborative building blocks, the collaborative building block is used as input for the model transformation process to generate analytical models. The model transformation process provides rules that convert the building blocks of the UML specification style into the building blocks of performance and performability analytical models. Section 2.6 and Papers 4, 5, 6, 7, and 9 focus on the model transformation rules. The model transformation rules presented in the papers are not traditional rules that only transform each element of the UML model into an element of the analytical model instead of the rules that are defined in such a way that, the single collaborative building block is directly transformed into a building block in the performance and performability models.

- ***How do we conduct the automated model transformation in a scalable way to accomplish performance and performability evaluation of the system?***

  To conduct the performance and performability evaluation, the analytical model is generated from the UML specification style, which captures the distributed system functional behavior. Generating the analytical model from UML model may be achieved in automated way following several model transformation rules, which consider the reusable collaborative model specification, deployment mapping decision that is realized by cost functions, and performance and performability related parameters as input. Model transformation rules are archived in a library for later use to perform the model transformation by applying them instead of understanding the inner complexity. Papers 1 - 9 consider several real and artificial case studies to show the scalability, rapidness, and automation in the model transformation for large and complex distributed systems. The transformation rules are defined in such a way that it ensures generality and reusability while conducting the model transformation. Papers 8 and 9 define the algorithms used for automated model transformation considering the model transformation rules.

- ***How do we ensure that we obtain the complete set of model transformation rules?***

  It is a challenging task to ensure that the set of model transformation rules we presented in our framework are sufficient to conduct the model transformation from UML models to analytical models. UML is utilized to capture the distributed system functional behavior, which is sometimes very complex. Moreover, in distributed systems, to achieve specific tasks, the software components need to execute as well as communicate with each other over this highly distributed environment, where interconnection exists among the hardware nodes. By considering the above mentioned factors, we considered several real and artificial case studies mentioned in Papers 1-9 and performed the model transformation for these case studies. This demonstrated that the set of model transformation rules were sufficient to consider the complex behavior of distributed systems and to accomplish the model transformation process using our performance and performability framework.

- ***How can the correctness of the UML model specifications and model transformations be ensured?***

  The UML collaboration oriented approach was used to compose the system functional behavior from sub-functionalities provided by the service components. Moreover, the UML model was formalized using cTLA mechanisms mentioned in Papers 8 and 9, where the superposition rule assures that the properties of the individual collaboration are reflected in the system because the system is the composition of this individual collaboration. The following papers also mentioned the algorithms, which were used to perform the automated model transformation with the specified model transformations rules in the correct way. Moreover, the correspondence between UML models and analytical models was demonstrated. This correspondence was visualized using real, complex, and generalized case scenarios and demonstrated the applicability of our framework.

In summary, the included papers present a rapid, scalable, and automated approach that spans from capturing the system functional behavior to generating analytical models for the performance and performability evaluation of the distributed systems in accordance with the factors specified in the introduction section and with the specified research questions. The approaches presented in this work are sufficiently general to illustrate a wide variety of application scenarios to demonstrate the applicability of our developed framework. We consider non-functional parameters which are very much realistic that have been utilized to derive performance and performability results. The results generated by our framework together with the validation of the approach rationalize the efficiency of our solutions. Lastly, we were able to prove that our framework was not only confined by the theoretical approach, but was also implemented with the tool support in a practical setting. The scenarios we considered were sufficiently complex to justify the functionality of our framework. Thus, this approach is scalable and can also handle a variety of real system specifications to conduct a performance and performability evaluation.

The same framework can be applied with simulations, though in this thesis, the focus is on analytic models. The performance and performability models used in this thesis can be solved both by analytic approaches and by simulations. Simulations under less restrictive assumptions than the analytic model can be used for cross-validation of the results.

CHAPTER 6

Future directions

There are many interesting research paths that are identified in this section that can be specified as future directions of the work presented in this thesis. The following items are not in listed according to priority.

Development of the missing plug-ins for the Arctis tool

The Arctis tool does not have support to define the UML deployment diagram, UML state machine diagram, and to incorporate performance and performability information into UML models. The development process is ongoing to implement the missing plug-ins for the Arctis tools to generate all of the input models for the performance and performability evaluation of the distributed system.

Analysis of UML models

We have specified the formalization of collaborative building blocks and other UML models. This enables automated model checking of the specified UML models. Further extension of this work would be to derive additional theorems and specifications for the extensive checking of error situations, which require additional investigation of the semantics of the UML models depending on the different and complex application logic.

Complete profile for annotation of UML models

We annotated our UML models for the performance and performability evaluation according to the *UML SPT* profile and subsequently, using the *UML MARTE* profile and *UML profile for Modeling Quality of Service and Fault Tolerance Characteristics and Mechanisms*. In addition, we also introduced several stereotypes and tagged values for annotating the UML models used in this work. Thus, we are working towards compiling a complete profile that will help to annotate our UML models in accordance with the performance and performability requirements.

Cost considerations regarding the migrating workload

The migrating workload is a demanding task and requires continuous investigation of the system execution environment. The deployment logic presented in this thesis should be capable of providing new configuration of the deployment mapping, when any change in the execution environment or workload configuration is encountered. Thus, the new deployment mapping might provide a better result. In that case, a new parameter in the cost function is needed to be introduced as the migration cost because of the changing deployment location of the software components in the execution environment.

Larger and complex problem sizes

One of the focuses on the results of this thesis was to build a method that ensured scalability in model transformation for large and complex problems. However, there remains a space for designing and implementing the model transformation for larger problem sizes with respect to the UML models used in this work, assessment of the deployment mapping, and generation of performance and performability models. Extension of this method can be considered from two main directions: where the more comprehensive scenarios might be considered for large networks, including more physical nodes and clusters, and the other direction might be the simultaneous deployment of a larger amount of services that can be encountered while designing cost functions.

Considering real time properties

While capturing system functional behavior, it might be essential to focus on the real time properties of some application domains. We already formalized the semantics of the UML model by focusing on the system design specification using cTLA. However, this solution can be used to specify the real time properties for some application domains with an introduction of the necessary modeling elements.

Introducing dynamic configurations

In our work, we consider service properties in design time, which give a static view. However, the properties of services in run time as well as the dynamic nature while capturing the functional behavior should be considered. In some systems, it may be desirable to deploy new functionality at runtime by exchanging existing components or making new components available via some discovery mechanisms. Currently, this form of dynamics is not directly addressed by our approach. As part of our future studies, we will specify the semantics that will capture the system functional behavior in such a way that it will be able to consider the internal behavior of collaborative building blocks in run time.

Procedure for deriving costs

Costs such as execution, communication, and overhead cost are mentioned instinctively in the application scenario and are utilized for reasoning of the deployment logic. A method that would be useful to derive the costs automatically and in realistic way is needed. The probable applicable method could be from code analysis, constant transition costs, various offline measurements, or other predication methods on expected demand [2].

Migrating Load

The deployment logic we introduced considers deployment mapping of the service components on the execution environment for a fixed topology. The target of the deployment logic was to ensure load balancing taking into account a specific and fixed number of physical nodes. However, in a real case, the topology is not static, but is very dynamic, where it might be a scenario that consists of topology changes for any constraint that requires an efficient method of load balancing among the physical nodes. One of the potential extensions in the design of our deployment logic is to consider the load balancing among the physical nodes because of the sudden change in execution environmental topology.

Providing feedback to functional design

We performed functional changes in the system design process based on the early assessment of software performance and performability evaluation. This process was not automatically performed. Further extension of our work would be to make the process automated to provide feedback to identify UML anti-patterns and to then change the functional design accordingly.

Providing feedback to improve deployment mapping

The process of comparing numerous execution environments and determining the optimal deployment mapping of the service components over the physical environments has not yet been completed through our current work. This goal can be achieved using the fixed-point iteration method, which provides feedback to the design alternatives of the execution environment.

Bibliography

[1] Frank Alexander Kraemer, "Engineering Reactive Systems: A Compositional and Model-Driven Method Based on Collaborative Building Blocks" PhD thesis, Norwegian University of Science and Technology, 2008

[2] Mate J. Csorba, "Cost-Efficient Deployment of Distributed Software Services", PhD thesis, Norwegian University of Science and Technology, 2012

[3] OMG 2009, "UML: Superstructure", Version-2.2

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

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

[6] K. Efe, "Heuristic models of task assignment scheduling in distributed systems", Computer, 1982

[7] Peter Herrmann, "Problemnaher korrektheitssichernder Entwurf von Hochleistungsprotokollen", PhD thesis, Universitat Dortmund, 1997

[8] Peter Herrmann and Heiko Krumm, "A Framework for Modeling Transfer Protocols", Computer Networks, Vol.34, No.2, pp.317–337, 2000

[9] OMG 2009, "UML Profile for Modeling Quality of Service and Fault Tolerance Characteristics and Mechanisms", V-1.1

[10] Frank Alexander Kraemer, "ARCTIS", Department of Telematics, NTNU, http://arctis.item.ntnu.no, retrieved May 2011

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

[12] Frank Alexander Kraemer and Peter Herrmann, "Service specification by composition of collaborations-an example", Proceedings of Web Intelligence – Intelligent Agent Technology workshops, pp. 129-133, IEEE computer society, 2006

[13] Frank Alexander Kraemer and Peter Herrmann, “Formalizing Collaboration-Oriented Service Specifications Using Temporal Logic”, Proceedings of the International Conference on Networking and Electronic Commerce Research Conference, p. 194-220, ATSMA Inc, 2007

[14] Lamport, “Specifying Systems”, Addison-Wesley, 2002

[15] Frank Alexander Kraemer, Peter Herrmann, and R. Bræk, “Aligning UML 2 state machines and temporal logic for the effecient execution of services”, Published in On the Move to Meaningful Internet Systems: CoopIS, DOA, GADA, and ODBASE, pp. 1613-1632, Lecture Notes of Computer Science, Springer, 2006.

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

[17] Yuan Yu, Panagiotis Manolios, and Leslie Lamport, “Model Checking TLA+ Specifications”, In L. Pierre and T. Kropf, editors, Proceedings of the 10th IFIP WG 10.5 Advanced Research Working Conference on Correct Hardware Design and Verification Methods (CHARME’99), volume 1703 of Lecture Notes in Computer Science, pages 54–66. Springer-Verlag, 1999

[18] Frank Alexander Kraemer, Vidar Slåtten, and Peter Herrmann, “Engineering Support for UML Activities by Automated Model-Checking – An Example”, Proceedings of the 4th International Workshop on Rapid Integration of Software Engineering Techniques, p. 51-66, 2007

[19] Mate J. Csorba, Poul E. Heegaard, and Peter Herrmann, “Cost-Efficient Deployment of Collaborating Components”, Proceedings of the 8th IFIP International Conference on Distributed Applications and Interoperable Systems, pp. 253–268, Springer, 2008

[20] Razib Hayat Khan and Poul E. Heegaard, “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, Lecture Notes of Computer Science, Springer, 2011

[21] OMG 2005, “UML Profile for Schedulability, Performance, and Time Specification”, V – 1.1

[22] 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

[23] K. S. Trivedi, "Probability and Statistics with Reliability, Queuing and Computer Science application", Wiley- Interscience publication, ISBN 0-471-33341-7, 2nd Edition, 2001

[24] Pakke Kahkipru, "UML based performance modeling framework for object-oriented distributed systems", Proceedings of the 2nd international conference on the unified modeling language: beyond the standard, pp. 356-371, Springer-Verlag Berlin, Heidelberg, 1999

[25] Peter King and Rob Pooley, "Derivation of Petri Net Performance Models from UML Specifications of Communications Software", B.R. Haverkort et al. (Eds.): TOOLS, pp. 262-276, Springer-Verlag Berlin, Heidelberg, 2000

[26] John Anil Saldhana and Sol M. Shatz, "UML Diagrams to Object Petri Net Models: An Approach for Modeling and Analysis", Proceedings of the International Conference on Software Engineering and Knowledge Engineering, pp. 103-110, 2003

[27] A Al Abdullatif and Rob Pooley, "A Computer Assisted State Marking Method For Extracting Performance Models from Design Models", International journal of simulation, Vol. 8, No. 3, pp. 36-46, 2008

[28] Rob Pooley, "Using UML to Derive Stochastic Process Algebra Models", http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.47.1058,retrieved April, 2012

[29] Nico De Wet and Pieter Kritzinger, "Using UML models for the performance analysis of network systems", The International Journal of Computer and Telecommunications Networking - Telecommunications and UML languages, Vol. 49, No. 5, pp. 627-642, 2005

[30] Jasemin K.S and R. Vashanta, "Derivation of UML Based Performance Models for Design Assessment in a Reuse Based Software Development Approach", Annals. Computer Science Series, Vol. 7, No. 1, 2009

[31] Tom Verdickt, Bart Dohedt, and Frank Gielen, "Incorporating SPE into MDA: including middleware performance details into system models", Proceedings of the 4th international workshop on Software and performance, pp. 120-124, ACM, 2004

[32] Hassan Gomaa and Daniel A. Menascé, "Performance Engineering of Component-Based Distributed Software Systems", R. Dumke et al. (Eds.): Performance Engineering, pp. 44-55, Springer-Verlag Berlin Heidelberg, 2001

[33] Juan Pablo Lopez-Grao, Jose Merseguer, and Javier Campos, "From UML activity diagrams to Stochastic Petri nets: application to software performance

engineering", Proceedings of the 4th international workshop on Software and performance, pp. 25-36, ACM press, 2004

[34] José Merseguer and Javier Campos, "On the integration of UML and Petri nets in software development", Proceedings of the 27th international conference on Applications and Theory of Petri Nets and Other Models of Concurrency, pp. 19-36, Springer-Verlag Berlin, Heidelberg, 2006

[35] Paolo Bocciarelli and Andrea D'Ambrogio, "A model-driven method for describing and predicting the reliability of composite services", Journal of Software and Systems Modeling, Vol. 10, No. 2, pp. 265-280, Springer-Verlag New York, 2011

[36] Salvatore Distefano, Marco Scarpa, and Antonio Puliafito, "From UML to Petri Nets: The PCM-Based Methodology", IEEE Transactions on Software Engineering, Vol. 37, No. 1, pp. 65-79, IEEE press, 2011

[37] Murray Woodside, Dorina C. Petriu, Dorin B. Petriu, Hui Shen, Toqeer Israr, and Jose Merseguer, "Performance by Unified Model Analysis (PUMA)", Proceedings of the 5th international workshop on Software and performance, pp. 1-12, ACM press, 2005

[38] Andrea D'Ambrogio, "A model transformation framework for the automated building of performance models from UML models", Proceedings of the 5th international workshop on Software and performance, pp. 75-86, ACM press, 2005

[39] Salvatore Distefano, Daniele Paci, Antonio Puliafito, and Marco Scarpa, "UML Design and Software Performance Modeling", Proceedings of the 19th International symposium on Computer and Information Sciences, pp. 564-573, Springer-Verlag Berlin Heidelberg, 2004

[40] Juan Pablo Lopez-Grao , Jose Merseguer , and Javier Campos, "Performance Engineering based on UML and SPN's: A software performance tool", Proceedings of the 17th International symposium on Computer and Information Sciences, pp. 405-409, CRC press, 2002

[41] Birgit Korherr and Beate List, "Extending the UML 2 Activity Diagram with Business Process Goals and Performance Measures and the Mapping to BPEL", Proceedings of the 2006 international conference on Advances in Conceptual Modeling: theory and practice, pp. 7-18, Springer-Verlag, Heidelberg, 2006

[42] Mirco Tribastone and Stephen Gilmore, "Automatic Extraction of PEPA Performance Models from UML Activity Diagrams Annotated with the MARTE Profile", Proceedings of the 7th international workshop on Software and performance, pp. 67-78, ACM press, 2008

[43] Yosr Jarraya, Andrei Soeanu, Mourad Debbabi, and Fawzi Hassaine, "Automatic Verification and Performance Analysis of Time-Constrained SysML Activity Diagrams", Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems, pp. 523-530, IEEE computer society, 2007

[44] B. Bharathi and G. Kulanthaivel, "A tool for architectural design evaluations using simplistic approach", IJCA Special Issue on Computational Science - New Dimensions and Perspectives, Vol. 4, No. 7, pp. 162–165, Foundation of Computer Science, 2011

[45] Dorina C. Petriu, "Deriving Performance Models from UML Models by Graph Transformations", Tutorial in Workshop on Software and Performance, 2000

[46] Simonetta Balsamo and Moreno Marzolla, "Performance Evaluation of UML Software Architectures with Multiclass Queueing Network Models", Proceedings of the 5th international workshop on Software and performance, pp. 37-42, ACM press, 2005

[47] H. Motameni, A. Movaghar, and M. Fadavi Amiri, "Mapping Activity Diagram to Petri Net: Application of Markov Theory for Analyzing Non-functional Parameters", IJE transactions, Vol.20, No. 1, pp. 65-76, 2007

[48] H. Motameni, A. Movaghar, I. Daneshfar, H. Nemat Zadeh, and J. Bakhshi, "Mapping to Convert Activity Diagram in Fuzzy UML to Fuzzy Petri Net", World Applied Sciences Journal, Vol. 3, No. 3, pp. 514-521, IDOSI Publications, 2008

[49] Ali I. El-Desouky, Hesham A. Ali, and Yousry M. Abdul-Azeem, "LQN-Based Performance Evaluation Framework of UML-Based Models for Distributed Object Applications", Proceeding of the INFOS, pp. 11-19, Cairo University press, 2008

[50] Michele Angelaccio and Andrea D'Ambrogio, "A Model-driven Framework for Managing the QoS of Collaborative P2P Service-based Applications", Proceedings of the 15th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises, pp. 95-102, IEEE Computer Society, 2006

[51] D. Kreische, "Performance and Dependability in Business Process Modeling", http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.11.5875, retrieved April 2012

[52] Antonio García-Domínguez and Inmaculada Medina-Bulo, "Model-Driven Design of Performance Requirements with UML and MARTE", Proceedings of ICOSOFT, pp. 54-63, SciTePress, 2011

[53] Tero Arpinen, Erno Salminen, Timo D. Hamalainen, and Marko Hannikainen, "Performance Evaluation of UML2-Modeled Embedded Streaming Applications with System-Level Simulation", EURASIP Journal on Embedded Systems, Vol. 2009, Article ID 826296, pp. 1-16, Hindawi Publishing Corporation, 2009

[54] Aad P. A., Van Moorsel, and Yiqing Huang, "Reusable Software Components for Performability Tools, and Their Utilization for Web-based Configurable Tools", http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.1.8730, retrieved April 2012, Springer, 1998

[55] Xiuping Wu and Murray Woodside, "Performance modeling from software components", Proceedings of the 4th international workshop on Software and performance, pp. 290-301, ACM press, 2004

[56] T. Kappler, H. Koziolek, K. Krogmann, and R. H. Reussner, "Towards Automatic Construction of Reusable Prediction Models for Component-Based Performance Engineering", Proceedings of Software Engineering, ser. LNI, K. Herrmann and B. Brügge, editors, vol. 121, pp. 140–154, 2008

[57] Shige Wang and Kang G. Shin, "Early-stage performance modeling and its application for integrated embedded control software design", Proceedings of the 4th international workshop on Software and performance, pp. 110-114, ACM press, 2004

[58] B.Bharathi and G.Kulanthaivel, "Towards Developing A Performance Evaluator for Component Based Software Architectures", Indian Journal of Computer Science and Engineering, Vol. 2, No. 1, pp. 136 – 142, 2011

[59] Rasha Tawhid and Dorina C. Petriu, "Towards automatic derivation of a product performance model from a UML software product line model", Proceedings of the 7th international workshop on Software and performance, pp. 91-102, ACM press, 2008

[60] Van Bui, Boyana Norris, Lois Curfman McInnes, Li Li, Oscar Hernandez, and Barbara Chapman, "A component infrastructure for performance and power modeling of parallel scientific applications", Proceedings of the 2008 compFrame/HPC-GECO workshop on Component based high performance, ACM press, 2008

[61] Bernd Hardung, Thorsten Kölzow, and Andreas Krüger, "Reuse of Software in Distributed Embedded Automotive Systems", Proceeding of the EMOSOFT, pp. 203-210, ACM press, 2004

[62] Shige Wang and Kang G. Shin, "Early-stage performance modeling and its application for integrated embedded control software design", Proceedings of the

4th international workshop on Software and performance, pp. 110-114, ACM press, 2004

[63] José Merseguer, “Web services UML modeling”, Tutorial on Web Services: Architecture, Concepts and Standards, University of Zaragoza

[64] A Al Abdullatif and Rob Pooley, “UML to EQN: Studying System Performance from an Early Stage of Systems Life Cycle”, Proceedings of the 25th UK Performance Engineering, pp. 111-122, 2009

[65] Hoda Amer and Dorina C. Petriu, “Software Performance Evaluation: Graph Grammar-based Transformation of UML Design Models into Performance Models”, Proceedings of the 12th International conference, Tools, LNCS, Springer-Verlag Berlin. 2002

[66] S. Balsamo, M. Marzolla, and R. Mirandola "Efficient Performance models in Component-Based Software Engineering", Proccedings of the 32nd Euromicro Conference on Software Engineering and Advanced Applications (SEAA), Component-Based Software Engineering track, pp. 64-71, IEEE computer society, 2006

[67] Hui Shen and Dorina C. Petriu, “Performance analysis of UML models using aspect-oriented modeling techniques”, Proceedings of the 8th international conference on Model Driven Engineering Languages and Systems, pp. 156-170, Springer-Verlag Berlin, Heidelberg, 2005

[68] F. Duarte, W. Hasling, R. Leao, E. Silva, and V. Cortellessa, “Extending model transformations in the performance domain with a node modeling library”, Proceedings of the 7th international workshop on Software and performance, pp. 157-164, ACM press, 2008

[69] D. Mania and J. Murphy, “Framework for predicting performance of Component based Systems”, Proceedings of the 7th International Conference on Software, Telecommunications and Computer Networks, IEEE computer society, 2002

[70] Yong Zhang, Ningjiang Chen, Jun Wei, and Tao Huang, “Completing UML model of component-based system with middleware for performance evaluation”, Proceedings of the 2006 international conference on Emerging Directions in Embedded and Ubiquitous Computing, pp. 72-82, Springer-Verlag Berlin, Heidelberg, 2006

[71] Elena Gomez-Martinez and Jose Merseguer, “Performance Modeling and Analysis of the Universal Control Hub”, Proceedings of the 7th European Performance Engineering Workshop, pp. 160-174, Springer-Verlag Berlin, Heidelberg, 2010

[72] Salvatore Distefano, Antonio Puliafito, Marco scarpa, Salvatore Distefano, Antonio Puliafito, and Marco Scarpa, “Implementation of the Software Performance Engineering Development Process”, Journal of Software, Vol. 5, No. 8, pp. 872-882, Academy Publisher

[73] Joel Wright, John Ibbotson, Christopher Gibson, Dave Braines, Thomas Klapiscak, Sahin Geyik, Boleslaw Szymanski, and David Thornley, “A Model-Driven Approach to the Construction, Composition and Analysis of Services on Sensor Networks”, Proceedings of the 3rd annual conference of international technology aliance, pp. 1-10, 2010

[74] James Skene and Wolfgang Emmerich, “Model Driven Performance Analysis of Enterprise Information Systems”, Electronic Notes in Theoretical Computer Science, Vol. 82, No. 6, Elsevier Science B.V., 2003

[75] M. Nikolaidou, N. Alexopoulou, A. Tsadimas, A. Dais, and D. Anagnostopoulos1, “Using UML to Model Distributed System Architectures”, Proceedings of the 18th International Conference on Computer Applications in Industry and Engineering, 2005

[76] John Hakansson, Leonid Mokrushin, Paul Pettersson, and Wang Yi, “An Analysis Tool for UML Models with SPT Annotations”, In online proceedings of International Workshop on Specification and Validation of UML models for Real Time and Embedded Systems (SVERTS), 2004

[77] Petri Kukkala, Marko Hannikainen, and Timo D. Hamalainen, “Performance Modeling and Reporting for the UML 2.0 Design of Embedded Systems”, Proceedings of the international symposium on System-on-Chip, pp. 50-53, IEEE computer society, 2005

[78] A. J. Bennett, A. J. Field, and C. M. Woodside, “Experimental evaluation of the UML profile for schedulability, performance, and times”, Lecture Notes in Computer Science, Vol. 3273, pp. 143-157, Springer, 2004

[79] Dorina C. Petriu, C.M. Woodside, D.B. Petriu, J. Xu, T. Israr, Geri Georg, Robert France, James M. Bieman, Siv Hilde Houmb, and Jan Jürjens, “Performance analysis of security aspects in UML models”, Proceedings of the 6th international workshop on Software and performance , pp. 91-102, ACM press, 2007

[80] Elena Gomez-Martinez and Jose Merseguer, “A Software Performance Engineering Tool based on the UML-SPT” Proceedings of the Second International Conference on the Quantitative Evaluation of Systems, pp. 247-248, IEEE computer society, 2007

[81] Vittorio Cortellessa, Antinisca Di Marco, Romina Eramo, Alfonso Pierantonio, and Catia Trubiani, “Digging into UML models to remove Performance

Antipatterns", Proceedings of the 2010 ICSE Workshop on Quantitative Stochastic Models in the Verification and Design of Software Systems, pp.9-16, ACM press, 2010

[82] H. Du, R. Gan, K. Liu, Z. Zhang, and D. Booy, "Method for Constructing Performance Annotation Model Based on Architecture Design of Information Systems", Proceeding of the International Conference on Research and Practical Issues of Enterprise Information Systems II, Vo. 2, pp. 1179-1189, 2007

[83] Dorina C. Petriu and Hui Shen, "Applying the UML Performance Profile: Graph Grammar-based Derivation of LQN Models from UML Specifications", Proceedings of the Tools, pp. 159—177, Springer-Verlag, 2002

[84] Hui Liu, "Transformation of UML 2.0 models extended with MARTE to core scenario models", M.Sc. Thesis, University of Carleton, 2008

[85] Jouni Riihimäki, Petri Kukkala, Tero Kangas, Marko Hännikäinen, and Timo D. Hämäläinen, "Interfacing UML 2.0 for Multiprocessor System-on-Chip Design Flow" Proceedings of the international symposium on System-on-Chip, pp. 108-111, IEEE computer society, 2005

[86] Simona Bernardi and José Merseguer, "QoS assessment via stochastic analysis", Proceedings of the IEEE Internet Computing, pp. 32-42, IEEE computer society, 2006

[87] Antonio García-Domínguez, Inmaculada Medina-Bulo, and Mariano Marcos-Bárcena, "Model-Driven Design of Performance Requirements with UML and MARTE", Proceedings of the 6th International Conference on Software and Data Technologies, 2011

[88] Elena Gomez-Martinez and Jose Merseguer, "ArgoSPE: Model-Based Software Performance Engineering", Proceedings of the 27th International Conference on Applications and Theory of Petri Nets and Other Models of Concurrency, pp. 401-410, LNCS, Springer-Verlag Berlin Heidelberg, 2006

[89] Reheb A. El-kaedy and Ahmed Sameh, "Performance Analysis and Characterization Tool for Distributed Software Development", International Journal of Research and Reviews in Computer Science, Vol. 2, No. 3, pp. 906-915, 2011

[90] Kendra Cooper, Lirong Dai, and Yi Deng, "Modeling Performance as an Aspect: a UML Based Approach", Proceedings of the 4th AOSD Modeling with UML Workshop, 2003

[91] Catherine Canevet, Stephen Gilmore, Jane Hillston, and Perdita Stevens, "Performance modeling with UML and stochastic process algebras", IEEE Proceedings: Computers and Digital Techniques, pp. 107-120, 2003

[92] P. Moura, R. Borges, and A. Mota, “Experimenting Formal Methods through UML”, Proceedings of the Brazilian Workshop on Formal Methods, 2003

[93] J. Warmer and A. Kleppe. “The Object Constraint Language: Precise Modeling with UML”, Addison-Wesley, 1999

[94] B.D. Theelen, P.H.A. van der Putten, and J.P.M. Voeten, “Using the SHE Method for UML-based Performance Modeling”, System Specification and Design Languages, pp. 143-160, 2004

[95] Yongfeng Yin, Bin Liu, Zhen Li, Chun Zhang, and Ning Wu, “The Integrated Application Based on Real-time Extended UML and Improved Formal Method in Real-time Embedded Software Testing” Journal of Networks, Vol. 5, No. 12, pp. 1410-1416, 2010

[96] Li Shuhao, Wang Ji, Dong Wei, and Qi Zhichang, “A framework of property-oriented testing of reactive systems”, Chinese Journal of Electronics, Vol.32, No.12A, pp.222-225, 2004

[97] Andolfi, F. Aquilani, S. Balsamo, and P. Inverardi, “Deriving performance models of software architectures from message sequence charts. In Proc. of 2nd International Workshop on Software and Performance, pages 47-57, ACM Press, 2000

[98] J. Cardoso and C. Sibertin-Blanc, “Ordering actions in sequence diagrams of UML”, Proceedings of 23rd International Conference on Information Technology Interfaces (ITI2001), 2001

[99] D. Latella, I. Majzik, and M. Massink, “Towards a formal operational semantics of UML statechart diagrams”, Proceedings of 3rd Int. Conference on Formal Methods for Open Object-Based Distributed Systems, pp. 331-347, Kluwer, 1999

[100] J. Lilius and I.P. Paltor, “The semantics of UML state machines”, Technical report no.273 - Turku Centre for Computer Science, Finland, May 1999

[101] A. J. H. Simons, “On the compositional properties of UML statechart diagrams”, Proceedings of the Rigorous Object-Oriented Methods, 2000

[102] A. Tsiolakis, “Integrating model information in UML sequence diagrams”, Proceedings of 2nd International Workshop on Graph Transformation and Visual Modeling Techniques, Electronic Notes in Theoretical Computer Science, Springer-Verlag, 2001.

[103] Rik Eshuis and Roel Wieringa, “A Real-Time Execution Semantics for UML Activity Diagrams”, Proceedings of the 4th International Conference on

Fundamental Approaches to Software Engineering, pp. 76-90, Springer-Verlag London, 2001

[104] C. Bolton and J. Davies, "Activity graphs and processes" Proceedings of the Integrated Formal Methods , LNCS 1945, Springer, 2000

[105] T. Gehrke, U. Goltz, and H. Wehrheim, "The dynamic models of UML: Towards a semantics and its application in the development process", Hildesheimer Informatik-Bericht 11/98, 1998

[106] E. Borger, A. Cavarra, and E. Riccobene, "An ASM Semantics for UML Activity Diagrams", In T. Rus, editor, Proceedings of the Algebraic Methodology and Software Technology, 8th International Conference, AMAST 2000, LNCS 1826. Springer, 2000

[107] W. M .P . Van Der Aalst, "The application of Petri nets to work flow management", The Journal of Circuits, Systems and Computers, Vol.8, No.1,pp. 21-66, 1998

[108] Dirk Wodtke and Gerhard Weikum, "A Formal Foundation for Distributed Workflow Execution Based on State Charts", Proceedings of the 6th International Conference on Database Theory, pp. 230-246, Springer-Verlag London,1997

[109] David Harel and Amnon Naamad, "The STATEMATE Semantics of Statecharts", ACM Transactions on Software Engineering and Methodology, Vol. 5, No. 4, pp. 293–333, 1996

[110] J. Trowitzsch and A. Zimmermann, "Using UML State Machine and Petri Nets for the Quantitative Investigation of ETCS", Proceedings of the 1st international conference on Performance evaluation methodolgies and tools, ACM press, 2006

[111] A. Zimmermann and Gunter Hommel, "Towards modeling and evaluation of ETCS real-time communication and operation", The Journal of Systems and Software, Vol.75, pp. 47-54, Elsevier, 2005

[112] H. Hermanns, D. N. Jansen, and Y. Usenko, "From stocharts to modest: a comparative reliability analysis of train radio communications", Proceedings of the 5th international workshop on Software and performance, pages 13–23, ACM Press, 2005

[113] C. Lindemann, A. Thummler, A. Klemm, M. Lohmann, and O. Waldhorst, "Performance Analysis of Time-enhanced UML Diagrams Based on Stochastic Processes", Proceedings of the 3rd Workshop on Software and Performance, pages 25–34, ACM press, 2002

[114] Valentina Boschian, Mariagrazia Dotoli, Maria Pia Fanti, Giorgio Iacobellis, and Walter Ukovich, “A metamodeling approach for performance evaluation of inter-modal transportation networks”, European Transport \ Trasporti Europei, Vol. 46 pp. 100-113, 2010

[115] L.B. Arief and N.A. Speirs, “A UML Tool for an Automatic Generation of Simulation Programs”, Proceedings of the 2nd international workshop on Software and performance, pp. 71-76, ACM press, 2000

[116] N. Sato and K. S. Trivedi, “Stochastic Modeling of Composite Web Services for Closed-Form Analysis of Their Performance and Reliability Bottlenecks”, Proceedings of the International Conference on Service Oriented Computing, pp. 107-118, Springer, 2007

[117] P. Bracchi, B. Cukic, and Cortellesa, “Performability modeling of mobile software systems”, Proceedings of the International Symposium on Software Reliability Engineering, pp. 77-84, 2004

[118] 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

[119] Gonczy, Deri, and Varro, “Model Driven Performability Analysis of Service Configurations with Reliable Messaging”, Proceedings of the Model-Driven Web Engineering Workshop, 2008

[120] Aad P. A. Van Moorsel, and Boudewijn R. Haverkort, “A Unified Performability Evaluation Framework for Computer and Communication Systems”, Proceedings of the second International Workshop on Performability Modeling of Computer and Communication Systems, INRIA Rennes, 1993

[121] S. Dalibor, A. Hein, and W. Hohl, “Application Dependent Performability Evaluation of Fault-Tolerant Multiprocessors”, Proceedings of the 4th Euromicro Workshop on Parallel and Distributed Processing, pp. 310-318, IEEE computer society, 1996

[122] John F. Meyer, “On Evaluating the Performability of Degradable Computing Systems ”, IEEE Transactions on Computers, Vol. 29, No. 8, pp. 720–731, 1980

[123] Yijie Hana, J. Richard Laa, M. Armand Makowskia, Seungjoon Lee, "Distribution of path durations in mobile ad-hoc networks—Palm’s Theorem to the rescue", Computer Networks, Vol. 50, No. 12, pp. 1887–1900, 2006

Part II

Included Papers

Paper 1

Translation from UML to Markov model: A performance modeling framework

Razib Hayat Khan, Poul E. Heegaard

Presented in International Joint Conferences on Computer, Information, and System Sciences, and Engineering (CISSE), 2009, Bridgeport, USA

Published in Innovations in Computer Sciences and Engineering, p. 365-371, Springer Science+Business Media V.B. 2010

Translation from UML to Markov model: A performance modeling framework

Razib Hayat Khan, Poul E. Heegaard

Department of Telematics
Norwegian University of Science and Technology (NTNU)
Trondheim, Norway
{rkhan, poul.heegaard}@item.ntnu.no

Abstract- Performance engineering focuses on the quantitative investigation of the behavior of a software system during the early phase of the system development life cycle. Bearing this concept, we delineate a performance modeling framework of the application for communication system that proposes a translation process from high level UML notation to Continuous Time Markov Chain model (CTMC) and solves the model for relevant performance metrics. The framework utilizes UML collaborations, activity diagrams and deployment diagrams to be used for generating performance model for a communication system. The system dynamics will be captured by UML collaboration and activity diagram as reusable specification building blocks, while deployment diagram highlights the components of the system. The collaboration and activity show how reusable building blocks in the form of collaboration can compose together the service components through input and output pin by highlighting the behavior of the components and later on, a mapping between collaboration and system component identified by UML deployment diagram will be delineated. Moreover, the UML models are annotated to associate performance related quality of service (QoS) information, which is necessary for solving the performance model for relevant performance metrics through our proposed framework. The applicability of our proposed performance modeling framework is delineated in the context of modeling a communication system.

1 Introduction

Communication systems are complex systems. To meet functional requirements are obviously important while designing applications for communication system, but they are not the only concern. Performance evaluation to meet user requirement is another important factor. Performance evaluation is the degree to which a system meets its objectives and satisfies user expectation, which is important in many cases and is critical in some real-time applications. It is necessary to take into account the performance issues earlier in the system development lifecycle and treating these as an essential part of the system development process. Therefore, finding a way to extract performance model from design model at early stage of system development process and solves the model for relevant performance metrics is a key issue in the perspective of system performance engineering. So the developers will be able to make informed decisions within the design process as well as readily explore 'what-if' scenarios and assessing the implication of changing logic in execution of application by considering all the dynamics, interaction among the system components as well as considering the system's execution environment (i.e. the deployment of network resources, network technology and network topology) and workload factors of the system, which all have greater impact on a system's performance. To consider all the above issues, our proposed framework utilizes UML collaboration [1], activity [1] and deployment diagram [1] as UML is the most widely used modeling language, which models both the system requirements and qualitative behavior through different notations. Collaboration and activity diagram will be specified to capture system dynamics and interaction among service components as reusable specification building blocks [2] by highlighting component's behavior. To compose the overall activity of the system in the form of collaboration events identified as input and output pins on the activities are connected together [2]. Deployment diagram will identify the system components, the process executes on the each component as well as considers the execution platform and network topology of the system. A mapping is delineated between system components and collaborations thereafter to show how the service is defined by the joint behavior of the system components. Moreover, the UML models are annotated incorporating performance related information. By the above specification style of UML, probable states and the performance parameters for triggering the change of the states of the performance model will be generated and solved by our proposed performance modeling framework.

Markov model [3], queuing network [3] and stochastic petrinet [3] are probably the best studied performance modeling techniques. Among all of them, we will choose Markov model as the performance model generated by our proposed framework due to its modeling generality, its well-developed numerical modeling analysis techniques, its ability to preserve the original architecture of the system, to facilitate any modification according to the feedback from performance evaluation and the existence of analysis tools.

The objective of this paper is to provide an extensive performance modeling framework that provides a translation process to generate performance model from system specification description captured by the UML behavioral diagram [1] and solves the model for relevant performance metrics at the early stage of system development life cycle. The rest of this paper is organized as follows: Section 2 focuses on related works, Section 3 presents our approach of specifying UML technique for performance modeling, Section 4 describes our performance modeling framework and Section 5 delineates the conclusion with work.

2 Related works

Related work includes a number of efforts are made generating a performance model from the system specification. Kähkipuro developed a performance modeling framework to generate queuing network with simultaneous resource possessions from the high level UML notations so that model can be solved for the relevant performance metrics [4]. Lopez-Grao *et al.* proposed a conversion method from annotated UML activity diagram to stochastic petrinet model [5]. Trowitzsch and Zimmermann proposed the modeling of technical systems and their behavior by means of UML and for the resulting models a transformation into a Stochastic Petri Net was established [6]. Abdullatif and Pooly presented a method for providing computer support for extracting Markov chains from a performance annotated UML sequence diagram [7]. The framework in this paper is the first known approach that introduces a new specification style utilizing UML behavioral diagrams as reusable specification building block that is used for generating performance model. The main focus here is to introduce reusable building blocks, from which systems can be composed and performance model will be generated. These building blocks are collaborations*,* which mean that one building block describes the behavior of several system components. This makes it easier to reuse complete services, since all interactions necessary to coordinate behavior among components can be encapsulated [8].

3 UML technique of performance modeling

In this paper, we utilize UML 2.2 collaboration, activity and deployment diagram to be used for generating performance model from system design specification. We outline a specification style using UML 2.2 collaboration and activity diagram, which is the part of the tool suite Arctis [8]. Arctis focuses on the Collaboration and activity as reusable specification building blocks describing the interaction between system components as well as internal behavior of the components [2]. To mention the overall behavior of the system by composing the reusable building blocks the events are identified as input and output pins on the activities that are connected together [2]. Deployment diagram is another integral part of our proposed framework that specifies a set of constructs that can be used to define the execution architecture of the systems that represent the assignment of software artifacts to the system components or physical nodes [1]. As an example, we utilize a system description, where users are equipped with cell phone or PDA want to receive weather information of the current location using his/her hand held devices. The user request is first transferred to the location server through base transceiver station to retrieve location information of the user. The location information is then transferred to weather server for retrieving the weather information according to the location of the user.

Figure 1 shows the UML collaboration, which focuses on the formulation of building block declaring the participants as collaboration role and connection between them [2]. User service request is generated from user's hand held devices. The users are part of the environment and therefore labeled as <<external>>. User service request is transferred between the mobile terminal (MT) and base transceiver station (BTS) is highlighted by collaboration *t*. BTS interacts with the location server (LS) for retrieving user location information by using collaboration *l*. The LS retrieves the desired information from databases (DB) using collaboration use *d1*. Then BTS interacts with the weather server (WS) for weather information by using collaboration *w* according to the location information of user supplied by the LS. The WS retrieves the desired information from DB using collaboration use *d2*.

While UML collaboration describes the structural aspect of the composed service the internal behavior of the collaboration is described by the UML activity [2]. Hereby, collaborations of Figure 1 are modeled by a call behavior action referring to the activity [9]. To deliver the requested information to the user through his/here mobile terminal, BTS participates in the collaboration *Request Location Info* together with the LS and *Request Weather info* together with WS. These are specified by the collaboration *l: Request Location Info* and *w: Request Weather info,* where the BTS plays the role client and the LS and WS play the role server. The behavior of the collaboration is described by the UML activity in Figure 3, where activity is divided into two partition one for each collaboration role (Client and Server) [2]. The activity is started on the client side, when the user request is provided as parameter *u_req* at the input pin. The *u_req* directly sent to the LS, where it is converted into a database request by the call operation action *processing.*

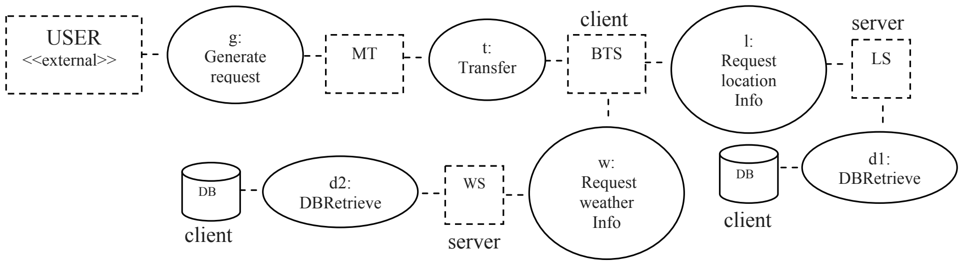


Figure 1. Collaboration diagram

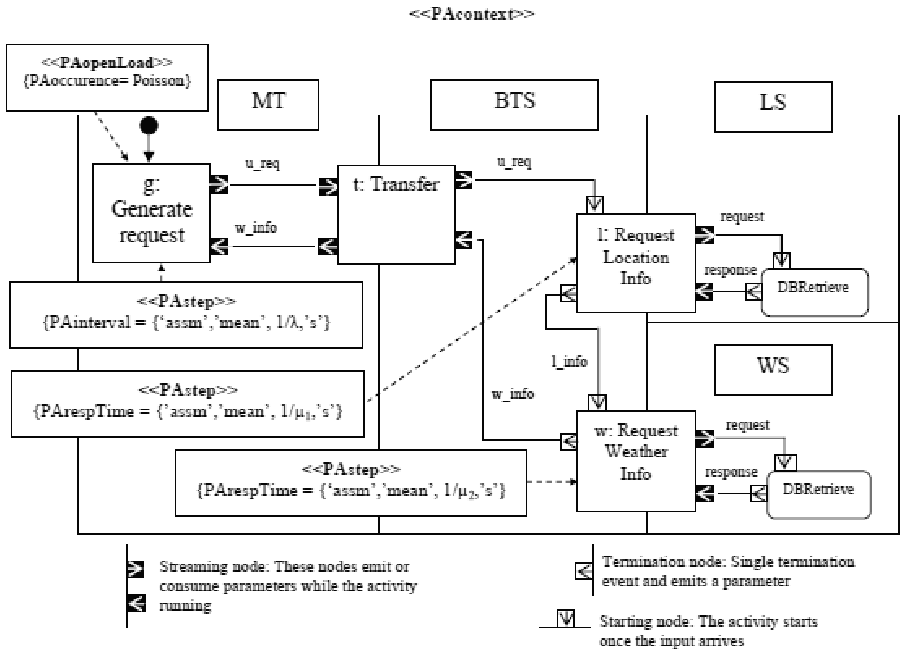

<<PAcontext>>
<<PAopenLoad>>
{PAoccurence= Poisson}
MT
BTS
LS
g: Generate request
u_req
w_info
t: Transfer
u_req
l: Request Location Info
request
response
DBRetrieve
<<PAstep>>
{PAinterval = {'assm','mean', 1/λ,'s'}
l_info
w_info
WS
<<PAstep>>
{PArespTime = {'assm','mean', 1/μ1,'s'}
w: Request Weather Info
request
response
DBRetrieve
<<PAstep>>
{PArespTime = {'assm','mean', 1/μ2,'s'}
Streaming node: These nodes emit or consume parameters while the activity running
Termination node: Single termination event and emits a parameter
Starting node: The activity starts once the input arrives

Figure 2. System activity to couple the collaboration

After that, it is the task of the collaboration between the server and the database to provide the stored information. To get the information the request leaves the activity *Request Location info* and the server waits for the reception of response. This is modeled with the input and output pins *request* and *response*. After getting the response, the result is delivered to the corresponding output pin in the client side and the activity is terminated. Here, we describe the behavior of collaboration *Request Location info.* Likewise, we can describe the behavior of *Request Weather info* through activity partition of client and server, where location information of user is forwarded by the client to request server for retrieving the weather information of that particular user location.

We use activity in Figure 2 to describe how the events of the individual collaborations between the system components are coupled with each other so that the desired overall system behavior is obtained [2]. The initial node (●) marks the starting of the activities. The activity is started on the client side. When a user service request is generated via MT, *g: Generate request* will transfer the user service request as parameter *u_req* to the BTS via collaboration *t: Transfer*. Once arrived at the BTS, request for location information is forwarded to the LS represented by activity *Request location info*. LS makes a database request, which is modeled by *d1: DBRetrieve* and terminates with result *l_info* (Location information). After getting the location information, request for weather information according to user current location is forwarded by the BTS to the WS represented by activity *Request weather info.* WS makes a database request, which is modeled by *d2: DBRetrieve* and terminates with result *w_info* (Weather information). After that, the final result is transferred to the user hand held device by BTS via collaboration *t: Transfer.* The structure of collaborations as well as the way to couple them facilitates the reuse of activities. For example, both the collaboration *d1* and *d2* are identical and can be instantiated from single collaboration type. Moreover, the collaboration *l* and *w* have very similar behavior and can be based on the same UML template. Thus, systems of a specific domain can often be composed of reoccurring building blocks by reusing them [2].

The deployment diagram of the overall system is shown in Figure 4 highlighting the physical resources of our system such as mobile terminal, base transceiver station, location server, weather server. Service request is deployed on the user's mobile terminal, which is then transferred by the base transceiver station to the location server, where process for retrieving the location information of user is deployed. After that, process for retrieving the weather information of the user location is deployed in the weather server.

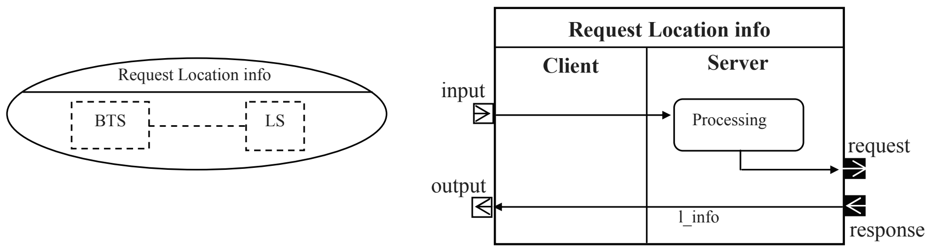


Figure 3. Structure (UML collaboration), Internal behavior (UML activity)

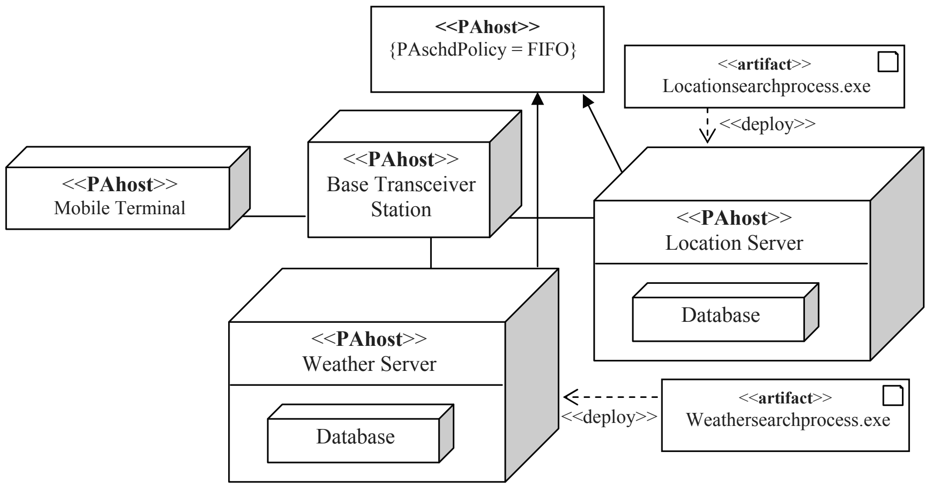


Figure 4. UML deployment diagram

4 Steps for building and evaluating the performance model (CTMC) from proposed modeling framework

Here, we describe how performance model will be generated and evaluated by our proposed framework shown in Figure 9 by utilizing the above specification style of UML. Steps 1 and 2 are the parts of the tool suite Arctis [8] and other steps are the extensions we needed generating the performance model by our proposed framework. The steps are as follows:

1) Construction of collaborative building block: This step defines the formulation of the building blocks in form of collaboration as major specification unit of our framework shown in Figure 1. The structure of the building block is defined by the UML collaboration shown in Figure 3. The building block declares the participants as collaboration role and connection between them. The internal behavior of the building block is described by a UML activity shown in Figure 3. It is declared as the classifier behavior of the collaboration and has one activity partition for each collaboration role in the structural description.

2) Composition of building blocks: For composition of building blocks, UML collaboration and activities are used complementary to each other. UML collaborations alone do not specify any behavior but only show how functionalities may be decomposed. Therefore, a UML activity is attached to a UML collaboration, which focuses on the behavior of collaborations as well as how behaviors of subordinate collaboration are composed. The activity in Figure 2 and the collaboration in Figure 1 show how reusable specification building blocks in form of collaboration can be composed.

3) Designing the deployment diagram and stating relation between system component and collaboration: Developing deployment diagram can be used to define the execution architecture of systems by identifying the system components and the assignment of software artifacts to those identified system components [1]. For our defined scenario the identified system components are mobile terminal, base transceiver station, location server and weather server shown in Figure 4. The artifact locationsearchprocess.exe is deployed on the location server and artifact weathersearchprocess.exe is deployed on the weather server. In our mentioned scenario, we consider single instance of location server and weather server.

After designing the deployment diagram the relation between system components and collaborations will be delineated describing the service delivered by the system. The service is delivered by the joint behavior of the system components, which may be physically distributed. The partial behavior of the component utilized to realize the collaboration is represented by the collaboration role [10]. In our scenario description, identified system components are mobile terminal, base transceiver station, location server, weather server. The behavior of the components mobile terminal, base transceiver station, location server, weather server is represented by collaboration roles MT, BTS, LS and WS to utilize the collaboration *t: transfer, l: request location info, w: request weather info*. Here, it is a one to one mapping between the system components and collaboration roles shown in Figure 5.

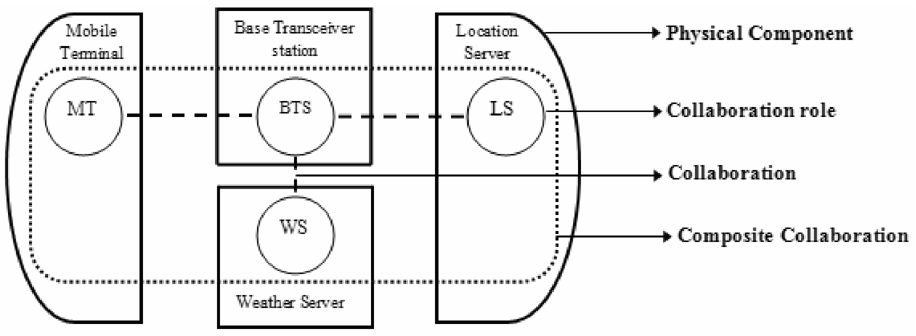


Figure 5. Relation between system components and collaborations

4) Annotation of source models: Performance information is incorporated into the UML activity diagram in Figure 2 and deployment diagram in Figure 4 according to the *UML Profile for Schedulability, Performance and Time* [11] to enable system performance to be evaluated by performance model solver for relevant performance metrics through our proposed framework. We use the stereotypes PAcontext, PAopenLoad, PAhost, PAstep and the tagged values PAoccurence, PAschdPolicy, PArespTime and PAinterval. A PAcontext models a performance analysis context. A PAopenLoad is modeled as a stream of requests that arrive at a given rate in predetermined pattern with PAoccurence. A PAhost models a processing resource with tagged PAschdPolicy defining the policy by which access to the resource is controlled. A PAstep models a scenario step with tagged

PArespTime defining a step's response time and PAinterval defines time interval between successive repetitions of a step.

5) State marking and Reachability graph: Here, we will describe how the probable states of the performance model will be generated. While generating the probable states for our performance model we consider only those participants, which have greater impact on the system performance and the states of the system will be generated based on the status of these participants shown by their internal behavior. For our example scenario, we will consider the participants location server and weather server as limiting factor for the system performance and the performance model states will be generated from location server and weather server status. The status of these servers is defined by their internal behavior through collaboration *request location info* and *request weather info*. The status of the both the servers are defined as idle, processing. When a step (annotated as <<PAstep>> in Figure 2) will be executed the status of the servers will be marked as performance model state as a whole from where a new state may be generated with a transition rate or return back to a already marked state with a transition rate mentioned in the annotated UML model in Figure 2. The states of the performance model are shown in Table 1 based on the status of both the servers as a whole. The states are: (idle, idle), (processing, idle), (idle, processing), (processing, processing), where the first part defines the status of the location server and second part defines the status of the weather server. If we assume initial marking such as the status of the location server and weather server is idle that means participants have no user request to process then we can derive all the reachable markings of the performance model from the initial marking. This can be done according to the arrival or departure of requests by following the interaction among the participants shown in the composition of building block through UML activity diagram in Figure 2. If we now construct the reachability graph with each of this reachable marking as a node and each edge between the nodes leveled with the trigger of their change by transition rate, we have a state transition diagram of the performance model. Here, if we assume system is stable, both servers buffer capacity are null and servers can process one request at a time, the state transition diagram is shown in Figure 6, where system states are generated from location server and weather server status shown in Table 1, where idle means no job is in the servers to process and processing means 1 job (number of job in the location server and weather server is mentioned by N and M and here, highest value

Table 1. states of performance model based on the status of location server & weather server

Location Server	Weather Server
Idle	Idle
processing	Idle
Idle	Processing
processing	Processing

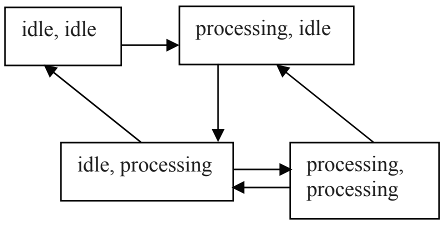


Figure 6. State transition diagram of the Markov model when number of request or job arrived and serviced by the system is 1

of N=M=1 in Figure 6) is processed by the servers. if we assume the system is stable, both servers buffer capacity is infinite, follow Poisson arrival pattern and FIFO (First In First Out) scheduling policy and servers can process one request at a time, the state transition diagram is shown in Figure 7 (where (N, M) >1 to infinity), which shows more states than the states generated from the status of both the servers. So if N=M=1 then the state transition diagram will be mentioned in Figure 6, which just reflect the internal behavior of the servers showing the change of system states mentioned in Table 1. If (N, M) > 1 the state transition diagram will be mentioned in Figure 7 highlighting more additional states including the states shown in Table 1 and Figure 6. The states will be marked by the total number of job N and M in the server, where 1 job will be processed by the servers and other remaining N-1 and M-1 job will be waiting in the buffer of location server and weather server for being processed. The generalized state transition diagram of our performance model is shown in Figure 8 including the boundary sates.

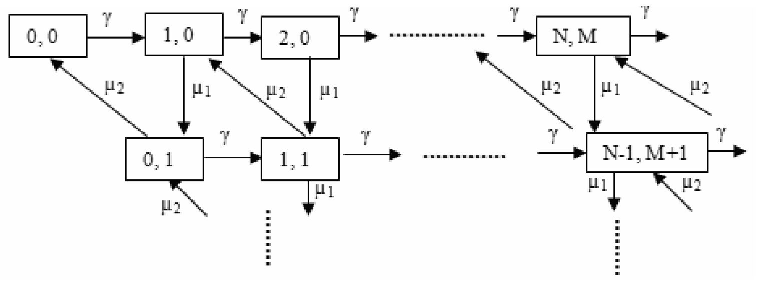


Figure 7. state transition diagram of the Markov model

6) Generating the performance model: From the mentioned reachability graph and annotated UML structure probable states (based on the value of N and M) and transition rate of the trigger of the change between states will be found based on which performance model will be generated, which can further be used as the input for the performance model solver.

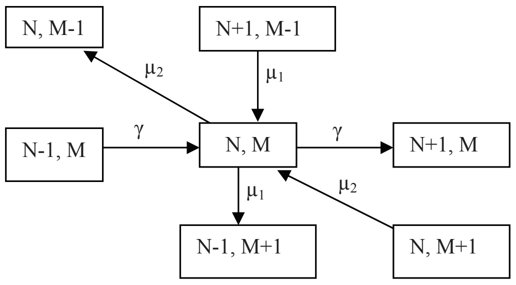


Figure 8. Generalized state transition diagram of the Markov model

7) Solving the performance model: The generated performance model will be solved by the SHARPE [12] performance model solver to generate performance results. Some of the performance results generated by the tools have been shown in the graph form in Figure 10.

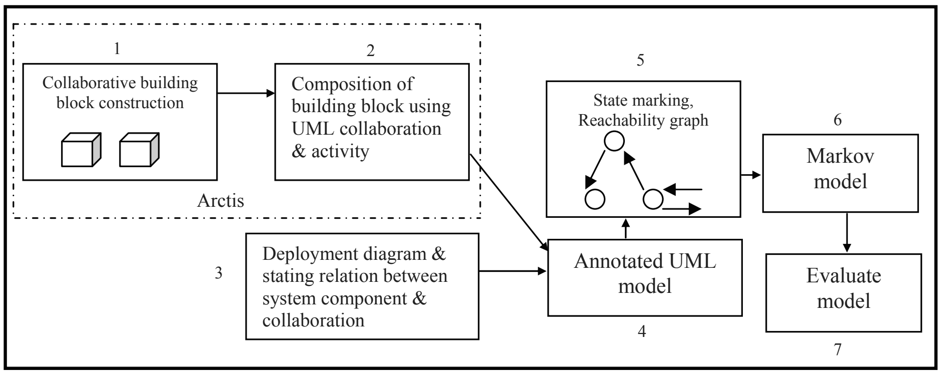

1
Collaborative building block construction
2
Composition of building block using UML collaboration & activity
Arctis
3
Deployment diagram & stating relation between system component & collaboration
5
State marking, Reachability graph
6
Markov model
Annotated UML model
4
Evaluate model
7

Figure 9. Performance modeling framework

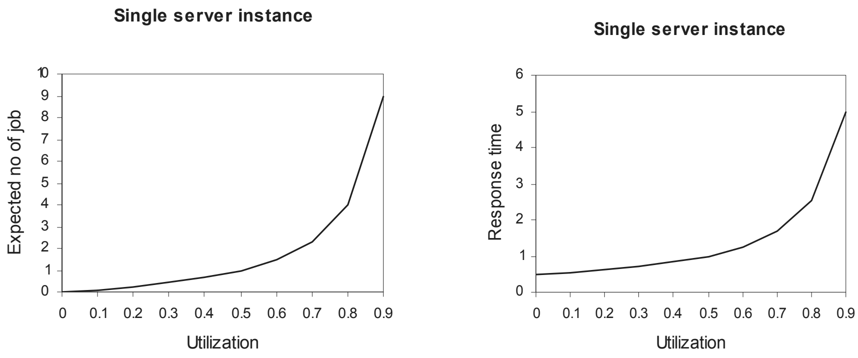


Figure 10. Expected number of jobs & average response time (sec)

5 Conclusion

In this paper, our main contribution is delineated as presenting a performance modeling framework of a software system by introducing a new specification style utilizing UML collaboration and activity diagram as reusable specification building blocks to capture the system dynamics, while UML deployment diagram identifies the physical resources or components of the system. This specification style later generates the probable states based on which our performance model will be generated and solved by our performance modeling framework for relevant performance metrics captured by the annotated UML models. However, the size of the underlying reachability set is major limitation for large and complex system. Further work includes automating the whole process of translating from our UML specification style to generate a performance model and the way to solve the performance model through our proposed framework as well as tackling the state explosion problems of reachability marking for large system.

References

[1] OMG 2009, “UML Superstructure”, Version-2.2

[2] F. A. Kraemer, P. Hermann, “Service specification by composition of collaborations-an example”, Proceedings of 2006 WI-IAT workshops, Hong kong, p. 129-133, IEEE, 2006

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

[4] Pakke Kahkipru, “UML based performance modeling framework for object oriented distributed systems”, Proceedings of the 2nd international conference on the unified modeling language: beyond the standard, pp. 356-371, Springer-Verlag Berlin, Heidelberg, 1999

[5] J. P. Lopez, J. Merseguer, J. Campos, “From UML activity diagrams to SPN: application to software performance engineering”, ACM SIGSOFT software engineering notes, NY, USA, 2004

[6] J. Trowitzsch, A. Zimmermann, “Using UML state machines and petri nets for the quantitative investigation of ECTS”, Proceeding of the 1st international conference on performance evaluation methodologies and tools, ACM, USA, 2006

[7] Abdullatif, R. Pooly, “A computer assisted state marking method”, International Journal of Simulation, Vol. 8, No. 3, ISSN – 1473-804x

[8] F. A. Kramer, “ARCTIS”, Department of Telematics, NTNU, http://arctis.item.ntnu.no.

[9] F. A. Kramer, “Engineering Reactive Systems: A Compositional and Model-Driven Method Based on Collaborative Building Blocks”, Doctoral thesis, NTNU, Norway, 2008

[10] F. A. Kramer, R. Bræk, P. Herrmann, “Synthesizes components with sessions from collaboration-oriented service specifications”, Proceedings of SDL 2007, V-4745, Lecture notes of Computer Science, p.166-185.

[11] OMG 2005, “UML Profile for Schedulability, Performance, and Time Specification”, Version – 1.1

[12] K. S. Trivedi, R. Sahner, “SHARPE: Symbolic Hierarchical Automated Reliability / Performance Evaluator”, Duke University, Durham, NC

Paper 2

Translation from UML to Markov model: A performance modeling framework for managing behavior of multiple collaborative sessions and instances

Razib Hayat Khan, Poul E. Heegaard

Presented in IEEE International Conference in Computer Science and Information Technology (ICCSIT), Chengdu, China, 2010

Published in the Proceedings of the IEEE International Conference in Computer Science and Information Technology (ICCSIT), Vol. 1, p. 677-686, IEEE Computer Society Press, 2010

Translation from UML to Markov model: A performance modeling framework for managing behavior of multiple collaborative sessions and instances

Razib Hayat Khan, Poul E. Heegaard

Department of Telematics
Norwegian University of Science and Technology (NTNU)
Trondheim, Norway
{rkhan, poul.heegaard}@item.ntnu.no

Abstract- Performance evaluation of a distributed system is always an intricate undertaking, where system behavior is normally distributed among several components those are physically distributed. Bearing this concept, we delineate a performance modeling framework for a distributed system that proposes a translation process from high level UML notation to Markov model and solves the model for relevant performance metrics. To capture the system dynamics through our proposed framework, we outline a specification style that focuses on UML collaboration and activity as reusable specification building blocks. To present the UML specification style, we focus on how to coordinate explicitly multiple collaborative sessions occurring at the same time. Design alternatives of system architecture are considered to generate the performance model to show how these design alternatives thus affect the system performance under different work load. The proposed performance modeling framework provides prediction result of a system such as mean response time and resource utilization. The applicability of our proposed framework is demonstrated in the context of performance modeling of a distributed system.

1 Introduction

Distributed system is one of the main streams of information and communication technology arena. Modeling, developing, and implementation of such complex systems are always a difficult endeavor. Likewise, performance evaluation is also a great concern of such complex system to evaluate whether the system meets the performance related system requirements. However, in a distributed system, system behavior is normally distributed among several objects. The overall behavior of the system is composed of the partial behavior of the objects of the system. So it is obvious to capture the behavior of the distributed objects of the system to evaluate the performance of the overall system. We therefore, adopt UML collaboration and activity oriented approach as UML is the most widely used modeling language, which models both the system requirements and qualitative behavior through different notations. Collaboration and activity diagrams are utilized to demonstrate the overall system behavior by defining both the structure of the partial object behavior as well as the interaction between them as reusable specification building blocks and later on, this UML specification style is applied to generate the Markov model by our proposed performance modeling framework. UML collaboration and activity provides a tremendous modeling framework containing several interesting properties [1]. Firstly, collaborations and activity model the concept of service provided by the system very nicely. They define structure of partial object behaviors, the collaboration roles and enable a precise definition of the overall system behavior. They also delineate the way to compose services by means of collaboration uses and role bindings [2].

In addition, the proposed modeling framework considers design alternatives of system execution architecture to generate performance model of the system. This will help showing the performance affect because of changing of the system execution architecture and to help finding out the better system architecture candidate to fulfill certain performance goal at the early stage of the system development process. Abstract view of the system architecture is captured by the UML deployment diagram, which defines the execution architecture of the systems by identifying the system components and the assignment of software artifacts to those identified system components [1]. Considering the system architecture while generating the performance model also resolves the bottleneck of system performance by finding a better allocation of service components to the physical components of the system.

The *Unified Modeling Language* (UML) is a widely accepted modeling language to model the system behavior [1]. But it is indispensable to extend the UML model to incorporate the performance related quality of service (QoS) information to allow modeling and evaluating the properties of a system like throughput, utilization, mean response time. So the UML models are annotated according to the *UML Profile for Schedulability, Performance, and Time* (SPT) [3] to include quantitative system parameters in the model.

Markov models, queuing networks, stochastic process algebras and stochastic petrinet are probably the best studied performance modeling techniques [4]. Among all of them, we will choose Markov model as the performance model generated by our proposed framework for providing performance prediction result of a system due to its modeling generality, its well-developed numerical modeling analysis techniques, its ability to preserve the original architecture of the system, to facilitate any modification according to the feedback from performance evaluation and the existence of analysis tools.

Numbers of efforts have been made already to generate a performance model from the system design specification. Kähkipuro developed a performance modeling framework to generate and solve queuing network with simultaneous resource possessions from the high level UML notations [5]. Lopez-Grao *et al.* proposed a conversion method from annotated UML activity diagram to stochastic petrinet model [6]. Abdullatif and Pooly presented a method for providing computer support for extracting Markov chains from a performance annotated UML sequence diagram [7]. The framework presented here is the first known approach that introduces a new specification style utilizing UML behavioral diagrams as reusable specification building block for managing multiple collaborative sessions that executed at the same time, which is later on, used for generating performance model to produce performance prediction result at early stage of the system development process.

The objective of the paper is to provide an extensive performance modeling framework that provides a translation process to generate markov performance model from system design specification captured by the UML behavioral diagram for multiple collaborative sessions that executed at the same time. The framework also considers design alternatives of the system architecture and later on, solves the model for relevant performance metrics to demonstrate performance prediction results at early stage of the system development life cycle. The paper is organized as follows: Section 2 introduces our proposed performance modeling framework, application example is demonstrated in Section 3 and Section 4 delineates the conclusion with future works.

2 Performance modeling framework

Our proposed performance modeling framework utilizes the tool suite Arctis, which is integrated as plug-ins into the eclipse IDE [8]. The proposed framework is composed of 7 steps shown in Figure 1, where steps 1 and 2 are the parts of Arctis tool suite. Arctis focuses on the abstract, reusable service specifications that are composed form UML 2.2 collaborations and activities. It uses collaborative building blocks as reusable specification units to create comprehensive services through composition. To support the construction of building block consisting of collaborations and activities, Arctis offers special actions and wizards. In addition, a number of inspections ensure the syntactic consistency of building blocks. A developer first consults a library to check if an already existing collaboration block or a collaboration of several blocks solves a certain task. Missing blocks can also be created from scratch and stored in the library for later reuse. The building blocks are expressed as UML models. The structural aspect, for example the service component and their multiplicity, is expressed by means of UML 2.2 collaborations. For the detailed internal behavior, UML 2.2 activities have been used. They express the local behavior of each of the service components as well as their necessary interactions in a compact and self-contained way using explicit control flows [8]. Moreover, the building blocks are combined into more comprehensive service by composition. For this composition, Arctis use UML 2.2 collaborations and activities as well. While collaborations provide a good overview of the structural aspect of the composition, i.e., which sub-services are reused and how their collaboration roles are bound, activities express the detailed coupling of their respective behaviors. To reason about the correctness of the specifications, we introduce formal reasoning on the level of collaborative service specifications using temporal logic specification style cTLA/c(c for collaborative), which is beyond the scope of the paper [8]. The step of our proposed modeling framework is described as follows:

1) Construction of collaborative building block: The proposed framework utilizes collaboration as main specification units. The specifications for collaborations are given as coherent, self-contained reusable building blocks. The structure of the building block is described by UML 2.2 collaboration. If the building block is elementary it only declares the participants (as collaboration roles) and connection between them. If it is composite, it may additionally refer to other collaborations between the collaboration roles by means of collaboration uses. The internal behavior of building block is described by UML activity. It is declared as the classifier behavior of the collaboration and has one activity partition for each collaboration role in the structural description. For each collaboration use, the activity declares a corresponding call behavior action refereeing to the activities of the employed building blocks.

Depending on the number of participants, connectivity to other blocks and level of decomposition, we distinguish three different kinds of building blocks [8]:

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

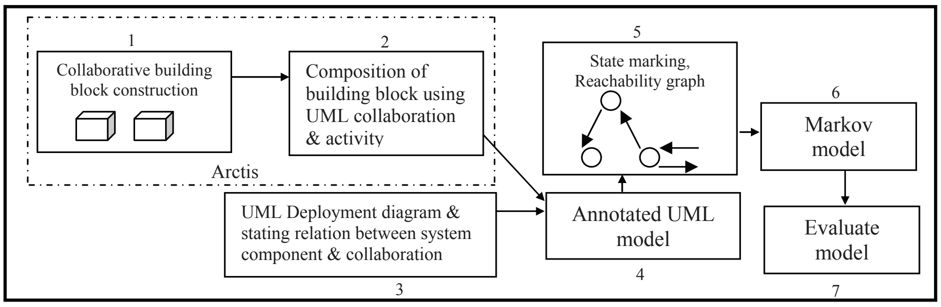

1
Collaborative building block construction
2
Composition of building block using UML collaboration & activity
Arctis
5
State marking, Reachability graph
6
Markov model
UML Deployment diagram & stating relation between system component & collaboration
3
Annotated UML model
4
Evaluate model
7

Figure 1. Performance modeling framework

2) Composition of building block using UML collaboration and activity: To generate the performance model, the structural information about how the collaborations are composed is not sufficient. It is necessary to specify the detailed behavior of how the different events of collaborations are composed so that the desired overall system behavior can be obtained. For the composition, UML collaborations and activities are used complementary to each other; UML collaborations focus on the role binding and structural aspect, while UML activities complement this by covering also the behavioral aspect for composition. For this purpose, call behavior actions are used. Each sub-service is represented by a call behavior action referring to respective activity of the building blocks. Each call behavior action represents an instance of a building block. For each activity parameter nodes of the referred activity, a call behavior action declares a corresponding pin. Pins have the same symbol as activity parameter nodes to represent them on the frame of a call behavior action. Arbitrary logic between pins may be used to synchronize the events of the building block and transfer data between them. By connecting the individual input and output pins of the call behavior actions, the events occurring in different collaborations can be coupled with each other. There are different kinds of pins described as follows [8]:

- Starting pins activate the building block, which is the precondition of any internal behavior.

- Streaming pin may pass tokens throughout the active phase of the building block.
- Terminating pins mark the end of the block's behavior.

If collaborations is started and terminated via several alternative pins, they must belong to different parameter sets. This is visualized in UML by an additional box around the corresponding node.

To present the UML specification style, we focus on how to coordinate explicitly multiple collaborative sessions occurring at the same time. To reflect the multiplicity of the service components, their partitions are represented by several layers. This multiplicity of partitions implies a certain multiplicity of collaborations that have to be coordinated by the service component. A token arriving from any of the collaboration instances simply enters the partition. Vice-versa, when a token should enter a specific collaboration instance from the partition, we need to determine which instance should receive the token. UML does not give any means to select such session. Therefore, we include one selection operator in the execution profile. To represent the overall system behavior for multiple session instances, the different sessions must be distinguished at run time.

3) UML deployment diagram and stating relation between system component and collaboration: Deployment diagram can be used to define the execution architecture of systems by identifying the system components and the assignment of software artifacts to those identified system components [1]. After designing the deployment diagram the relations between system components and collaborations will be delineated to describe the service delivered by the system. The service is delivered by the joint behavior of the system components, which may be physically distributed. The partial behavior of the components utilized to realize the collaborations is represented by the collaboration role. In this way, it is possible to expose direct mapping between the collaboration roles to the system components to show the probable deployment of service components in the physical nodes of the system.

4) Annotating the UML model: Performance information is incorporated into the UML activity diagram and deployment diagram according to the *UML Profile for Schedulability, Performance and Time* [3]. UML model is needed to annotate for evaluating system performance by performance model solver for relevant performance metrics through our proposed framework.

5) State marking and reachability graph: To generate the reachability graph, we consider the behavior of the system components, which are the subject of the bottleneck of the system performance and the states of the system will be generated based on the activity performed by those components, which are shown by their internal behavior explained in step 1. This section involves a state marking procedure that will mark the states of the system for each step executed and produce a reachability graph of the whole system. Each of the marked state will represent a state of the overall Markov chain. The method for state marking of the system is as follows: first an initial situation is marked, which is defined as the initial state of the system before executing the first step. Then, when a step will be executed the activity performed by the system component in this step will be marked as a performance model state, from where a new state may be generated or return back to an already marked state with a transition rate. This procedure will be continued until all the steps are executed. After that the reachability graph of the system will be produced. The execution of steps will be outlined in the composition of building blocks while describing the overall system behavior. The number and sequence of execution of steps will differ according to the system design specification.

6) Generating markov model: If each of the reachable marking is represented as a node and each edge between the nodes is leveled with the trigger of their change by the transition rate (mentioned in annotated UML model), we can generate the markov model of the system.

7) Evaluate model: Generated markov model will be used as input for the SHARPE tool [9] to generate performance prediction result of the system.

3 Application example

As a representative example, we introduce a scenario description to show the applicability of our proposed framework in designing, modeling and performance evaluating of a distributed system. Several users are equipped with cell phones or PDAs want to receive weather information of their current location using their hand held devices. The user request is first transferred to location servers through base transceiver station to retrieve the location information of the user. The location information is then transferred to weather servers for retrieving the weather information according to the location of the user. Figure 2 defines this scenario as UML 2.2 collaboration. Participants in the system are users, mobile terminals, base transceiver stations, location servers, weather servers, which are represented by the collaboration roles user, MT, BTS, LS, and WS. The users are the part of the environment and therefore labeled as <<external>>.The default multiplicity of the users, MT, BTS, LS, WS are many, which are denoted by [1..*]. The interactions between the collaboration roles are represented by the collaboration use such as MT and BTS interact through *t*: *transfer*, BTS and LS, WS interact through collaboration uses *l: request location info, w: request weather info,* while the user interacts with the MT by collaboration use *g: generate request*.

1) Construction of collaborative building block: This step defines the formulation of the building blocks in form of collaboration as major specification unit of our framework shown in Figure 2. The structure of the building block is defined by the UML collaboration shown in Figure 3(a). The building block declares the participants as collaboration roles and connection between them. While the UML collaboration describes the structural aspect of the composed service, the internal behavior of the collaboration is described by the UML activity [2]. Hereby, collaborations of Figure 2 are modeled by a call behavior action referring to the activity to describe the behavior of the corresponding collaboration [10]. Activity diagram presents complete behavior in a quite compact form and may define connections to other behaviors via input and output pins. We specify the behavior of one user request to show how the request is generated from user's MT and processed by the BTS, LS and WS and later on, compose this behavior for multiple user requests to show how the requests will be processed by the multiple BTS, multiple instance of LS and WS so that the overall system behavior can be delineated. To deliver the requested information to the user through his/her MT, BTS participates in the collaboration *Request Location Info* together with the LS and *Request Weather info* together with WS. These are specified by the collaboration *l: Request Location Info* and *w: Request Weather info,* where the BTS plays the role client and the LS and WS play the role server. The behavior of the collaboration is described by the UML activity in Figure 3(b), where activity is divided into two partitions: one for each collaboration role (Client and Server). The activity is started on the client side, when the user request is provided as parameter *u_req* at the input pin. The *u_req* directly sent to the LS, where it is converted into a database request by the call behavior action *processing.* After that, it is the task of the collaboration between the server and the database to provide the stored information. To get the information the request leaves the activity *Request Location info* and the server waits for the reception of response. This is modeled with the input and output pins *request* and *response*.

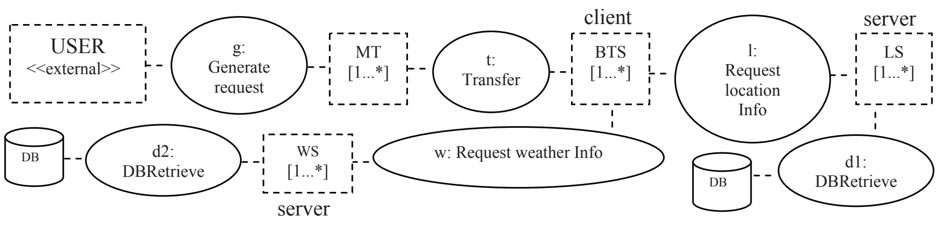


Figure 2. Collaboration diagram

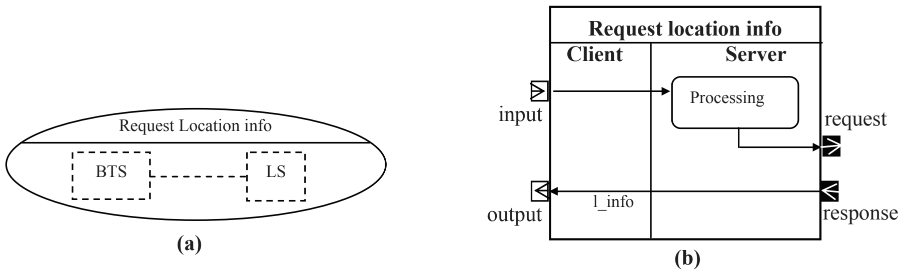
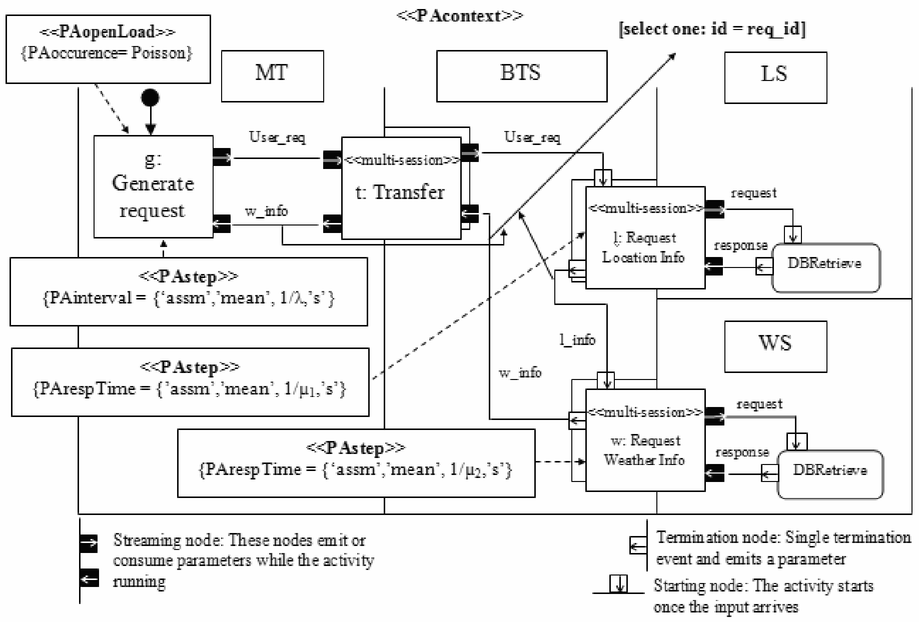


Figure 3. Structure (UML collaboration), Internal behavior (UML activity)

After getting the response the result *l_info* (location information) is delivered to the corresponding output pin in the client side by the call behavior action *delivery* and the activity is terminated. We describe the behavior of collaboration *Request Location info.* Likewise, we can describe the behavior of *Request Weather info* through activity partition of client and server, where location information of user is forwarded by the client to request server to retrieve weather information of the user location.

2) Composition of building block: Figure 4 shows the activity diagram for our system to highlight the overall behavior of the system by composing all the building blocks via several pins. The initial node (●) marks the starting of the activities. The activity is started on the client side. When a user service request is generated via MT, *g: Generate request* will transfer the user service request as parameter *u_req* to the BTS via collaboration *t: Transfer*. Once arrived at the BTS request for location information is forwarded to the LS represented by activity *Request location info*. LS makes a database request, which is modeled by *d1: DBRetrieve* and terminates with result *l_info* (Location information). After getting the location information, request for weather information according to user current location is forwarded by the BTS to the WS represented by activity *Request weather info.* WS makes a database request, which is modeled by *d2: DBRetrieve* and terminates with result *w_info* (Weather information). After that, the final result is transferred to the user hand held device by BTS via collaboration *t: Transfer.* The structure of collaborations as well as the way to couple them facilitates the reuse of activities. For example, both the collaboration *d1* and *d2* are identical and can be instantiated from single collaboration type. Moreover, the collaboration *l* and *w* have very similar behavior and can be based on the same UML template. Thus, systems of a specific domain can often be composed of reoccurring building blocks by reusing them [10].

From the viewpoint of one user, one location server and one weather server, there is exactly one collaboration session for the collaboration use *t, l* and *w* towards the BTS at certain time. This can be handled easily with the UML activity diagram in their standard form. But one BTS has to maintain several sessions with each of the user and each of the location and weather server at certain time. From the viewpoint of one BTS, several instances of the collaboration use *t, l* and *w* are executed at the same time; one instance for each user, location server and weather server. From the viewpoint of BTS, the collaborations that it participates are called *multi-session* collaboration. We express this by applying a stereotype <<multi-session>> to the call behavior actions and represents it graphically by multiple borders in those partitions, where sessions are multiple shown in Figure 4 [2]. One of the important issues is that how the different instances of collaborations may be distinguished and coordinated, so that desired overall system behavior is obtained. Therefore, we need a selection mechanism so that selection of sessions must take place, whenever a token enters a multi-session sub-collaboration and the overall system behavior can be reflected correctly for multiple instance of session for users, location servers and weather servers. While in some cases we may want to address all of the sessions, in other ones we like to select only a subset or one particular session. The UML standard however does not elaborate this matter. This is too restrictive, as most system exhibit patterns with several executions going on at a time that possibly need coordination [2]. We therefore, added the new operators **select** to our execution profile. To represent the overall system behavior for multiple session instances, the different sessions must be distinguished at run time. This resembles the well-known session pattern [11], which is found in client/server communication, where server has some kind of identifier to distinguish different sessions. For our case, we can assign an ID to the each request (req_id) to identify the session instance of *the transfer, request location info and request weather info* collaboration. When BTS receives the response form the location server about the location of the user, a token leaves output pin *l_info* and enters *w: request weather info*. Here, we have to select the session instance of the user so that user request can be successfully processed. Likewise, selection of session instance of user should be chosen to deliver the result to the user hand held devices. As they are distinguished by the request id we leave this number as attribute *req_id* inside the token and extract it by writing **select one : id = req_id**. The complete EBNF definition for session selection is given in Figure 5.



<<PAcontext>>
[select one: id = req_id]
<<PAopenLoad>>
{PAoccurence= Poisson}
MT
BTS
LS
WS
g: Generate request
User_req
w_info
<<multi-session>>
t: Transfer
User_req
<<multi-session>>
l: Request Location Info
request
response
DBRetrieve
l_info
w_info
<<multi-session>>
w: Request Weather Info
request
response
DBRetrieve
<<PAstep>>
{PAinterval = {'assm','mean', 1/λ,'s'}
<<PAstep>>
{PArespTime = {'assm','mean', 1/μ1,'s'}
<<PAstep>>
{PArespTime = {'assm','mean', 1/μ2,'s'}
Streaming node: These nodes emit or consume parameters while the activity running
Termination node: Single termination event and emits a parameter
Starting node: The activity starts once the input arrives

Figure 4. System activity to couple the collaboration

select := 'select' mod ':' [{filter}] [{'/' filter}].
mod := 'one' | 'all'.
filter := name | 'self' | 'active'| 'id=' variable.

Figure 5. EBNF for select

3) UML deployment diagram and stating relation between system component and collaboration: We consider two design alternatives of system architecture to demonstrate the relationship between collaboration and system component. In the first case the identified system components by our deployment diagram shown in Figure 6(a) are mobile terminal, base transceiver station, location server and weather server. The artifacts locationserverprocess.exe and weatherserverprocess.exe are assigned successively to location server and weather server. After designing the deployment diagram the relationship between system component and collaboration will be delineated to describe the service delivered by the system. The service is delivered by the joint behavior of the system components, which may be physically distributed. The partial behavior of the component utilized to realize the collaboration is represented by the collaboration role.

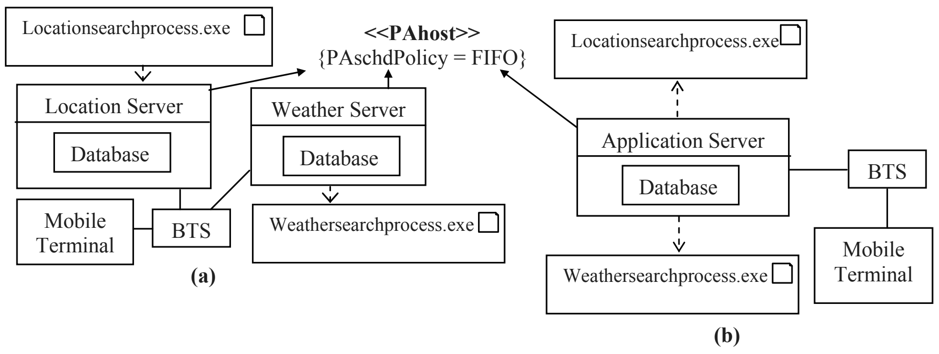


Figure 6. Two variations of UML deployment diagram

For our defined system description the behavior of the components mobile terminal, base transceiver station, location server, Weather server are represented by the collaboration roles MT, BTS, LS and WS to utilize the collaboration *t: transfer, l: request location info, w: request weather info*. Here, it is one to one mapping between system components and collaboration roles shown in Figure 7.

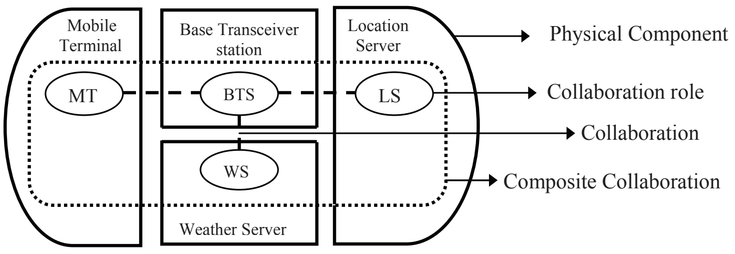


Figure 7. Relation between system components and collaborations for first variation of deployment diagram

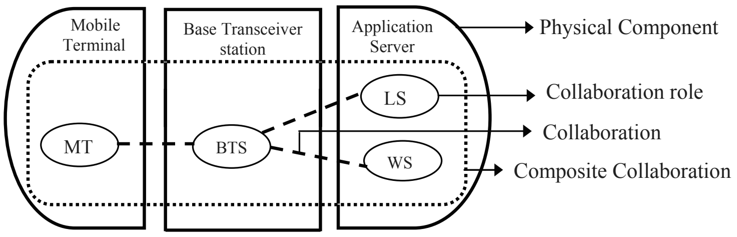


Figure 8. Relation between system components and collaborations for second variations of deployment diagram

Later on, we consider other variation of the deployment diagram for our mentioned scenario, which is shown in Figure 6(b). In this variation of deployment diagram the identified system components are mobile terminal, base transceiver station, application server, where the artifact locationserverprocess.exe, weatherserverprocess.exe are assigned jointly to application server. In this case, the behavior of the components mobile terminal and base transceiver station is represented by the collaboration roles MT and BTS to utilize collaboration *t: transfer* and the behavior of the component application server is represented jointly by the collaboration role LS and WS to utilize collaboration *l: request location info* and *w: request weather info* shown in Figure 8. In the second case, the mapping between system components and collaboration roles is generalized into one to many relations.

4) Annotating the UML model: Performance information is incorporated into the UML activity diagram in Figure 4 and deployment diagram in Figure 6 according to the *UML Profile for Schedulability, Performance and Time* [4] to enable system performance to be evaluated by performance model solver for relevant performance metrics through our proposed framework. We use the stereotypes PAcontext, PAopenLoad, PAhost, PAstep and the tagged values PAoccurence, PAschdPolicy, PArespTime and PAinterval [3]. A performance context can be modeled by an activity graph that is stereotyped as a PAcontext. This means that all interactions specified in that activity graph represent scenarios in the sense of this profile (shown in Figure 4). A PAopenLoad is modeled as a stream of requests that arrive at a given rate in predetermined pattern with PAoccurence. A PAhost models a processing resource with tagged PAschdPolicy defining the policy by which access to the resource is controlled. A PAstep models a scenario step with tagged PArespTime defining a step's response time and PAinterval defines time interval between successive repetitions of a step.

5) State marking and reachability graph: To generate the performance model by our proposed framework, we will consider the above two design alternatives of the system architecture, where the components have multiple instances. For our example scenario and first variation of deployment diagram, we will consider the participants location server and weather server as limiting factor for the system performance. The statuses of these servers are defined by their internal behavior through collaboration *request location info* and *request weather info*. The status of the both the servers are defined as idle and processing. When a step (annotated as <<PAstep>> in Figure 4) will be executed the status of the servers will be marked as performance model state, from where a new state may be generated with a transition rate or return back to a already marked state with a transition rate mentioned in the annotated UML model in Figure 4. The states of the performance model are shown in Table 1 based on the status of both the servers. The states are: (idle, idle), (processing, idle), (idle, processing), (processing, processing), where the first part defines the status of the location server and second part defines the status of the weather server. If we assume initial marking such as the status of the location server and weather server is idle that means servers have no user request to process then we can derive all the reachable markings to produce reachability graph of the system from the initial marking according to the arrival or departure of client requests.

6) Markov model generation: If each of the reachable marking is represented as a node and each edge between the nodes level with the trigger of their change by the transition rate (mentioned in annotated UML model by λ, μ_1, μ_2), we can generate the markov model of the system. We consider the number of location server and weather server in our system is 3, where each server can process one request at a time. We assume system is stable and both servers buffer capacity are null, the state transition diagram is shown in Figure 9, where system states are generated from location server and weather server status shown in Table 1, where idle means no job is in the servers to process and processing means 1 job (number of job in the location server and weather server is mentioned by N and M and here highest value of N=M=1 in Figure 9) is processed by the servers. if we assume the system is stable, both servers buffer capacity is infinite, follow Poisson arrival pattern and FIFO (First In First Out) scheduling policy and servers can process one request at a time, the state transition diagram is shown in Figure 10 (where (N, M) >1 to infinity), which shows more states than the states generated from the status of both the servers. So if N=M=1 then the state transition diagram will be mentioned in Figure 9, which just reflect the internal behavior of the servers showing the change of system states mentioned in Table 1. If (N, M) > 1 the state transition diagram will be mentioned in Figure 10 highlighting more additional states including the states shown in Table 1 and Figure 9, where the states will be marked by the total number of job N and M in the server. In this case, if (N, M) $\leq$ (n, m), there are N, M customers in the system and n-N, m-M servers are idle and If (N, M) $\geq$ (n, m) there are N, M customers in the system, the n, m servers are busy and there are N-n, M-m customers in the queue (Here, n, m are the number of location servers and weather servers).

Table1. States of performance model based on the status of location server & weather server

Location Server	Weather Server
Idle	Idle
processing	Idle
Idle	Processing
processing	Processing

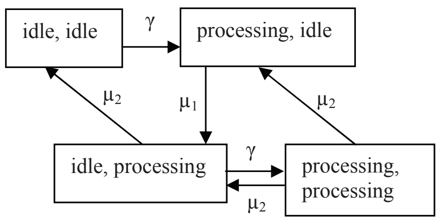


Figure 9. State transition diagram of the Markov model when number of request or job arrived and serviced by the system is 1

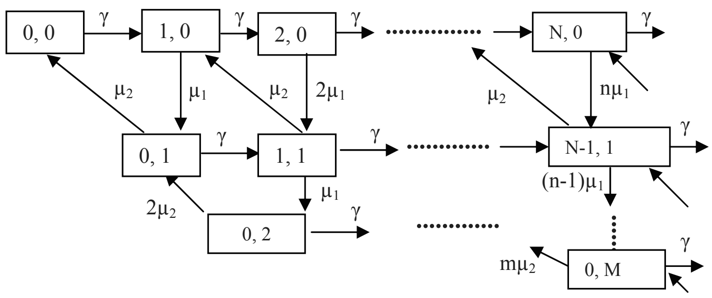


Figure 10. State transition diagram of the Markov model for 1st variation of deployment diagram

The markov model for the second variation of deployment diagram can be generated by following the above procedure as well, where application server will be considered as performance limiting factor of the system. The performance model for this case is shown in Figure 11.

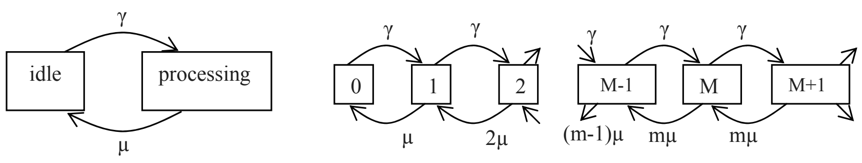


Figure 11. State transition diagram of the Markov model for 2nd variation of deployment diagram

7) Evaluate model: The generated performance model will be solved by the SHARPE performance model solver [9] to generate performance prediction result. Some of the performance evaluation results generated by the tool are shown in the graph form in Figure 12 for both variations of the deployment diagrams, which highlight their affect on the system performance.

We modeled and solved the system for both design alternatives of system architectures. The performance prediction result such as mean response time and utilization of servers are derived for the same arrival rate of the client requests and service rate of the servers for both design alternatives of the system architecture. The comparison of the result is demonstrated in the graph in Figure 12 for the servers of the both design alternatives. The result clearly shows how the response time of the system varies with the same server utilization for considering the different system architecture candidates that helps the developers to resolve the bottleneck of system performance by finding a better allocation of service components to the physical components of the system.

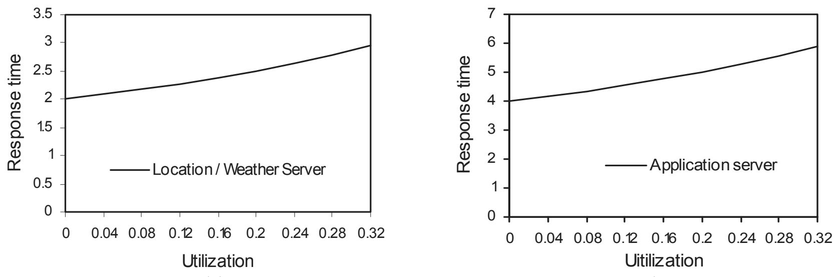


Figure 12. Response time vs. Utilization (a) for the server of 1st (b) for the server of 2nd variations of deployment diagram

4 Conclusion

Our main contribution is delineated as to present the UML collaboration and activity oriented approach to capture the system dynamics that is utilized to sketch the performance model for a distributed system, where every collaboration performs separate task. The behavior of the collaboration and the composition of collaboration to highlight the overall system behavior are demonstrated by utilizing UML activity. To present the behavior and composition of the collaboration using activity, we extend the notation to handle the collaboration that is executed not only in the single session but also in multiple sessions at the same time, where different instances of collaborations are distinguished and coordinated by adding notation **select** to our execution profile. The **select** notation can outline the relations between multiple sessions unambiguously on an abstract level. Later a mapping between collaboration role and system component is outlined to show how the service of the distributed system is realized by the joint behavior of the system components that are physically distributed. Different variations of deployment diagram are considered to generate the performance model to show how the variations in the deployment diagram thus affect the system performance under different work load. Further work includes automating the whole process of translation from our UML specification style to generate a performance model and the way to solve the performance model through our proposed framework as well as to tackle the state explosion problem for large systems.

References

1. OMG 2009, "UML Superstructure", Version-2.2
2. F. A. Kramer, R. Bræk, P. Herrmann, "Synthesizes components with sessions from collaboration-oriented service specifications", Proceedings of SDL 2007, V-4745, Lecture notes of Computer Science, p.166-185, 2007.
3. OMG 2005, "UML Profile for Schedulability, Performance, and Time Specification", V – 1.1
4. K. S. Trivedi, "Probability and Statistics with Reliability, Queuing and Computer Science application", Wiley-Interscience publication, ISBN 0-471-33341-7
5. Abdullatif, R. Pooly, "A computer assisted state marking method", International Journal of Simulation, Vol. 8, No. 3, ISSN – 1473-804x
6. Pakke Kahkipru, "UML based performance modeling framework for object oriented distributed systems", Proceedings of the 2nd international conference on the unified modeling language: beyond the standard, pp. 356-371, Springer- Verlag Berlin, Heidelberg, 1999
7. J. P. Lopez, J. Merseguer, J. Campos, "From UML activity diagrams to SPN: application to software performance engineering", ACM SIGSOFT software engineering notes, NY, 2004
8. F. A. Kramer, "ARCTIS", Department of Telematics, NTNU, http://arctis.item.ntnu.no.
9. K. S. Trivedi, R. Sahner, "SHARPE: Symbolic Hierarchical Automated Reliability / Performance Evaluator", Duke University, Durham, 2002
10. F. A. Kraemer, P. Herrmann, "Service specification by composition of collaborations-an example", Proceedings of WI-IAT workshops, Hong kong, p. 129-133, 2006
11. Linda Rising, "Design patterns in Communications Software", Cambridge University Press, NY, USA, 2001

Paper 3

Translation from UML to SPN model: A performance modeling framework for managing behavior of multiple collaborative sessions and instances

Razib Hayat Khan, Poul E. Heegaard

Presented in the International Conference on Computer Design and Application (ICDDA), 2010, Qinhuangdao, China

Published in the Proceedings of the International Conference on Computer Design and Application (ICDDA), Vol. 5, p. 72-80, IEEE Computer Society Press, 2010

Translation from UML to SPN model: A performance modeling framework for managing behavior of multiple collaborative sessions and instances

Razib Hayat Khan, Poul E. Heegaard

Department of Telematics
Norwegian University of Science and Technology (NTNU)
Trondheim, Norway
{rkhan, poul.heegaard}@item.ntnu.no

Abstract- Performance evaluation of a distributed system is always an intricate undertaking, where system behavior is normally distributed among several components those are physically distributed. Bearing this concept, we delineate a performance modeling framework for a distributed system that proposes a translation process from high level UML notation to SPN model and solves the model for relevant performance metrics. To capture the system dynamics through our proposed framework, we outline a specification style that focuses on UML collaboration and activity as reusable specification building blocks. To present the UML specification style, we focus on how to coordinate explicitly multiple collaborative sessions occurring at the same time. Design alternatives of system architectures are considered to generate the performance model to show how these design alternatives thus affect the system performance under different work load. The proposed performance modeling framework provides prediction result of a system such as mean response time. The applicability of our proposed framework is demonstrated in the context of performance modeling of a distributed system.

1 Introduction

Distributed system is one of the main streams of information and communication technology arena. Modeling, developing and implementation of such complex systems are always a difficult endeavor. Likewise, performance evaluation is also a great concern of such complex system to evaluate whether the system meets the performance related system requirements. However, in a distributed system, system behavior is normally distributed among several objects. The overall behavior of the system is composed of the partial behavior of the distributed objects of the system. So it is obvious to capture the behavior of the distributed objects of the system to evaluate the performance of the overall system. We therefore, adopt UML collaboration and activity oriented approach as UML is the most widely used modeling language, which models both the system requirements and qualitative behavior through different notations [2]. Collaboration and activity diagram are utilized to demonstrate the overall system behavior by defining both the structure of the partial object behavior as well as the interaction between them as reusable specification building blocks and later on, this UML specification style is applied to generate the SPN model by our proposed performance modeling framework. UML collaboration and activity provides a tremendous modeling framework containing several interesting properties. Firstly, collaborations and activity model the concept of service provided by the system very nicely. They define structure of the partial object behavior, the collaboration roles and enable a precise definition of the overall system behavior. They also delineate the way to compose services by means of collaboration uses and role bindings [1].

In addition, the proposed modeling framework considers design alternatives of system architecture to generate the performance model of the system to show the performance affect because of the changing of system architecture and to help finding out the better system architecture candidate to fulfill certain performance goal at the early stage of the system development process. Abstract view of the system architecture is captured by the UML deployment diagram, which defines the execution architecture of the system by identifying the system components and the assignment of software artifacts to those identified system components [2]. Considering the system architecture to generate the performance model also resolves the bottleneck of system performance by finding a better allocation of service components to the physical components.

The *Unified Modeling Language* (UML) is a widely accepted modeling language to model the system behavior [2]. But it is indispensable to extend the UML model to incorporate the performance-related quality of service (QoS) information to allow modeling and evaluating the properties of a system like throughput, utilization, and mean response time. So the UML models are annotated according to the *Profile for Schedulability, Performance, and Time* (SPT) to include quantitative system parameters [3].

Markov models, queuing networks, stochastic process algebras and stochastic petrinet (SPN) are probably the best studied performance modeling techniques [4]. Among all of them, we will choose SPN as the performance model generated by our proposed framework for providing performance prediction result of a system due to its increasingly popular formalism for describing and analyzing systems, its modeling generality, its ability to capture complex system behavior concisely, its ability to preserve the original architecture of the system, to facilitate any modification according to the feedback from performance evaluation and the existence of analysis tools.

Numbers of efforts have been made to generate a performance model from the system design specification. Lopez-Grao *et al.* proposed a conversion method from annotated UML activity diagram to stochastic petrinet model [5]. Kähkipuro developed a performance modeling framework to generate and solve queuing network with simultaneous resource possessions from the high level UML notations [6]. Abdullatif and Pooly presented a method for providing computer support for extracting Markov chains from a performance annotated UML sequence diagram [7]. The framework presented here is the first known approach that introduces a new specification style utilizing UML behavioral diagrams as reusable specification building block for managing multiple collaborative sessions that executed at the same time, which is later on, used for generating performance model to produce performance prediction result at early stage of the system development process.

The objective of the paper is to provide an extensive performance modeling framework that provides a translation process to generate SPN performance model from system design specification captured by the UML behavioral diagram for multiple collaborative sessions that executed at the same time and for design alternatives of the system architecture and later on, solves the model for relevant performance metrics to demonstrate performance prediction results at early stage of the system development life cycle. The work presented here is the extension of our previous work described in [12], where we presented our proposed framework with respect to the execution of single collaborative session at certain time and considered single system architecture candidate to describe the system behavior. The paper is organized as follows: Section 2 introduces our proposed performance modeling framework, Section 3 demonstrates the application example to show the applicability of our performance modeling framework, Section 4 delineates the conclusion with future works.

2 Performance modeling framework

Our proposed performance modeling framework utilizes the tool suite Arctis, which is integrated as plug-ins into the eclipse IDE [8]. The proposed framework is composed of 6 steps shown in Figure 1, where steps 1 and 2 are the parts of Arctis tool suite. Arctis focuses on the abstract, reusable service specifications that are composed form UML 2.2 collaborations and activities. It uses collaborative building blocks as reusable specification units to create comprehensive services through composition. To support the construction of building block consisting of collaborations and activities, Arctis offers special actions and wizards. In addition a number of inspections ensure the syntactic consistency of building blocks. A developer first consults a library to check if an already existing collaboration block or a collaboration of several blocks solves a certain task. Missing blocks can also be created from scratch and stored in the library for later reuse. The building blocks are expressed as UML models. The structural aspect, for example the service component and their multiplicity, is expressed by means of UML 2.2 collaborations. For the detailed internal behavior, UML 2.2 activities have been used. They express the local behavior of each of the service components as well as their necessary interactions in a compact and self-contained way using explicit control flows [8]. Moreover, the building blocks are combined into more comprehensive service by composition. For this composition, Arctis uses UML 2.2 collaborations and activities as well. While collaborations provide a good overview of the structural aspect of the composition, i.e., which sub-services are reused and how their collaboration roles are bound, activities express the detailed coupling of their respective behaviors [8]. To reason about the correctness of the specifications, we introduce formal reasoning on the level of collaborative service specifications using temporal logic specification style cTLA/c(c for collaborative), which is beyond the scope of this paper [8]. The steps of our proposed modeling framework are described as follows:

(1) Construction of collaborative building block: The proposed framework utilizes collaboration as main specification units. The specifications for collaborations are given as coherent, self-contained reusable building blocks. The structure of the building block is described by UML 2.2 collaboration. If the building block is elementary it only

declares the participants (as collaboration roles) and connection between them. If it is composite, it may additionally refer to other collaborations between the collaboration roles by means of collaboration uses. The internal behavior of building block is described by UML activity. It is declared as the classifier behavior of the collaboration and has one activity partition for each collaboration role in the structural description. For each collaboration use, the activity declares a corresponding call behavior action refereeing to the activities of the employed building blocks.

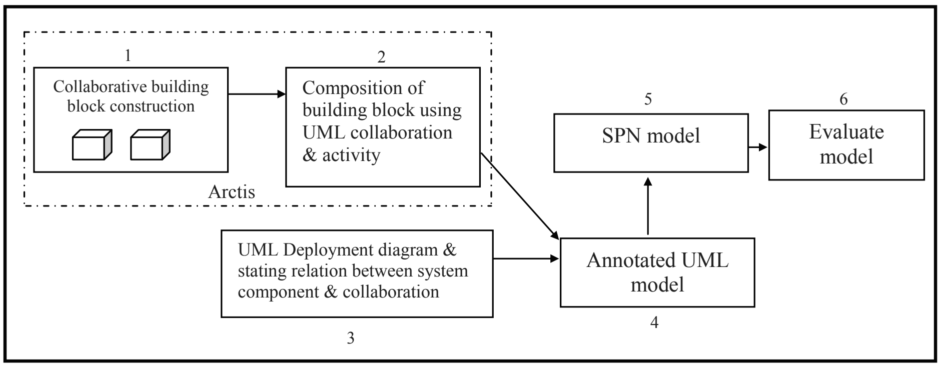

1
Collaborative building block construction
2
Composition of building block using UML collaboration & activity
Arctis
5
SPN model
6
Evaluate model
UML Deployment diagram & stating relation between system component & collaboration
3
Annotated UML model
4

Figure 1. Performance modeling framework

Depending on the number of participants, connectivity to other blocks and level of decomposition, we distinguish three different kinds of building blocks [10]:

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

(2) Composition of building block using UML collaboration and activity: To generate the performance model, the structural information about how the collaborations are composed is not sufficient. It is necessary to specify the detailed behavior of how the different events of collaborations are composed so that the desired overall system behavior can be obtained. For the composition, UML collaborations and activities are used complementary to each other; UML collaborations focus on the role binding and structural aspect, while UML activities complement this by covering also the behavioral aspect for composition. For this purpose, call behavior actions are used. Each sub-service is represented by a call behavior action referring to the respective activity of the building blocks. Each call behavior action represents an instance of a building block. For each activity parameter node of the referred activity, a call behavior action declares a corresponding pin. Pins have the same symbol as activity parameter nodes to represent

them on the frame of a call behavior action. Arbitrary logic between pins may be used to synchronize the building block events and transfer data between them. By connecting the individual input and output pins of the call behavior actions, the events occurring in different collaborations can be coupled with each other. There are different kinds of pins described as follows [10]:

- Starting pins activate the building block, which is the precondition of any internal behavior.
- Streaming pin may pass tokens throughout the active phase of the building block.
- Terminating pins mark the end of the block's behavior.

If collaborations is started and terminated via several alternative pins, they must belong to different parameter sets. This is visualized in UML diagram by an additional box around the corresponding node.

To present the UML specification style we focus on how to coordinate explicitly multiple collaborative sessions occurring at the same time. To reflect the multiplicity of the service components, their partitions are represented by several layers. This multiplicity of partitions implies a certain multiplicity of collaborations that has to be coordinated by the service component. A token arriving from any of the collaboration instances simply enters the partition. Vice-versa, when a token should enter a specific collaboration instance from the partition, we need to determine, which instance should receive the token. To represent the overall system behavior for multiple session instances, the different sessions must be distinguished at run time. UML does not give any means to select such session. Therefore, we include one selection operator in the execution profile.

(3) UML deployment diagram and stating relation between system component and collaboration: Deployment diagram can be used to define the execution architecture of system by identifying the system components and the assignment of software artifacts to those identified system components [2]. After designing the deployment diagram the relation between system component and collaboration will be delineated to describe the service delivered by the system. The service is delivered by the joint behavior of the system components, which may be physically distributed. The necessary partial behavior of the component used to realize the collaboration is represented by the collaboration role. In this way, it is possible to expose direct mapping between the collaboration roles to the system components to show the probable deployment of service components in the physical nodes of the system.

(4) Annotating the UML model: Performance information is incorporated into the UML activity diagram and deployment diagram according to the *UML Profile for Schedulability, Performance and Time* [3]. UML model is needed to annotate for evaluating system performance by performance model solver for relevant performance metrics through our proposed framework.

(5) Deriving SPN model: To generate the probable states of the SPN performance model we consider the system components, which are the subject of the bottleneck of the system performance. From the internal behavior of those system components and annotated UML structure, probable states and transition rate for triggering the change between

states will be found based on which our SPN performance model will be generated. To generate the SPN model of the system, first the SPN model of each component will be generated based on their internal behavior highlighted in step 1. After deriving individual SPN model for each component, all the individual SPN models will be integrated according to composite structure of the building blocks highlighted in step 2 to generate SPN model of the system. This makes easier to depict the SPN model of the system graphically under appropriate timing and probabilistic assumption.

(6) Evaluate model: SPN performance model will be used as input for SHARPE tool [9] for generating performance prediction result of a system.

3 Application example

As a representative example, we introduce a scenario description to show the applicability of our proposed framework in designing, modeling and performance evaluating of a distributed system. Several users are equipped with cell phone or smart phone want to receive weather information of their current location using his/her hand held devices. The user request is first transferred to authentication server through base transceiver station to ensure the authenticity of the user. Thereafter, the request of the legitimate user is transferred to the location server to retrieve the location information of the user. The location information is then transferred to weather server for retrieving the weather information according to the location of the user. Figure 2 defines this scenario as UML 2.2 collaboration. Participants in the system are users, mobile terminals, base transceiver stations, authentication servers, location servers, weather servers, which are represented by the collaboration roles user, MT, BTS, AuS, LS, and WS. The users are the part of the environment and therefore labeled as <<external>>. The default

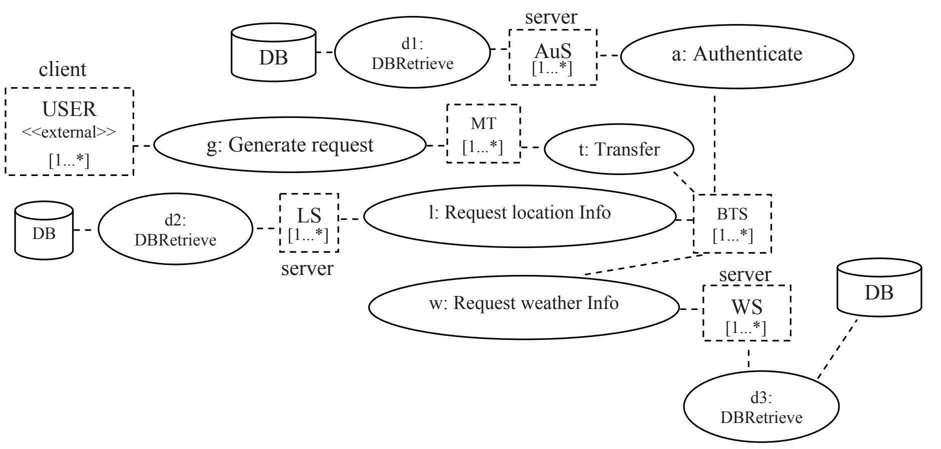


Figure 2. Collaboration diagram

multiplicity of the users, MT, BTS, AuS, LS, WS are many, which are denoted by [1..*]. The interactions between the collaboration roles are represented by the collaboration use such as MT and BTS interact through *t*: *transfer*, BTS and AuS, LS, WS interact successively through *a*: *authenticate, l: request location info, w: request weather info,* while the user interacts with the MT by collaboration use *g*: *generate request*.

(1) Construction of collaborative building block: The structure of the building block is defined by the UML collaboration shown in Figure 2. The building block declares the participants as collaboration roles and connection between them. The internal behavior of the collaboration is described by the UML activity [1]. Hereby, collaborations of Figure 2 are modeled by a call behavior action referring to the activity describing the behavior of the corresponding collaboration [1]. Activity diagram presents complete behavior in a quite compact form and may define connections to other behaviors via input and output pins [12]. Here, we specify the behavior of one user request to show how the request is generated from his/her MT and served by the BTS, AuS, LS, and WS and later on, compose this behavior for multiple user requests to show how the requests will be processed by the multiple instance of BTS, AuS, LS, and WS so that the overall system behavior can be delineated.

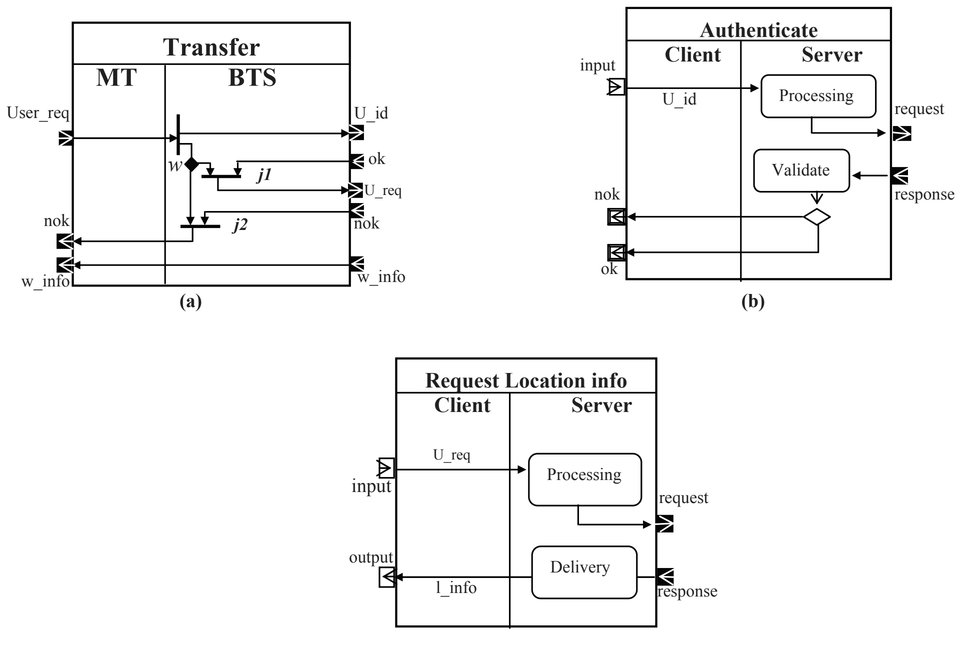


Figure 3. Internal behavior (UML activity)

The activity *transfer* describes the behavior of the corresponding collaboration shown in Figure 3(a). It has one partition for each collaboration role: MT and BTS. Activities base their semantics on token flow [1]. The system starts by placing a token in the initial node of the MT, when one request is generated by the user through his/her MT. The token is then transferred to the BTS, where it moves through the fork node generating two flows.

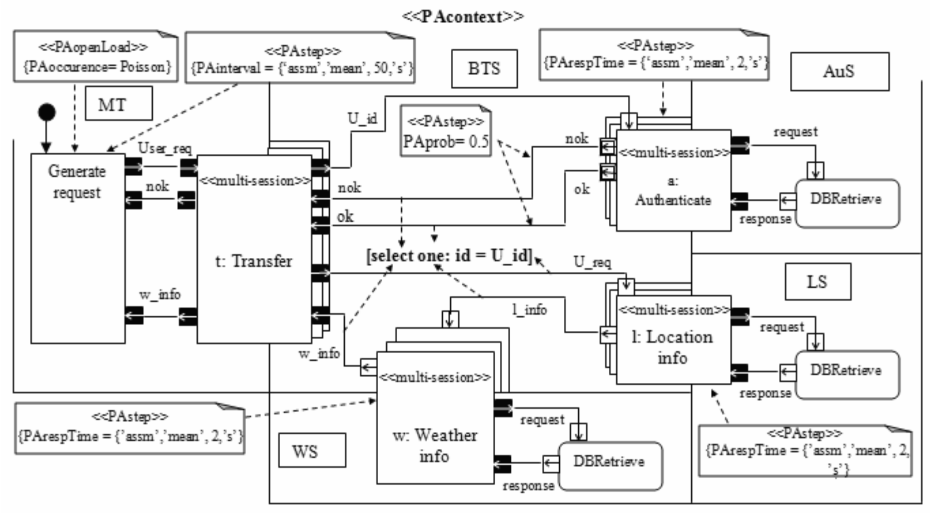
One flow places a token in the waiting decision node *w,* which is the extension of a decision node with the difference that it may hold a token similar to an initial node, as defined in [1]. *w* is used in combination with join nodes *j1* and *j2* to explicitly model the acceptance or rejection of the user request based on the user authenticity. The other flow is forwarded as input to the AuS to check whether the user is legitimate to generate service request. If the user is legitimate to generate the request a token is offered to the join node *j1*. If *w* still has its token *j1* can fire, which emits a token, which then forwarded to the LS for further processing. If the user is not legitimate to generate the request, a token is offered to the join node *j2*. If *w* still has its token *j2* can fire notifying the user upon the cancellation of request and then terminates the collaboration.

To validate the user identity (mobile number in this case) provided by a user who requests for service, BTS participates in the collaboration *authenticate* together with the AuS. This is specified by collaboration *a: authenticate,* where BTS plays the role client and the AuS plays the role server. The behavior of the collaboration defined by the UML activity, which is divided into two partitions, one for each collaboration role: client and server shown in Figure 3(b). The activity is started on the client side, when user id is provided as parameter *u_id* at the input pin. The input is then directly sent to server, where it is converted into a database request in the call behavior action *processing*. Thereafter, it is the task of the collaboration between the server and the database to provide the stored user information. To get the information, the request leaves the activity *authenticate* and the server waits for the reception of the response. This is modeled with the input and output pins *request* and *response*. Depending on the validity of the user id, the server may decide to report *ok* or *nok* (not ok) to the client by the call behavior action *validate*. The result is then forwarded to the corresponding output pin in the client side and the activity is terminated. The semantics of all the pins are given in [12].

Likewise, we can describe the behavior of collaboration *l: Request Location info* (shown in Figure 3(c)) and w: *Request Weather info* through activity partition of client and server, where BTS plays the role client and LS and WS play the role server to deliver the requested information to the user through his/her mobile terminal.

(2) Composition of building blocks: Figure 4 shows the activity diagram for our system to highlight the overall behavior of the system by composing all the building blocks. The initial node (●) marks the starting of the activity. The activity is started on the client side. When a user service request is generated via MT, *g: Generate request* will transfer the user service request as parameter *u_req* to the BTS via collaboration *t: Transfer*. Once the request arrived at the BTS, the user id as parameter *u_id* is transferred to the AuS to check whether the user is authenticate to accept the service and the activity is represented by *a: authenticate*. The activity *authenticate* initiates a database request, modeled by collaboration *d1: DBRetrieve* and terminates with one of the alternative results *ok* or *nok*. After arriving of the positive response at the BTS, request for location information is forwarded to the LS represented by activity *Request location info*. LS makes a database request, which is modeled by *d1: DBRetrieve* and terminates with result *l_info* (Location information). After getting the location information, request for weather information according to user current location is forwarded by the BTS to WS represented by activity

Request weather info. WS makes a database request, which is modeled by *d2: DBRetrieve* and terminates with result *w_info* (Weather information). After that, the final result is transferred to the user hand held device by BTS via activity *t: Transfer.* But if the user is failed to prove his identity then immediately a *nok* is sent to the user's hand held device.



<<PAcontext>>
<<PAopenLoad>> {PAoccurence= Poisson}
<<PAstep>> {PAinterval = {'assm','mean', 50,'s'}
BTS
<<PAstep>> {PArespTime = {'assm','mean', 2,'s'}
AuS
MT
Generate request
User_req
nok
w_info
<<multi-session>> t: Transfer
U_id
<<PAstep>> PAprob= 0.5
nok
ok
<<multi-session>> a: Authenticate
request
response
DBRetrieve
[select one: id = U_id]
U_req
LS
l_info
<<multi-session>> l: Location info
request
response
DBRetrieve
w_info
<<multi-session>> w: Weather info
request
response
DBRetrieve
<<PAstep>> {PArespTime = {'assm','mean', 2,'s'}
WS
<<PAstep>> {PArespTime = {'assm','mean', 2, 's'}

Figure 4. System activity to couple the collaboration

From the viewpoint of one user, one authentication server, one location server and one weather server, there is exactly one collaboration session for the collaboration use *t, a, l, w* towards the BTS at certain time. This can be handled easily with the UML activity diagram in their standard form. But one BTS has to maintain several sessions with each of the user and each of the authentication server, location server and weather server at certain time. From the viewpoint of one BTS, several instances of the collaboration use *t, a, l, w* are executed at the same time; one instance for each user, authentication server, location server and weather server. From viewpoint of BTS, collaborations that it participates are called *multi-session* collaboration. We express these by applying a stereotype <<multi-session>> to the call behavior action and represent them graphically by multiple borders in those partitions, where sessions are multiple. One of the important issues here is that how the different instances of collaborations may be distinguished and coordinated, so that desired overall system behavior can be obtained. So we need selection mechanism so that selection of sessions must take place whenever a token enters a multi-session sub-collaboration and the overall system behavior can be reflected correctly for multiple instances of session for users, authentication servers, location servers and weather servers. While in some cases we may want to address all of the sessions, in other ones we like to select only a subset or one particular session. The UML standard however does not elaborate this matter. This is too restrictive, as most systems exhibit patterns with several executions going on at a time, that possibly need coordination. We therefore, added the new operator **select** to our execution profile [1]. To represent the overall system behavior for multiple session instances, the different sessions must be distinguished at run time. This resembles the well-known session pattern [11], which is found in client/server communication, where server has some kind of identifier to distinguish different sessions.

For our case, we can use the ID of the user (mobile number of a user in this case) to identify the session instance of *the transfer, authenticate, request location info* and *request weather info* collaboration. When response form the authentication server about the user authenticity is decided, a token leaves either output pin *ok* or *nok* and enters *t: transfer*. Here, we have to select the session instance of the user so that user request can be successfully processed. Likewise, selection of session instance of user should be chosen properly to process the user's request and to deliver the result successfully to the user hand held devices. As they are distinguished by the user id number we leave this number as attribute *u_id* inside the token and extract it by writing **select one : id = u_id** [1]. The complete EBNF definition for session selection is given in Figure 5. It allows specifying several filters that are applied in order of listings [1].

```
select := 'select' mod ':' [{filter}] [{'/' filter}].
mod := 'one' | 'all'.
filter := name | 'self' | 'active'| 'id=' variable.
```

Figure 5. EBNF for select

(3) UML deployment diagram and stating relation between system component and collaboration: We consider two design alternatives of system architecture captured by UML deployment diagram to demonstrate the relationship between collaboration and system component. For our defined scenario the identified system components by the 1st variation of deployment diagram are mobile terminal, base transceiver station, authentication server, location server and weather server. After designing the deployment diagram the relationship between system component and collaboration will be delineated to describe the service delivered by the system. The service is delivered by the joint behavior of system components, which may be physically distributed. The necessary partial behavior of the component used to realize the collaboration is represented by the collaboration role. For our defined system description behavior of the components mobile terminal, base transceiver station, authentication server, location server, weather server are represented by the collaboration roles MT, BTS, AuS, LS and WS to utilize the collaboration *t: transfer, a: authenticate, l: request location info, w: request weather info*. Here, it is one to one mapping between system component and collaboration role shown in Figure 6(a).

Later on, we consider other variation of deployment diagram for mentioned scenario. In this variation of deployment diagram the identified system components are mobile terminal, base transceiver station, application server. In this case, the behavior of the components mobile terminal and base transceiver station is represented by the collaboration roles MT and BTS to utilize the collaboration *t: transfer* and the behavior

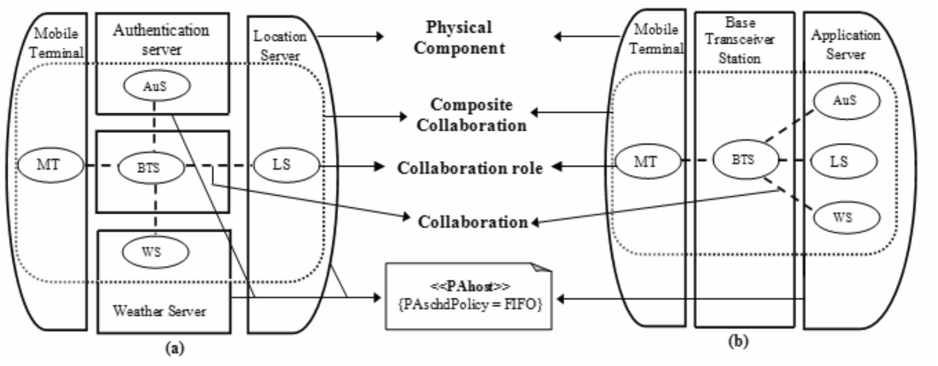


Figure 6. Relation between system components and collaborations

of the component application behavior is represented jointly by the collaboration role AuS, LS and WS to utilize the collaboration *a: authenticate, l: request location info, w: request weather info*. In second case, the mapping between system component and collaboration role is generalized into one to many relations shown in Figure 6(b).

(4) Annotating the UML model: To annotate the UML activity diagram in Figure 4 we use the stereotypes PAcontext, PAopenLoad, PAstep, PAhost and the tagged values PAoccurence, PArespTime, PAinterval, PAprob and PAschdPolicy [3]. A performance context can be modeled by an activity graph that is stereotyped as a PAcontext. This means that all interactions specified in that activity graph represent scenarios in the sense of this profile (shown in Figure 4). A PAopenLoad is modeled as a stream of requests that arrives at a given rate in predetermined pattern with PAoccurence. A PAstep models a scenario step with tagged PArespTime defining a step's response time, PAinterval defines time interval between successive repetitions of a step and PAprob defines the probability of occurring a step. A PAhost models a processing resource with tagged PAschdPolicy defining the policy by which access to the resource is controlled.

(5) Generating the SPN model: To generate the SPN performance model by our proposed framework, we will consider the above two design alternatives of system architecture, where the components have multiple instances. From the internal behavior of the system components and annotated UML structure, probable states and transition rate for triggering the change between states will be found based on which SPN model will be generated. To generate the performance model for the 1st variations of deployment diagram, we will consider the behavior of mobile terminal (as client), authentication server, location server and weather server as performance limiting factors of the system. The activities of the mobile terminal, authentication server, location server and weather server are *transfer*, *processing, validation* and *delivery* identified from their internal behaviors shown in Figure 3 are utilized as the states of the SPN performance model. From the internal behavior of each component, we generate the individual SPN model for each component and the individual SPN model for each component is then combined according to composite structure shown in Figure 4 to produce the SPN performance model for our system. The SPN performance model is shown in Figure 7, where states are

defined as places. We consider the number authentication server, location server, weather server in our system is 3, where each server can process one request at a time. So the initially places *Idle1*, *Idle*$_2$ and *Idle*$_3$ in Figure 7 contain 3 tokens each.

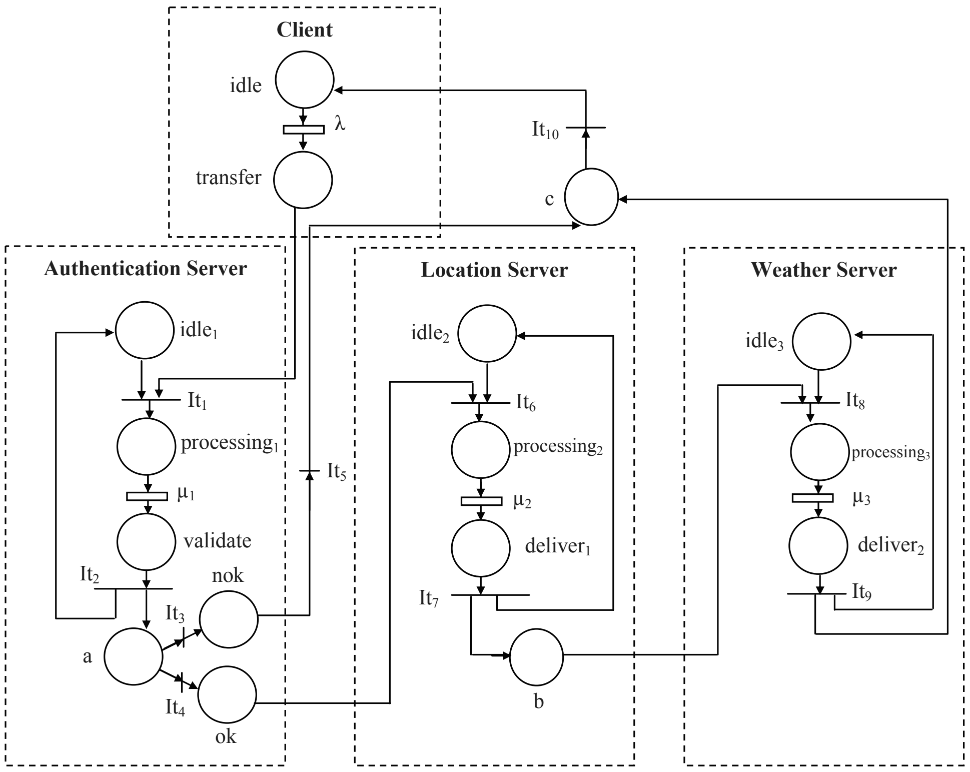


Figure 7. SPN model of our system while considering 1st design alternative of system architecture

The system starts by placing a token in the initial node of the client, when one request is generated by the client through his/her mobile terminal. The token is then transferred to the authentication server. The arrival rate of client request is mentioned by a timed transition parameter λ. If the authentication server is in idle state, when the client request will arrive the immediate transition It_1 will be fired, which will offer a token to the place *processing*$_1$ and the processing of the client request will be started immediately by the authentication server. The processing rate of the authentication server is mentioned by timed transition parameter μ_1. After completing the processing, token will be passed to the place *validate* and instantly the immediate transition It_2 will be fired. Firing of It_2 will create two flows. One flow will offer a token to the place *idle*$_1$ indicating that the authentication server is ready for serving the next request. Another flow will emit a token that will be forwarded either to the place *nok* or *ok* under the probabilistic assumption It_3 and It_4. If a token is placed in the node *nok* the immediate transition It_5 will be fired, which emits a token to notify the client immediately upon the cancellation of request because of the failing of verifying the client's authenticity. On the other hand, if a token is left to the place *ok* and if the location server is in idle state the immediate transition *It₆* will be fired, which will offer a token to the place *processing₂* and the location server will start processing of user request immediately. The processing rate of the location server is mentioned by a timed transition parameter μ_2. When the processing will be done by the location server a token will be placed in the node *deliver₁* and instantly, the immediate transition *It₇* will be fired. Firing of *It₇* will create two flows. One flow will offer a token to the place *idle₂* indicating that the location server is ready for serving the next request. Another flow will emit a token that will be immediately forwarded to the weather server. If the weather server remains in the idle state after getting the token from the location server the immediate transition *It₈* will be fired, which will offer a token in the place *processing₃.* This indicates the starting of the processing of the client request by the weather server immediately. The processing rate of the weather server is mentioned by a timed transition parameter μ_3. When the processing will be done by the weather server a token will be placed in the node *deliver₂* and instantly, immediate transition *It₉* will be fired. Firing of *It₉* will create two flows. One flow will then emit a token that will be immediately forwarded to the client as final result, which indicates the finishing of the processing of client request. Another flow will offer a token to the place *idle₃* indicating that the weather server is ready for serving next request. The processing of the client request will be delayed if any server remains busy just after arrival of client request. The value of the timed transition and the probabilistic assumption will be derived from annotated UML model shown in Figure 4.

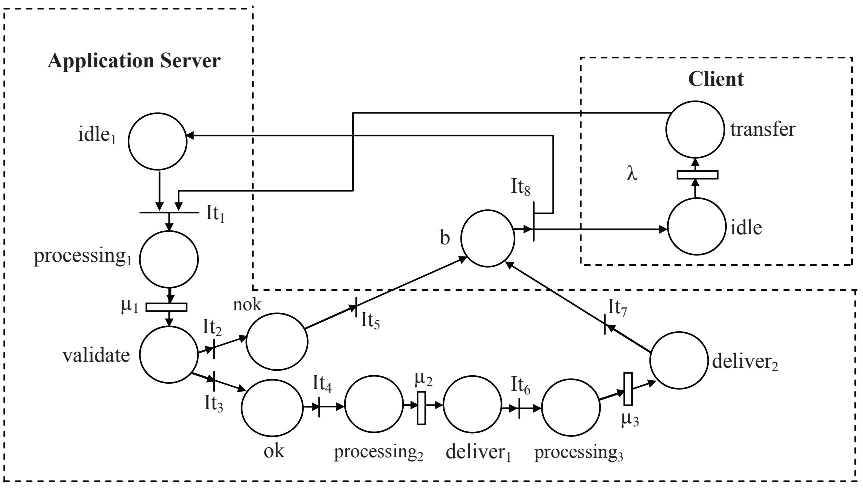


Figure 8. SPN model of our system while considering 2nd design alternative of system architecture

To generate the performance model for the 2nd variation of deployment diagram, we will consider the behavior of mobile terminal (as client) and application server as performance limiting factor of the system. The SPN performance model (Figure 8) is generated in the same way as mentioned above, where states are defined as places. We consider the number of application server in our system is 3, where server can process one request at a time. So the place Idle$_1$ in Figure 8 contains 3 tokens initially. In this case, there is one server, which is responsible for completing all the processing to serve user's requests.

(6) Evaluate Model: The generated model will be used as input for the SHARPE tool [9] to generate performance prediction result. The performance result generated by the tool is shown in a graph in Figure 9.

We modeled and solved the system for both design alternatives of system architecture. The performance prediction result such as mean response time of the system is derived against the increasing number of user requests for the same arrival rate of the client request and service rate of the servers for both design alternatives of system architecture. The comparison of the result is demonstrated in the graph, which clearly shows how the mean response time of the system varies with the increasing number of user requests because of considering the different system architecture candidates and deployment of the service components on physical components. It thus helps the system developer to

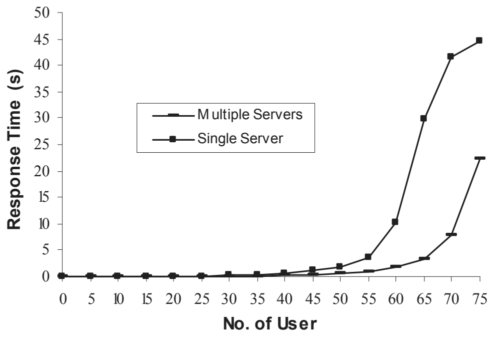


Figure 9. Performance evaluation result

resolve the bottleneck of system performance by finding a better allocation of service component on the physical component of the system at the early stage of the system development process.

4 Conclusion

Our main contribution is delineated as presenting the UML collaboration and activity oriented approach to capture the system dynamics that is utilized to sketch the performance model for distributed system, where each collaboration performs separate task. The behavior of the collaboration and the composition of collaboration to highlight the overall system behavior are demonstrated by utilizing UML activity. To present the behavior and composition of the collaboration using activity, we extend the notation to handle collaboration that is executed not only in the single session but also in multiple sessions at the same time, where different instances of collaborations are distinguished and coordinated by adding notation **select** to our execution profile. The **select** notation can outline the relations between multiple sessions unambiguously on an abstract level. Later on, a mapping between collaboration role and system component is outlined to show how the service of the distributed system is realized by the joint behavior of the system components that are physically distributed. Different variations of deployment diagram are considered to generate the performance model showing how the variations in the deployment diagram thus affect the system performance under different work load. However, the size of the underlying reachability set to generate SPN model is major limitation for large and complex systems. Further work includes automating the whole translation process from our UML specification style to generate a performance model and the way to solve the performance model through our proposed framework as well as to tackle the state explosion problems of reachability marking for large systems.

References

1. F. A. Kramer, R. Bræk, P. Herrmann, “Synthesizes components with sessions from collaboration-oriented service specifications”, Proceedings of SDL 2007, V-4745, Lecture notes of Computer Science, 2007.
2. OMG 2009, “UML Superstructure”, Version-2.2
3. OMG 2005, “UML Profile for Schedulability, Performance, and Time Specification”, V – 1.1
4. K. S. Trivedi, “Probability and Statistics with Reliability, Queuing and Computer Science application”, Wiley-Interscience publication, ISBN 0-471-33341-7
5. J. P. Lopez, J. Merseguer, J. Campos, “From UML activity diagrams to SPN: application to software performance engineering”, ACM SIGSOFT software engineering notes, NY, 2004
6. Pakke Kahkipru, “UML based performance modeling framework for object oriented distributed systems”, Proceedings of the 2nd international conference on the unified modeling language: beyond the standard, pp. 356-371, Springer-Verlag Berlin, Heidelberg, 1999
7. Abdullatif, R. Pooly, “A computer assisted state marking method for extracting performance models from design models ”, International Journal of Simulation, Vol. 8, No. 3, ISSN – 1473-804x
8. F. A. Kramer, “ARCTIS”, Department of Telematics, NTNU, http://arctis.item.ntnu.no.
9. K. S. Trivedi, R. Sahner, “SHARPE: Symbolic Hierarchical Automated Reliability / Performance Evaluator”, Duke University, Durham.
10. F. A. Kraemer, P. Herrmann, “Service specification by composition of collaborations- an example”, Proceedings of WI-IAT workshops, Hong Kong, p. 129-133, 2006
11. Linda Rising, “Design patterns in Communications Software”, Cambridge University Press, NY, USA, 2001.
12. R. H. Khan, P. E. Heegaard, “Translation from UML to SPN model: A performance modeling framework”, Accepted in Networked services and Application- Engineering, Control and Management (EUNICE), June 28-29, 2010, Trondheim, Norway.

Paper 4

A performance modeling framework incorporating cost efficient deployment of collaborating components

Razib Hayat Khan, Poul E. Heegaard

Presented in the International Conference on Software Technology and Engineering (ICSTE), Puerto Rico, USA, 2010

Published in the Proceedings of the International Conference on Software Technology and Engineering (ICSTE), Vol. 1, P. 340-349, IEEE Computer Society Press, 2010

A performance modeling framework incorporating cost efficient deployment of collaborating components

Razib Hayat Khan, Poul E. Heegaard

Department of Telematics
Norwegian University of Science and Technology (NTNU)
7491, Trondheim, Norway
{rkhan, poul.heegaard }@item.ntnu.no

Abstract-Performance evaluation of a distributed system is always an intricate undertaking, where system behavior is normally distributed among several components those are physically distributed. Bearing this concept in mind, we delineate a performance modeling framework for a distributed system that proposes a transformation process from high level UML notation to SPN model and solves the model for relevant performance metrics. To capture the system dynamics through our proposed framework, we outline a specification style that focuses on UML collaboration and activity as reusable specification building blocks, while deployment diagram identify the physical components of the system and the assignment of software artifacts to those identified system components. Optimal deployment mapping of the software artifacts on the physically available system resources is investigated by deriving the cost function. The proposed performance modeling framework provides transformation rules of UML elements into corresponding SPN representations and also the prediction result of a system such as throughput. The applicability of our proposed framework is demonstrated in the context of performance modeling of a distributed system.

1 Introduction

Distributed system poses one of the main streams of information and communication technology arena with immense complexity. Designing and implementation of such complex systems are always an intricate endeavor. Likewise, performance evaluation is also a great concern of such complex system to evaluate whether the system meets the performance related system requirements. Hence, modeling phase plays an important role in the whole design process of the system for qualitative and quantitative analysis. However, in a distributed system, system behavior is normally distributed among several objects. The overall behavior of the system is composed of the partial behavior of the distributed objects of the system. So it is obvious to capture the behavior of the distributed objects of the system for appropriate analysis to evaluate the performance related factors of the overall system. We therefore, adopt UML collaboration and activity oriented approach as UML is the most widely used modeling language, which models both the system requirements and qualitative behavior through different notations [2]. Collaboration and activity diagram are utilized to demonstrate the overall system behavior by defining both the structure of the partial object behavior as well as the interaction between them as reusable specification building blocks and later on, this UML specification style is applied to generate the SPN model by our proposed performance modeling framework. UML collaboration and activity provides a tremendous modeling framework containing several interesting properties. Firstly, collaborations and activity model the concept of service provided by the system very nicely. They define structure of the partial object behavior, the collaboration roles and enable a precise definition of the overall system behavior. They also delineate the way to compose the services by means of collaboration uses and role bindings [1].

In addition, the proposed modeling framework considers system execution architecture to realize the deployment of the service components. Abstract view of the system architecture is captured by the UML deployment diagram, which defines the execution architecture of the system by identifying the system components and the assignment of software artifacts to those identified system components [2]. Considering the system architecture to generate the performance model resolves the bottleneck of system performance by finding a better allocation of service components to the physical nodes. This needs for an efficient approach to deploy the service components on the available hosts of distributed environment to achieve preferably high performance and low cost levels. The most basic example in this regard is to choose better deployment architectures by considering only the latency of the service. The easiest way to satisfy the latency requirements is to identify and deploy the service components that require the highest volume of interaction onto the same resource or to choose resources that are at least connected by links with sufficiently high capacity and spread the work load to the available physical resources by maintaining equilibrium among them with respect to the execution cost [3].

The *Unified Modeling Language* (UML) is a widely accepted modeling language to model the system behavior [2]. But it is indispensable to extend the UML model to incorporate the performance-related quality of service (QoS) information to allow modeling and evaluating the properties of a system like throughput, utilization, and mean response time. So the UML models are annotated according to the *Profile for Schedulability, Performance, and Time* (SPT) to include quantitative system parameters [4]. Thus, it helps to maintain consistency between system design and implementation with respect to requirement specification.

Markov models, queuing networks, stochastic process algebras and stochastic petrinet (SPN) are probably the best studied performance modeling techniques [5]. Among all of them, we will choose stochastic petrinet as the performance model generated by our proposed framework for providing performance prediction result of a system due to its increasingly popular formalism for describing and analyzing systems, its modeling generality, its ability to capture complex system behavior concisely, its ability to preserve the original architecture of the system, to facilitate any modification according to the feedback from performance evaluation and the existence of analysis tools.

Several approaches have been followed to generate the performance model from system design specification. Lopez-Grao *et al.* proposed a conversion method from annotated UML activity diagram to stochastic petrinet model [6]. Kähkipuro developed a performance modeling framework to generate and solve queuing network with simultaneous resource possessions from the high level UML notations [7]. Zimmermann and Hommel presented a SPN model of communication failure and recover behavior of future European train control system with performance evaluation showing the significant impact of packet delays and losses on the reliable operation of high-speed trains [8]. However, most existing approaches do not highlight more on the issue that how to optimally conduct system modeling and performance evaluation. The framework presented here is the first known approach that introduces a new specification style utilizing UML behavioral diagrams as reusable specification building block, which is later on, used for generating performance model to produce performance prediction result at early stage of the system development process. Building blocks describe the local behavior of several components and the interaction between them. This provides the advantage of reusability of building blocks, since solution that requires the cooperation of several components may be reused within one self-contained, encapsulated building block. In addition, the resulting deployment mapping provided by our framework has greater impact with respect to QoS provided by the system. Our aim here is to deal with vector of QoS properties rather than restricting it in one dimension. Our presented deployment logic is surely able to handle any properties of the service, as long as we can provide a cost function for the specific property. The cost function defined here is flexible enough to keep pace with the changing size of search space of available host in the execution environment to ensure an efficient deployment of service components. Furthermore, we aim to be able to aid the deployment of several different services at the same time using the same proposed framework. The novelty of our approach is also reflected in showing the optimality of our solution with respect to both deployment logic and evaluation of performance metrics.

The objective of the paper is to provide an extensive performance modeling framework that provides a translation process to generate SPN performance model from system design specification captured by the UML behavioral diagram and later solves the model for relevant performance metrics to demonstrate performance prediction results at early stage of the system development life cycle. Incorporating cost functions to draw relation between service component and available physical resource permit us to identify an efficient deployment mapping in a fully distributed manner. The work presented here is the extension of our previous work described in [9] [10], where we presented our proposed framework with respect to the execution of single and multiple collaborative session at certain time and considered alternatives system architecture candidate to describe the system behavior and evaluate the performance factors. The paper is organized as follows: Section 2 introduces our proposed performance modeling framework, Section 3 demonstrates the application example to show the applicability of our modeling framework, Section 4 delineates conclusion with future works.

2 Performance modeling framework

Our proposed performance modeling framework utilizes the tool suite Arctis, which is integrated as plug-ins into the eclipse IDE [11]. The proposed framework is composed of 6 steps shown in Figure 1, where steps 1 and 2 are the parts of Arctis tool suite. Arctis focuses on the abstract, reusable service specifications that are composed form UML 2.2 collaborations and activities. It uses collaborative building blocks as reusable specification units to create comprehensive services through composition. To support the construction of building block consisting of collaborations and activities, Arctis offers special actions and wizards. In addition, a number of inspections ensures the syntactic consistency of building blocks. A developer first consults a library to check if an already existing collaboration block or a collaboration of several blocks solves a certain task. Missing blocks can also be created from scratch and stored in the library for later reuse. The building blocks are expressed as UML models. The structural aspect, for example the service component and their multiplicity, is expressed by means of UML 2.2 collaborations. For the detailed internal behavior, UML 2.2 activities have been used. They express the local behavior of each of the service components as well as their necessary interactions in a compact and self-contained way using explicit control flows [11]. Moreover the building blocks are combined into more comprehensive service by composition. For this composition, Arctis uses UML 2.2 collaborations and activities as well. While collaborations provide a good overview of the structural aspect of the composition, i.e., which sub-services are reused and how their collaboration roles are bound, activities express the detailed coupling of their respective behaviors [11].

1) Construction of collaborative building block: The proposed framework utilizes collaboration as main specification units. The specifications for collaborations are given as coherent, self-contained reusable building blocks. The structure of the building block

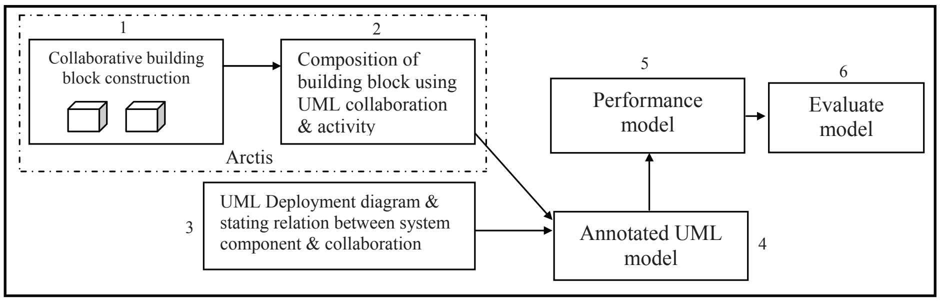

1
Collaborative building block construction
2
Composition of building block using UML collaboration & activity
Arctis
5
Performance model
6
Evaluate model
3
UML Deployment diagram & stating relation between system component & collaboration
Annotated UML model
4

Figure 1. Performance modeling framework

is described by UML 2.2 collaboration. If the building block is elementary it only declares the participants (as collaboration roles) and connection between them. If it is composite, it may additionally refer to other collaborations between the collaboration roles by means of collaboration uses. The internal behavior of building block is described by UML activity. It is declared as the classifier behavior of the collaboration and has one activity partition for each collaboration role in the structural description. For each collaboration use, the activity declares a corresponding call behavior action refereeing to the activities of the employed building blocks. For example, the general structure of the building block *t* is given in Figure 2(a), where it only declares the participants A and B as collaboration roles and the connection between them is defined as collaboration use t. The internal behavior of the same building block is shown in Figure 3(b). The activity $transfer_{AB}$ describes the behavior of the corresponding collaboration. It has one activity partition for each collaboration role: A and B. Activities base their semantics on token flow [1]. The activity starts by placing a token, when there is response to transfer by either participant A or B. The token is then transferred by the participant A to participant

B represented by the call behavior action *transfer$_A$* and/or participant B to Participant A represented by the *transfer$_B$* after completion of the processing by the collaboration role A and B. The composite structure of building *t* is shown in Figure 2(b), where it may additionally refer to other collaborations *s* between the collaboration roles by means of collaboration uses.

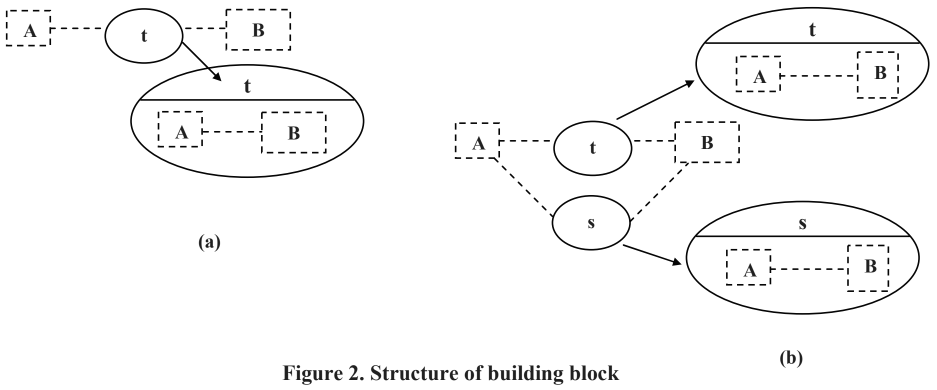
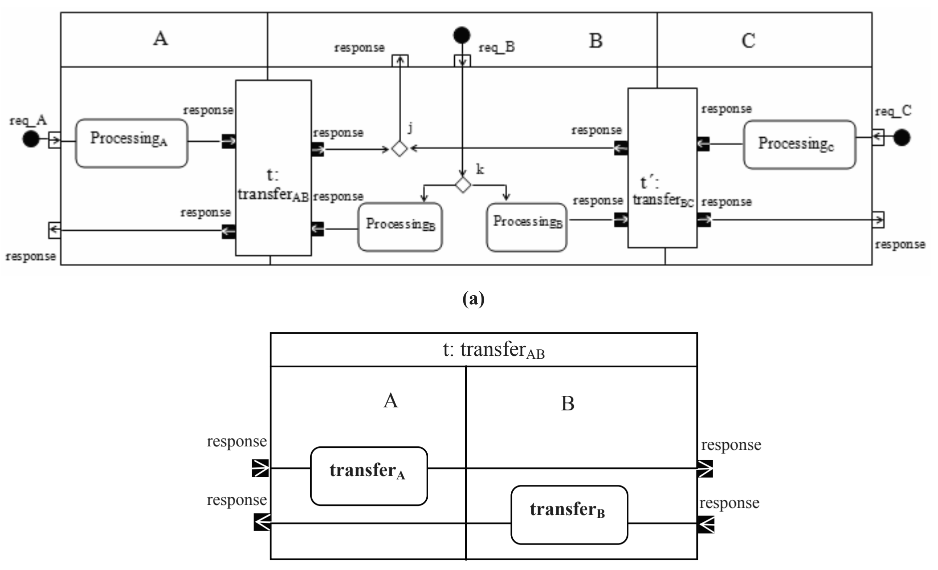


Figure 2. Structure of building block

2) Composition of building block using UML collaboration and activity: To generate the performance model, the structural information about how the collaborations are composed is not sufficient. It is necessary to specify the detailed behavior of how the different events of collaborations are composed so that the desired overall system behavior can be obtained. For the composition, UML collaborations and activities are used complementary to each other; UML collaborations focus on the role binding and structural aspect, while UML activities complement this by covering also the behavioral aspect for composition. For this purpose, call behavior actions are used. Each sub-service is represented by call behavior action referring 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 activity parameter nodes to represent them on the frame of a call behavior action. Arbitrary logic between pins may be used to synchronize the building block events and transfer data between them. By connecting the individual input and output pins of the call behavior actions, the events occurring in different collaborations can be coupled with each other. Semantics of the different kinds of pins are given in more detailed in [1]. For example, the detail behavior and composition of the collaboration is given in following Figure 3(a). The initial node (●) indicates the starting of the activity. The activity is started at the same time from the each participant A, B and C. The request for the execution of the task B is passed through a decision node *k* and only one flow is activated at the certain time instance. The request for the execution of the task A, B and C are mentioned by *req_A, req_B* and *req_C*. After getting the input, each participant starts its processing of request, which is mentioned by call behavior action *Processing$_A$, Processing$_B$* and *Processing$_C$*. After completing the processing, the responses are delivered to the corresponding participants. The response for the execution of the task A and C are transferred to B and the response for the execution of the task B is transferred to either A or C, which is mentioned by collaboration *t: transfer$_{AB}$* and *t′: transfer$_{BC}$*. There are two responses for B, which are connected by a merge node *j*, describing that outgoing flow is activated, when both or either of the incoming flow arrives.



(b)

Figure 3(a). Detailed behavior of the event of the collaboration using activity (b). Internal behavior of the collaboration

3) Designing UML deployment diagram and stating relation between system components and collaborations: To solve the deployment problem we identify two types of input required, profiles and provided property. Profile are used to specify the goals for the deployment logic, while provided properties determine the search space and are to be extracted from the net-map, which in turn describes the context or environment, where the service is being executed. We model the system as collection of N interconnected nodes. Our objective is to find a deployment mapping for this execution environment for a set of service components C available for deployment that comprises a service. The deployment mapping can be defined as (M : C → N) between a number of components instances c, onto nodes n. A component $c_i \in C$ can be a client process or a service process, while a node, $n \in N$ can be a transit node, e.g. a traditional IP router, a server node, which is capable of accommodating a service component, a client node, which is aggregation point for client components, or a mixed node that can accommodate both client and service components.

In order to get an overview of our service deployment concept, we consider the basic example in Figure 4. In the example, the service engineer has developed a service, *Service$_k$*, which consists of three software components *ServComp$_i$*, *i = 1,..,3*. The service is expected to be accessed by two distinguishable set of users *Client1* and *Client2*.

Thereafter, the designer must provide the second input for the decision logic, which is the *net-map*, specifying four possible types of nodes and the links. Generally, nodes can have different responsibilities, such as providing services (*S1*), relaying traffic (*R1*), accommodating clients (*C1*), or a mixture of these (*SC1*). Nodes accommodating the client side of a particular service are considered as an aggregation point for the clients accessing the given service. In case the components of a service are described as a traditional client-server model, properties of the client side such as the expected amount of clients, the expected service demand, etc. have to be taken into account within the same logic we employ for deployment mapping. Constraints that will heavily impact the optimal deployment can be assigned to nodes and links. In the case of links, constraints might appear as costs of using the particular link (*li*) for example. Constraints assigned to nodes of the net-map for instance can represent memory sizes restricting placement of component instances to a certain number at a place. Using the above specified input, the deployment logic provides an optimal deployment architecture taking into account the QoS requirements for the components providing the specified services. The deployment logic providing optimal deployment architecture is realized by the cost function F(M) that is used to evaluate the current suggestion in several iterations. Often, however, the components cannot be freely assigned to nodes but due to certain system constraints are restricted to particular ones. This can be based on policies given by the service provider (e.g. service level agreements of ISPs). Limitations can be further based on access restrictions as well as on the provided and requested capabilities (soft costs) and on capacity requirements (hard costs, e.g. bandwidth limitations). The cost function F(M) has to consider these limitations, which on the other hand, restrict the search space and, in consequence, support the efficiency of our deployment mapping. The evaluation of cost function F(M) is mainly influenced by our way of service definition. Service is defined in our approach as a collaboration of total E components labeled as c_i (where i = 1…. E) to be deployed and total K collaboration between them labeled as k_j, (where j = 1 … K). The execution cost of each service component can be labeled as f_{c_i} and the communication cost between the service components is labeled as f_{k_j} . Accordingly we only observe the total load ($\hat{l}_n$, n = 1…N) of a given deployment mapping at each node.

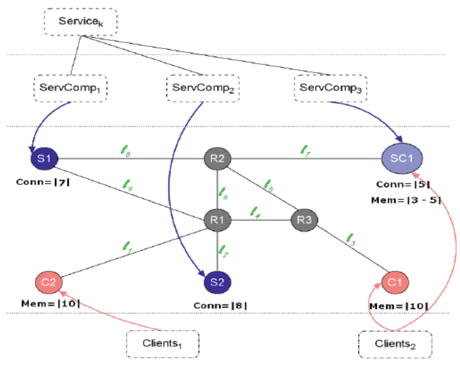


Figure 4. Component mapping example

The communication cost between two components is considered significant only if it appears between two separate nodes and we will strive for an optimal solution of equally distributed load among the processing nodes and the lowest cost possible, while taking into account the execution cost f_{c_i} , i = 1….E and communication cost f_{k_j} , j = 1….K. f_{c_i} and f_{k_j} are derived from the service specification, thus the offered execution load can be calculated as $\sum_{i=1}^{|E|} f_{c_i}$. This way, the logic can be aware of the target load [3]: $T = \frac{\sum_{i=1}^{|E|} f_{c_i}}{|N|}$

Given a mapping M = {m_n} (where m_n is the set of components at 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 [3]:

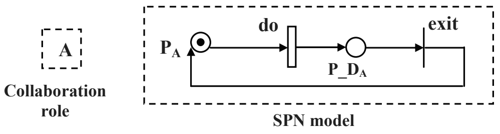
$$F(M) = \sum_{n=1}^{|N|} |\hat{l}_n - T| + \sum_{j=1}^{|K|} I_j f_{k_j} \tag{1}$$

Where $I_j = \begin{cases} 1, \text{ if } k_j \text{ external} \\ 0, \text{ if } k_j \text{ internal to a node} \end{cases}$

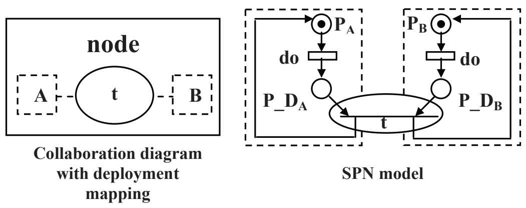
4) Annotating the UML model: Performance information is incorporated into the UML activity diagram and deployment diagram according to the *UML Profile for Schedulability, Performance and Time (SPT)* [4] for evaluating system performance by performance model solver.

5) Deriving the SPN model: Here, we define our rules for transforming into SPN model by utilizing the specification of reusable building blocks. By considering the internal behavior of the reusable building blocks (step 1), composition of different events of the building blocks (step 2), the relation between system component and collaboration and annotated UML structure, probable states and transition rate for triggering the change between states will be found based on which our SPN performance model will be generated. To generate the SPN model of the system, firstly, we generate the SPN model of the individual system components and later on, compose them together to generate the system level SPN model. The rules are based on decomposition of UML collaboration and activity diagram into basic elements of SPN model like states as places, timed transition and immediate transition. In addition, the rules are based on the rendezvous synchronization, which means, when communication between two processes of two interconnected nodes occur it follows the rendezvous synchronization [12]. Rendezvous provides synchronization between two threads while they communicate. In rendezvous synchronization, a synchronization and communication point called an entry is constructed as a function call. One process defines its entry and makes it public. Any process with knowledge of this entry can call it as an ordinary function call. The process that defines the entry accepts the call, executes it and returns the results to the caller. The issuer of the entry call establishes a rendezvous with the process that defined the entry [12]. The rules are following:

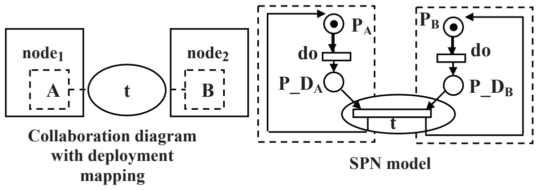
1 Basic SPN model for a collaboration role of a building block is defined by the two states such as *processing (P)* and *processing_done (P_D)* and the passing of token from state processing to processing done is realized by the timed transition, which is derived from the annotated UML model. One immediate transition is also associated from sate *processing_done* to state *processing* to indicate the ending of the activity of the collaboration role. For example, SPN model of collaboration role A in Figure 2 is shown in the following:



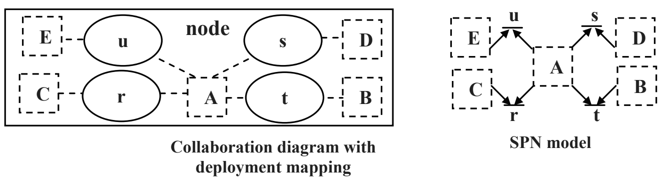
2 Collaboration can connect only two collaboration roles and replaced by an immediate transition while generating the SPN model, where collaboration roles deploy on the same physical node.



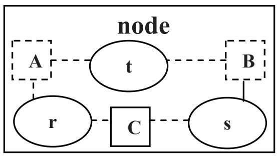
3 Collaboration can connect only two collaboration roles and replaced by a timed transition while generating the SPN model, where collaboration roles deploy on the different physical node.



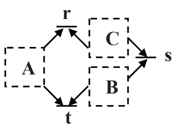
4 For a composite structure, if a collaboration role *A* connects with *n* collaboration roles by *n* collaboration uses like a star graph (where *n*>1), where each collaboration connects only two collaboration roles, then Only one instance of collaboration role A exists during the it's basic state transition and the single instance of collaboration role A connects with all other collaboration roles by immediate or timed transitions based on their deployment on the same or different physical components to generate the SPN model.



<u>**5**</u> If combinations of *n* collaboration roles (where *n*>2) create one or more circuits among them and each collaboration role satisfies the condition of rule 4 then only one instance of each collaboration role exists and connects with other collaboration roles by immediate or timed transitions based on their deployment on the same or different physical components to generate SPN model.

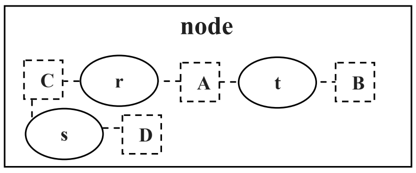


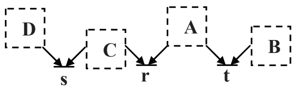
Collaboration diagram with deployment mapping



SPN model

<u>**6**</u> This rule is applicable for *n* collaboration roles arrange like a path of Graph (where *n*>3). Then only one instance of those collaboration roles exist that satisfies the condition of rule 4 and connects with other collaboration roles by immediate or timed transitions based on their deployment on the same or different physical components to generate SPN model.

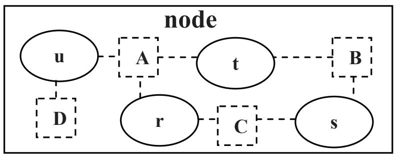




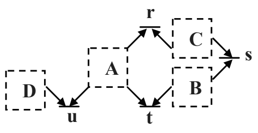
SPN model

Collaboration diagram with deployment mapping

<u>**7**</u> This rule is applicable for combinations of *n* collaboration roles those maintain both circuit and path like structure (where *n*>3). Then only one instance of those collaboration roles exist that satisfies the condition of rule 4 and connects with other collaboration roles by immediate or timed transitions based on their deployment on the same or different physical components to generate SPN model.



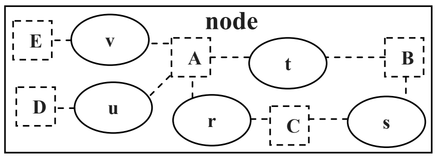
Collaboration diagram with deployment mapping



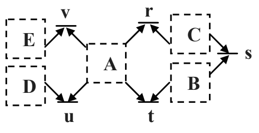
SPN model

<u>**8**</u> This rule is applicable for combinations of *n* collaboration roles those maintain both star and circuit like structure (where *n*>4). Then only one instance of those collaboration roles exist that satisfies the condition of rule 4 and connects with other collaboration roles

by immediate or timed transitions based on their deployment on the same or different physical components to generate SPN model.

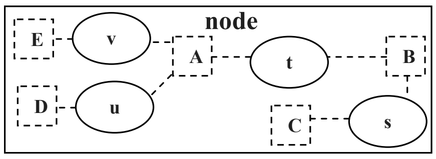


Collaboration diagram with deployment mapping

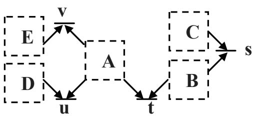


SPN model

9 This rule is applicable for combinations of *n* collaboration roles those maintain both star and path like structure (where *n*>4). Then only one instance of those collaboration roles exist that satisfies the condition of rule 4 and connects with other collaboration roles by immediate or timed transition based on their deployment on the same or different physical components to generate SPN model.

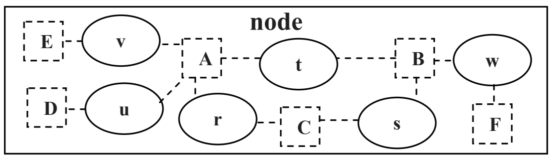


Collaboration diagram with deployment mapping

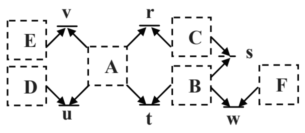


SPN model

10 This rule is applicable for combinations of *n* collaboration roles those maintain star, circuit and path like structure (where *n*>5). Then only one instance of those collaboration roles exist that satisfies the condition of rule 4 and connects with other collaboration roles by immediate or timed transitions based on their deployment on the same or different physical components to generate SPN model.



Collaboration diagram with deployment mapping



SPN model

6) Evaluate the model: We focus on measuring the throughput of the system from the developed model. Before deriving formula for throughput estimation we consider several assumptions in this regard. Firstly, when communication between two processes of two interconnected nodes occur it follows the rendezvous synchronization. Secondly, all the communications among the interconnected nodes occur in parallel. Moreover, the communications between interconnected nodes will be started following the completion of all the processing inside each physical node. By considering all the assumption we define the throughput as function of total expected number of jobs, E(N) and cost of the network, C_{Net}:

Throughput ~ f (E(N), C_{Net})

The value of E(N) is calculated by solving the SPN model using SHARPE [14]. The value of C_{Net} is evaluated by considering a subnet, which is the performance limiting factor of the whole network i.e., which posses maximum cost with respect to its own execution cost and communication cost with other subnet. Execution cost of the network, Exc_{net} is defined as:

$Exc_{net} = \sum_{i=1}^{|N|} exc\_subnet_i$; [*N* is the total no. of subnet that comprises whole network, exc_subnet_i = execution cost of ith subnet]

$exc\_subnet_i = \sum_{k=1}^{|M|} f_{c_k}$; [*M* = total number of service components deployed on the $subnet_i$, f_{c_k} = execution cost of the kth component of $subnet_i$]

$c\_subnet_i = exc\_subnet_i + \sum_{j=1}^{|k|} I_j f_{k_j}$; [$k \subseteq K$, c_subnet_i = cost of the ith subnet, j= 1…*k*; which defines the total number of collaborations in the ith subnet]

$= exc\_subnet_i$; [I_j = 0 in this case according to (1) as k_j internal to a node]

Now we evaluate the cost between each pair of subnet ($sbunet_a$ and $subnet_b$; where $(a,b) \in N$, a ≠ b) with respect to the subnet's own processing cost and the cost associated with the communication with other subnet in the network. Cost of a subnet pair, $C\_subnetp_y$ is defined as:

$C\_subnetp_y = \max\{\max\{c\_subnet_a, c\_subnet_b\} + I_j\, f_{k_j}\}$; [j= 1…*k*; which defines the total number of collaborations between $subnet_a$ and $subnet_b$, y=1…n, defines the total subnet pair in the network, I_j = 1 according to (1) as k_j external to the nodes]

$C_{Net} = \max\{C\_subnetp_1, \ldots, \ldots, C\_subnetp_n\}$;

$$\text{Throughput} = \frac{E(N)}{C_{Net}} \qquad (2)$$

3 Application example

As a representative example, we consider the scenario originally from Efe dealing with heuristically clustering of modules and assignment of clusters to nodes [13]. This scenario, even though artificial and may not be tangible from a designer's point of view, is sufficiently complex to show the applicability of our proposed framework. The problem is defined in our approach as a service of collaboration of E = 10 components or

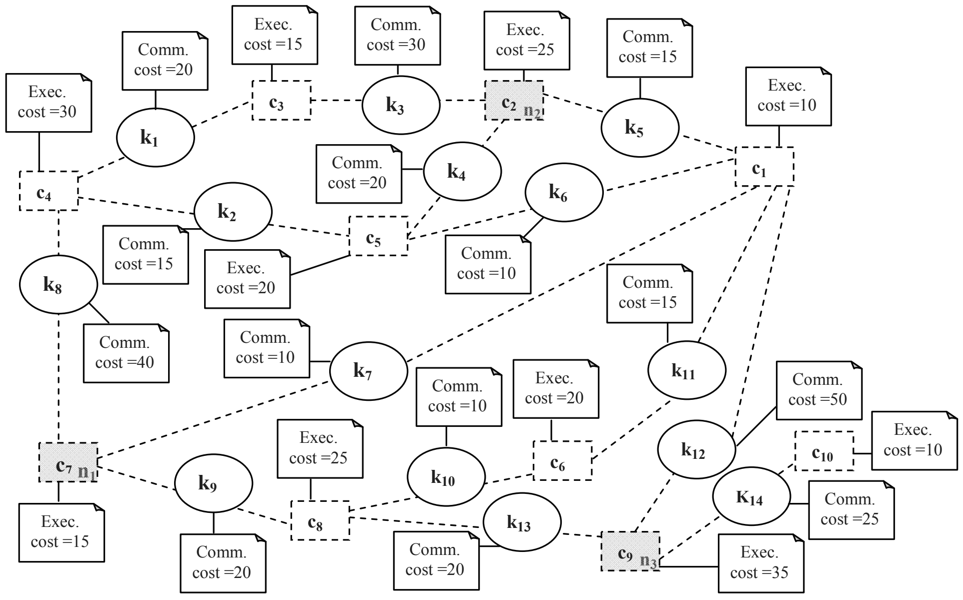


Figure 5. Collaborations and components in the example scenario

collaboration roles (labeled $c_1 \dots c_{10}$) to be deployed and K = 14 collaborations between them depicted in Figure 5. We consider three types of requirements in this specification. Besides the execution and communication costs, we have a restriction on components c_2, c_7, c_9 regarding their location. They must be bound to nodes n_2, n_1, n_3 respectively. The internal behavior of the collaboration K_i and composition of the collaboration role c_i as reusable building block is realized by the call behavior action through the same UML activity diagram already demonstrated in Figure 3(b) and Figure 3(a). The call behavior action is later on, utilized as the states of the performance model.

In this example, the target environment consists only of N = 3 identical, interconnected nodes with a single provided property, namely processing power and with infinite communication capacities depicted in Figure 6. The optimal deployment mapping can be observed in Table 1. The lowest possible deployment cost, according to equation (1) is 17 + 100 = 117.

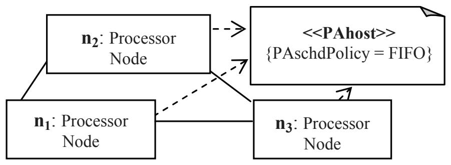


Figure 6. The target network of hosts

To annotate the UML diagram in Figure 5 and 6 we use the stereotypes RTaction, PAhost and the tagged values RTduration depicted in Table 2. RTaction models any action that takes time. The duration of the time is mentioned by the tagged value RTduration. A PAhost models a processing resource with tagged PAschdPolicy defining the policy by which access to the resource is controlled. The annotation of service components and collaboration of Figure 5 is shown in Table 2.

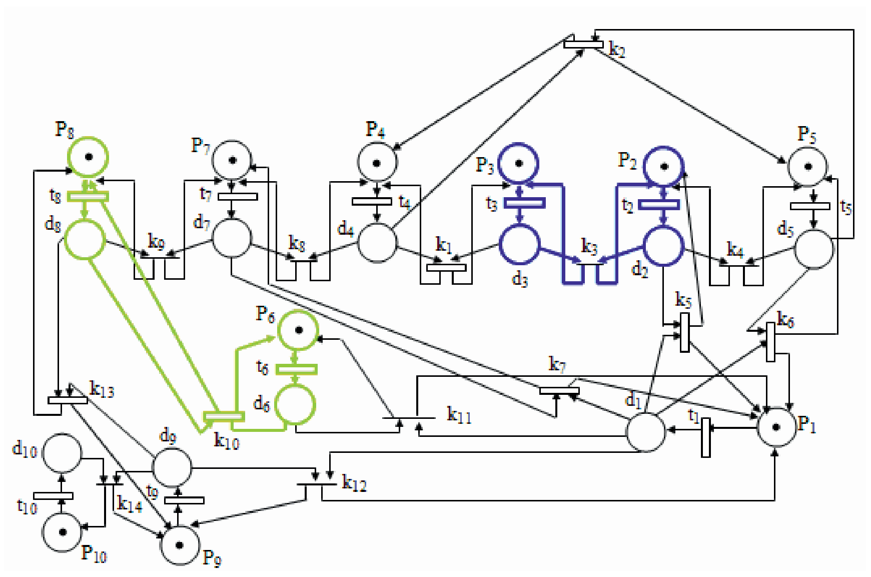
Table 1. Optimal deployment mapping in the example scenario

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

Table 2. Annotating UML model according to SPT

C_1	<<RTaction>> {RTduration=10, s}		**K_1**	<<RTaction>> {RTduration=20, s}

By considering the above deployment mapping and the transformation rule the analogous SPN model of our example scenario is depicted in Figure 7. The states of the SPN model are derived from the call behavior action of the corresponding collaboration role and collaboration among them, where p_i and d_i stand for processing$_i$ and done$_i$ of the i^{th} service components. 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 t_i (stands for transition$_i$ for the i^{th} service component), which is derived from the annotated UML model. For generating the SPN model, firstly, we will consider the collaboration roles deployed on the processor node $\mathbf{n_1}$, which are c_4, c_7 and c_8. Components c_7 are connected with c_4 and c_8 and maintains star graph like structure. So according to rule 4, only one instance of collaboration role c_7 is existed during the basic state transition (p_7 and d_7) and connects with states of the components c_4 (p_4 and d_4) and c_8 (p_8 and d_8)with immediate transition k_8 and k_9 to generate the SPN model. Collaboration roles c_2, c_3 and c_5 deploy on the processor node $\mathbf{n_2}$. Components c_2 are connected with c_3 and c_5 and maintains star graph like structure. So according to rule 4, only one instance of collaboration role c_2 is existed during the basic state transition (p_5 and d_5) and connects with states of the components c_3 (p_3 and d_3) and c_5 (p_5 and d_5)with immediate transitions k_3 and k_4 to generate the SPN model. Collaboration roles c_6, c_1, c_9 and c_{10} deploy on the processor node **n_3**. The components arrange like a path of graph. So according to rule 6, one instance of component c_1 and c_9 (as c_1 and c_9 satisfy the condition of rule 4) is presented during the basic state transition and state d_1 and d_9 is connected with immediate transition k_{12}, state d_1 is connected with the d_6 (state of component c_6) by the immediate transition k_{11} and state d_9 is connected with the d_{10} (state of component c_{10}) by the immediate transition k_{14}. In order to generate the system level SPN model, we need to compose the entire three SPN models generated for three processor nodes by considering the interconnection among them. To compose the SPN models of processor node n_1 and n_2, state d_4 and d_3 connect by the timed transition k_1 and state d_4 and d_5 connect by the timed transition k_2 according to rule 2. Likewise, to compose the SPN models of processor node n_2 and n_3, state d_2 and d_1 connect by the timed transition k_5 and state d_5 and d_1 connect by the timed transition k_6 according to rule 2. To compose the SPN models of processor node n_1 and n_3, state d_7 and d_1 connect by the timed transition k_7, state d_8 and d_6 connect by the timed transition k_{10} and state d_8 and d_9 connect by the timed transition k_{13} according to rule 2. By the above way, the system level SPN model is derived.



Reusable building block where collaboration roles 🟩 deploy on the different physical node	Reusable building block where collaboration roles 🟦 deploy on the same physical node

Figure 7. SPN model of the example scenario

The throughput calculation according to equation (2) for the different deployment mapping including the optimal deployment mapping is shown in Table 3. The throughput is $0.0723s^{-1}$ while considers the optimal deployment mapping, where E(N) = 6.877 (calculated using SHARPE [14]) and C_{Net} = 95s. The optimal deployment mapping according to equation (2) also show the optimality in case of throughput calculation. We will not present here the throughput calculation of all the deployment mapping of the software artifacts but obviously the approach presented here confirms the efficiency in both deployment mapping and throughput calculation.

Table 3. Possible cost and throughput for different deployment mapping

Node	Components	Possible cost (s)	Throughput (s^{-1})
$\{n_1, n_2, n_3\}$	$\{\{c_4, c_7, c_8\}, \{c_2, c_3, c_5\}, \{c_1, c_6, c_9, c_{10}\}\}$	117	0.0723
$\{n_1, n_2, n_3\}$	$\{\{c_4, c_7\},\{c_2, c_3, c_5, c_6\}, \{c_1, c_8, c_9, c_{10}\}\}$	162	0.0633
$\{n_1, n_2, n_3\}$	$\{\{c_4, c_5, c_7, c_8\}, \{c_2, c_3\}, \{c_1, c_6, c_9, c_{10}\}\}$	162	0.0631
$\{n_1, n_2, n_3\}$	$\{\{c_5, c_7, c_8\}, \{c_2, c_3, c_4\}, \{c_1, c_6, c_9, c_{10}\}\}$	157	0.0623
$\{n_1, n_2, n_3\}$	$\{\{c_4, c_6, c_7, c_8\}, \{c_2, c_3, c_5\}, \{c_1, c_9, c_{10}\}\}$	148	0.0609
$\{n_1, n_2, n_3\}$	$\{\{c_1, c_6, c_7, c_8\}, \{c_2, c_3, c_4\}, \{c_5, c_9, c_{10}\}\}$	177	0.0588
$\{n_1, n_2, n_3\}$	$\{\{c_4, c_7, c_8\}, \{c_1, c_2, c_3, c_5\}, \{c_6, c_9, c_{10}\}\}$	147	0.0568
$\{n_1, n_2, n_3\}$	$\{\{c_1, c_6, c_7, c_8\}, \{c_2, c_3, c_5\}, \{c_4, c_9, c_{10}\}\}$	187	0.0553
$\{n_1, n_2, n_3\}$	$\{\{c_3, c_6, c_7, c_8\}, \{c_1, c_2, c_4, c_5\}, \{c_9, c_{10}\}\}$	218	0.0488
$\{n_1, n_2, n_3\}$	$\{\{c_6, c_7, c_8\}, \{c_1, c_2, c_4, c_5\}, \{c_3, c_9, c_{10}\}\}$	232	0.0488

4 Conclusion

We present a novel approach for model based performance evaluation of distributed system, which spans from capturing the system dynamics through UML diagram as reusable building block to efficient deployment of service components in a distributed manner by capturing the QoS requirements. System dynamics is captured through UML collaboration and activity oriented approach. Furthermore, quantitative analysis of the system is achieved by generating SPN performance model from the UML specification style. The transformation from UML diagram to corresponding SPN elements like states, different pseudostates and transitions is proposed. Performance related QoS information is taken into account and included in the SPN model with equivalent timing and probabilistic assumption for enabling the evaluation of performance prediction result of the system. In addition, the logic, as it is presented here, is applied to provide the optimal, initial mapping of components to hosts, i.e. the network is considered rather static. However, our eventual goal is to develop support for run-time redeployment of components, this way keeping the service within an allowed region of parameters defined by the requirements. As the results with our proposed framework show our logic will be a prominent candidate for a robust and adaptive service execution platform.

References

[1] F. A. Kramer, R. Bræk, P. Herrmann, “Synthesizes components with sessions from collaboration-oriented service specifications”, Proceedings of SDL 2007, V-4745, LNCS

[2] OMG 2009, "UML Superstructure", Version-2.2
[3] M. Csorba, P. Heegaard, P. Herrmann, "Cost-Efficient Deployment of Collaborating Components", DAIS 2008
[4] OMG 2005, "UML Profile for Schedulability, Performance, and Time Specification", V – 1.1
[5] K. S. Trivedi, "Probability and Statistics with Reliability, Queuing and Computer Science application", Wiley-Interscience publication, ISBN 0-471-33341-7
[6] J. P. Lopez, J. Merseguer, J. Campos, "UML activity diagrams to SPN: application to software performance engineering", ACM SIGSOFT software engineering notes, NY, 2004
[7] Pakke Kahkipru, "UML based performance modeling framework for object oriented distributed systems", Proceedings of the 2nd international conference on the unified modeling language: beyond the standard, pp. 356-371, Springer-Verlag Berlin, Heidelberg, 1999
[8] J. Trowitzsch, A. Zimmermann, "Using UML State Machines and Petri Nets for the Quantitative Investigation of ETCS", Valuetools, October 11-13, 2006, Pisa, Italy
[9] R. H. Khan, P. E. Heegaard, "Translation from UML to SPN model: A performance modeling framework", Proceeding of the EUNICE, Springer, Norway, 2010.
[10] R H Khan, P Heegaard, "Translation from UML to SPN model: Performance modeling framework for managing behavior of multiple session and instance" Proceedings of the ICCDA, IEEE computer society, China, 2010
[11] F. A. Kramer, "ARCTIS", Department of Telematics, NTNU,
[12] Rendezvoussynchronization,http://book.opensourceproject.org.cn/embedded/cmprealtime/ opensource/ 5107final/lib0091.html, retrieved June, 2010
[13] Efe, K.: Heuristic models of task assignment scheduling in distributed systems. Computer (June 1982)
[14] k. Trivedi, R. Sahner, "Symbolic Hierarchical Automated Reliability / Performance Evaluator (SHARPE)", Duke University, Durham, NC, 2002

Paper 5

A performance modeling framework incorporating cost efficient deployment of multiple collaborating instances

Razib Hayat Khan, Poul E. Heegaard

Presented in the International Conference on Software Engineering & Computer System (ICSECS), Pahang, Malaysia, 2011

Published in the Communications in Computer & Information Science, Vol. 179, p. 31-45, Springer-Verlag Berlin Heidelberg, 2011

A performance modeling framework incorporating cost efficient deployment of multiple collaborating instances

Razib Hayat Khan, Poul E. Heegaard

Department of Telematics
Norwegian University of Science and Technology (NTNU)
7491, Trondheim, Norway
e-mail: {rkhan, poul.heegaard }@item.ntnu.no

Abstract- Performance evaluation of a distributed system is always an intricate undertaking, where system behavior is distributed among several components those are physically distributed. Bearing this concept, we delineate a performance modeling framework for a distributed system that proposes a transformation process from high level UML notation to SRN model and solves the model for relevant performance metrics. To capture the system dynamics through our proposed framework we outline a specification style that focuses on UML collaboration and activity as reusable specification building blocks, while deployment diagram identify the physical components of the system and the assignment of software artifacts to identified system components. Optimal deployment mapping of software artifacts on the available physical resources of the system is investigated by deriving the cost function. The way to deal with parallel thread processing of the network nodes by defining the upper bound is precisely mentioned to generate the SRN model.

1 Introduction

Modeling phase plays an important role in the whole design process of the distributed system for qualitative and quantitative analysis. However in a distributed system, system behavior is normally distributed among several objects. The overall behavior of the system is composed of the partial behavior of the distributed objects of the system. So it is obvious to capture the behavior of the distributed objects for appropriate analysis to evaluate the performance related factors of the overall system. We therefore adopt UML collaboration and activity oriented approach as UML is the most widely used modeling language, which models both the system requirements and qualitative behavior through different notations [2]. Collaboration and activity diagram are utilized to demonstrate the overall system behavior by defining both the structure of the partial object behavior as well as the interaction between them as reusable specification building blocks and later on, this UML specification style is applied to generate the SRN model by our proposed performance modeling framework. UML collaboration and activity provides a tremendous modeling framework containing several interesting properties. Firstly, collaborations and activity model the concept of service provided by the system very nicely. They define structure of the partial object behavior, the collaboration roles and enable a precise definition of the overall system behavior. They also delineate the way to compose the services by means of collaboration uses and role bindings [1]. The proposed modeling framework considers system execution architecture to realize the deployment of the service components. Considering the system architecture to generate the performance model resolves the bottleneck of system performance by finding a better allocation of service components to the physical nodes. This needs for an efficient approach to deploy the service components on the available hosts of distributed environment to achieve preferably high performance and low cost levels. The most basic example in this regard is to choose better deployment architectures by considering only the latency of the service. The easiest way to satisfy the latency requirements is to identify and deploy the service components that require the highest volume of interaction onto the same resource or to choose resources that are connected by links with sufficiently high capacity [3].

It is indispensable to extend the UML model to incorporate the performance-related quality of service (QoS) information to allow modeling and evaluating the properties of a system like throughput, utilization and mean response time. So the UML models are annotated according to the *UML profile for MARTE: Modeling and Analysis of Real-Time Embedded Systems* to include quantitative system parameters [4].

We will focus on the stochastic reward net (SRN) [5] as the performance model generated by our proposed framework due to its increasingly popular formalism for describing and analyzing systems, its modeling generality, its ability to capture complex system behavior concisely, its ability to preserve the original architecture of the system, to allow marking dependency firing rates and reward rates defined at the net level, to facilitate any modification according to the feedback from performance evaluation and the existence of analysis tools.

Several approaches have been followed to generate the performance model from system design specification. However, most existing approaches [6] [7] [8] do not highlight more on the issue that how to optimally conduct the system modeling and performance evaluation. The framework presented here is the first known approach that introduces a new specification style utilizing UML behavioral diagrams as reusable specification building block, which is later on, used for generating performance model to produce performance prediction result at early stage of the system development process. Building blocks describe the local behavior of several components and the interaction between them. This provides the advantage of reusability of building blocks, since solution that requires the cooperation of several components may be reused within one self-contained, encapsulated building block. In addition, the resulting deployment mapping provided by our framework has great impact with respect to QoS provided by the system. Our aim here is to deal with vector of QoS properties rather than restricting it in one dimension. Our presented deployment logic is surely able to handle any properties of the service, as long as we can provide a cost function for the specific property. The cost function defined here is flexible enough to keep pace with the changing size of search space of available host in the execution environment to ensure an efficient deployment of service components. Furthermore, we aim to be able to aid the deployment of several different services at the same time using the same proposed framework. The novelty of our approach is also reflected in showing the optimality of our solution with respect to both deployment logic and evaluation of performance metrics.

The objective of the paper is to provide an extensive performance modeling framework that provides a translation process to generate SRN performance model from system design specification captured by the UML behavioral diagram and later solves the model for relevant performance metrics. To incorporate the cost function to draw relation between service component and available physical resources permit us to identify an efficient deployment mapping in a fully distributed manner. The way to deal with parallel thread processing of the network node by defining the upper bound is precisely mentioned while generating the SRN model through the proposed framework. The work presented here is the extension of our previous work described in [9] [10] [14], where we presented our proposed framework with respect to the execution of single and multiple collaborative sessions and considered alternatives system architecture candidate to describe the system behavior and evaluate the performance factors. The paper is organized as follows: Section 2 introduces our proposed performance modeling framework, Section 3 demonstrates the application example to show the applicability of our modeling framework, Section 4 delineates conclusion.

2 Performance modeling framework

The framework is composed of 6 steps shown in Figure 1, where steps 1 and 2 are the parts of Arctis tool suite [11]. Arctis focuses on the abstract, reusable service

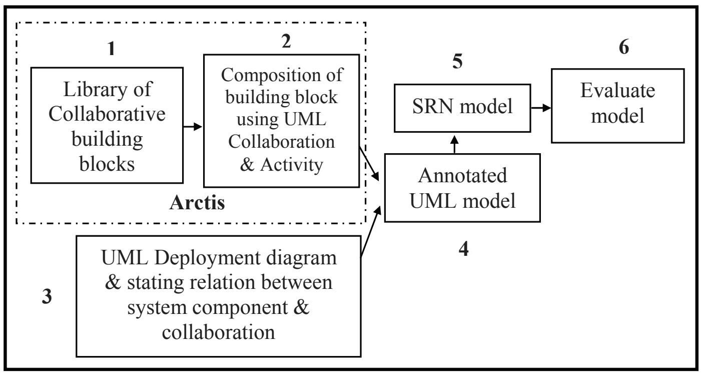

1
Library of Collaborative building blocks
2
Composition of building block using UML Collaboration & Activity
Arctis
3
UML Deployment diagram & stating relation between system component & collaboration
4
Annotated UML model
5
SRN model
6
Evaluate model

Figure 1. Performance modeling framework

specifications that are composed form UML 2.2 collaborations and activities. It uses collaborative building blocks as reusable specification units to create comprehensive services through composition. To support the construction of building block consisting of collaborations and activities, Arctis offers special actions and wizards. In addition, a number of inspections ensure the syntactic consistency of building blocks. A developer first consults a library to check if an already existing collaboration block or a collaboration of several blocks solves a certain task. Missing blocks can also be created from scratch and stored in the library for later reuse. The building blocks are expressed as UML models. The structural aspect, for example the service component and their multiplicity, is expressed by means of UML 2.2 collaborations. For the detailed internal behavior, UML 2.2 activities have been used. They express the local behavior of each of the service components as well as their necessary interactions in a compact and self-contained way using explicit control flows [11]. Moreover, the building blocks are combined into more comprehensive service by composition. For this composition, Arctis uses UML 2.2 collaborations and activities as well. While collaborations provide a good overview of the structural aspect of the composition, i.e., which sub-services are reused and how their collaboration roles are bound, activities express the detailed coupling of their respective behaviors [11]. The steps are illustrated below:

1) Construction of collaborative building block: The proposed framework utilizes collaboration as main specification units. The specifications for collaborations are given as coherent, self-contained reusable building blocks. The structure of the building block is described by UML 2.2 collaboration. The building block declares the participants (as collaboration roles) and connection between them. The internal behavior of building block is described by UML activity. It is declared as the classifier behavior of the collaboration and has one activity partition for each collaboration role in the structural description. For each collaboration use, the activity declares a corresponding call behavior action refereeing to the activities of the employed building blocks. For example, the general structure of the building block *t* is given in Figure 2(a), where it only declares the participants A and B as collaboration roles and the connection between them is defined as collaboration use t_x (x=1…n_{AB} (number of collaborations between collaboration roles A and B)). The internal behavior of the same building block is shown in Figure 2(b). The activity $transfer_{ij}$ (*where ij* = AB) describes the behavior of the corresponding collaboration. It has one activity partition for each collaboration role: A and B. Activities base their semantics on token flow [1]. The activity starts by placing a token, when there is a response (indicated by the streaming pin *res*) to transfer by either participant A or B.

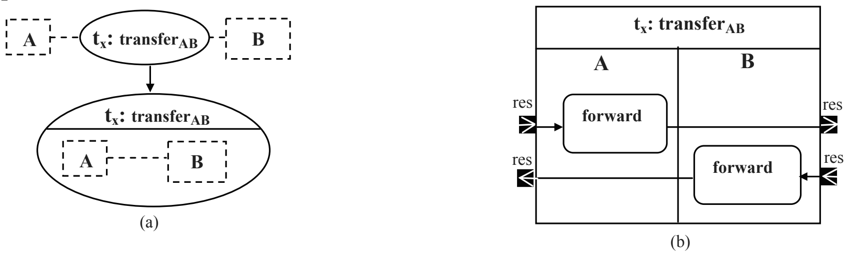


Figure 2(a). Structure of the building block using collaboration diagram (b). Internal behavior of the building block using activity diagram

After completion of the processing by the collaboration role A and B the token is transferred from the participant A to participant B and from participant B to Participant A, which is represented by the call behavior action *forward.*

2) Composition of building block using UML collaboration and activity: To generate the performance model, the structural information about how the collaborations are composed is not sufficient. It is necessary to specify the detailed behavior of how the different events of collaborations are composed so that the desired overall system behavior can be obtained. For the composition, UML collaborations and activities are used complementary to each other; UML collaborations focus on the role binding and structural aspect, while UML activities complement this by covering also the behavioral aspect for composition. For this purpose, call behavior actions are used. Each sub-service is represented by call behavior action referring the respective activity of building blocks. Each call behavior action represents an instance of a building block. For each activity parameter node of the referred activity, a call behavior action declares a corresponding pin. Pins have the same symbol as activity parameter nodes to represent them on the frame of a call behavior action. Arbitrary logic between pins may be used to synchronize

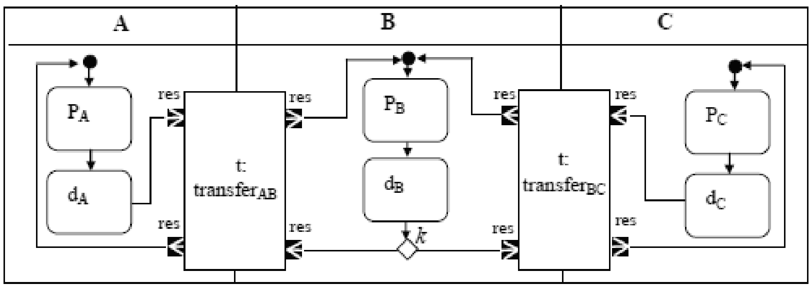


Figure 3. System activity to couple the collaboration

the building block events and transfer data between them. By connecting the individual input and output pins of the call behavior actions, the events occurring in different collaborations can be coupled with each other. Semantics of the different kinds of pins are given in more detailed in [1]. For example the detailed behavior and composition of the collaboration is given in following Figure 3. The initial node (●) indicates the starting of the activity. The activity is started at the same time from each participant. After being activated, each participant starts its processing of the request, which is mentioned by call behavior action P_i (Processing$_i$, where i = A, B, and C). Completions of the processing by the participants are mentioned by the call behavior action d_i (Processing_done$_i$, i = A, B and C). After completion of the processing, the responses are delivered to the corresponding participants indicated by the streaming pin *res*. When the processing of the execution of the task by the participant B completes the result is passed through a decision node *k* and only one flow is activated at the certain time instance. The response of the collaboration role A and C are forwarded to B and the response of collaboration role B is forwarded to either A or C, which is mentioned by collaboration *t: transfer$_{ij}$* (where ij = AB or BC).

3) Designing UML deployment diagram and stating relation between system components and collaborations: We model the system as collection of N interconnected nodes. Our objective is to find a deployment mapping for execution environment for a set of service components C available for deployment that comprises service. Deployment mapping can be defined as (M:C→N) between a numbers of service components instances c, onto physical nodes n. Components can communicate via a set of collaborations. We consider four types of requirements in the deployment problem. Components have execution costs, collaborations have communication costs and costs for running of background process and some of the components can be restricted in the deployment mapping to specific nodes, which are called bound components. Furthermore, we consider identical nodes that are interconnected in a full-mesh and are capable of hosting components with unlimited processing demand. We observe the processing load that nodes impose while host the components and also the target balancing of load between the nodes available in the network. By balancing the load the deviation from the global average per node execution cost will be minimized. Communication costs are considered if collaboration between two components happens remotely, i.e. it happens between two nodes [3]. In other words, if two components are placed onto the same node the communication cost between them will not be considered. The cost for executing the background process for conducting the communication between the collaboration roles is always considerable no matter whether the collaboration roles deploy on the same or different nodes. Using the above specified input, the deployment logic provides an optimal deployment architecture taking into account the QoS requirements for the components providing the specified services. We then define the objective of the deployment logic as obtaining an efficient (low-cost, if possible optimum) mapping of component onto the nodes that satisfies the requirements in reasonable time. The deployment logic providing optimal deployment architecture is guided by the cost function F(M). The evaluation of cost function F(M) is mainly influenced by our way of service definition. Service is defined in our approach as a collaboration of total E components labeled as c_i (where i = 1…. E) to be deployed and total K collaboration between them labeled as k_j, (where j = 1 … K). The execution cost of each service component can be labeled as 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 only observe the total load ($\widehat{l_n}$, n = 1…N) of a given deployment mapping at each node. We will strive for an optimal solution of equally distributed load among the processing nodes and the lowest cost possible, while taking into account the execution cost 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 load can be calculated as $\sum_{i=1}^{|E|} f_{c_i}$. This way, the logic can be aware of the target load [3]: $T = \frac{\sum_{i=1}^{|E|} f_{c_i}}{|N|}$ (1)

Given a mapping M = {m_n} (where m_n is the set of components at 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 (where I_j = 1, if k_j external or 0 if k_j internal to a node):

$$F(M) = \sum_{n=1}^{|N|} |\hat{l}_n - T| + \sum_{j=1}^{|K|} (I_j f_{k_j} + f_{B_j}) \tag{2}$$

4) Annotating the UML model: Performance information is incorporated into the UML activity diagram and deployment diagram according to *UML profile for MARTE: Modeling and Analysis of Real-Time Embedded Systems* [4] for evaluating system performance by performance model solver.

5) Deriving the SRN model: To generate the SRN model of the system, firstly, we generate the SRN model of the individual system components and later compose them together to generate the system level SRN model. The rules are based on decomposition of UML collaboration, activity and deployment diagram into basic elements of SRN model like states as places, timed transition and immediate transition. In addition, the rules are based on the rendezvous synchronization, which means, when communication between two processes of two interconnected nodes occur it follows the rendezvous synchronization [12].

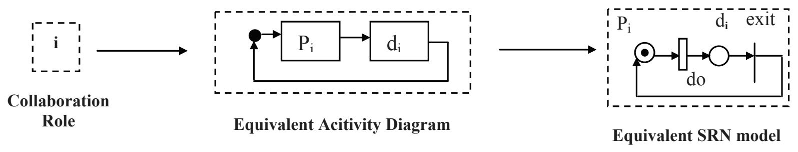
SRN model of the collaboration role of a reusable building block is mentioned by the 6-tuple {**Φ, T, A, K, N, $\mathbf{m_0}$**} in the following way [5]:

Φ = Finite set of the places (drawn as circles)
T = Finite set of the transition (drawn as bars)
A $\subseteq \{\Phi \times T\} \cup \{T \times \Phi\}$ is a set of arcs connecting Φ and T
K: T → {Timed (time>0, drawn as thick transparent bar), Immediate (time = 0, drawn as thin bar)} specifies the type of the each transition
N: A→ {1, 2, 3…} is the multiplicity associated with the arcs in A
m: Φ → {0, 1, 2...} is the marking that denotes the number of tokens for each place in Φ. The initial marking is denoted as $\mathbf{m_0}$.

The rules are the following:

Rule 1: The SRN model of the collaboration role of a reusable building block is represented by the 6-tuple in the following way:
$\mathbf{\Phi_i} = \{P_i, d_i\}$; $\mathbf{T} = \{do, exit\}$
$\mathbf{A} = \{\{(P_i \times do) \cup (do \times d_i)\}, \{(d_i \times exit) \cup (exit \times P_i)\}\}$
$\mathbf{K} = (do \rightarrow Timed, exit \rightarrow Immediate)$
$\mathbf{N} = \{(P_i \times do) \rightarrow 1, (do \times d_i) \rightarrow 1, (d_i \times exit) \rightarrow 1, (exit \times P_i) \rightarrow 1\}$
$\mathbf{m_o} = \{(P_i \rightarrow 1\}, (d_i \rightarrow 0)\}$.

Here, places are represented by P_i and d_i. Transitions are represented by *do* and *exit,* where *do* is a timed transition and *exit* is an immediate transition. Initially place P_i contains one token and place d_i contains no token. SRN model of the collaboration role is graphically represented by the following way:



Rule 2: The SRN model of a collaboration, where collaboration connects only two collaboration roles is represented by the 6-tuple in the following way:

Φ = $\{\Phi_i, \Phi_j\} = \{P_i, d_i, P_j, d_j\}$

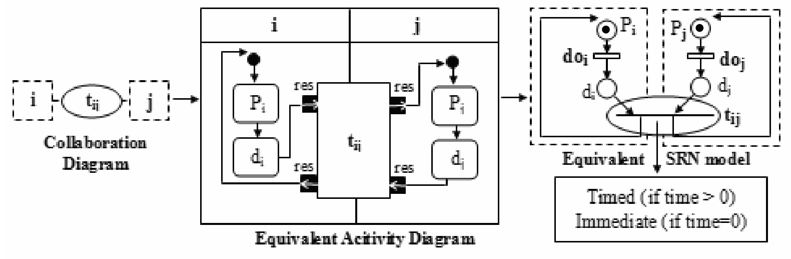
T = $\{do_i, do_j, t_{ij}\}$; **A** = $\{\{(P_i \times do_i) \cup (do_i \times d_i)\}, \{(d_i \times t_{ij}) \cup (t_{ij} \times P_i)\}, \{(P_j \times do_j) \cup (do_j \times d_j)\} \{(d_j \times t_{ij}) \cup (t_{ij} \times P_j)\}\}$

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

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

$\mathbf{m_0} = \{(P_i \rightarrow 1, d_i \rightarrow 0, P_j \rightarrow 1, d_j \rightarrow 0\}$

SRN model of the collaboration is graphically represented in the following way:



Here, places are represented by P_i, d_i, P_j, d_j, transitions are represented by do_i, do_j and t_{ij}, where do_i and do_j are timed transition and t_{ij} is a timed transition if the two collaboration roles deploy on the different physical node (communication time > 0) or immediate transition if the two collaboration roles deploy on the same physical node (communication time = 0).

Rule 3: When the collaboration role of a reusable building block deploys onto a physical node the equivalent SRN model is represented by 6-tuple in following way:

$\mathbf{\Phi_i} = \{P_i, d_i, P_\Omega\}$

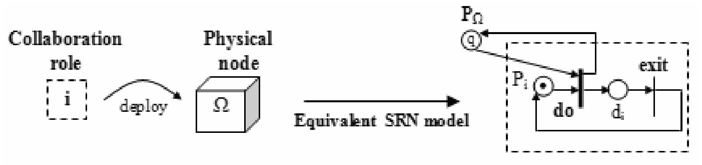
T= {do, exit}; **A**= $\{\{(P_i \times do) \cup (do \times d_i)\}, \{(P_\Omega \times do) \cup (do \times P_\Omega)\}, \{(d_i \times exit) \cup (exit \times P_i)\}\}$

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

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

$\mathbf{m_0} = \{(P_i \rightarrow 1\}, (d_i \rightarrow 0), (P_\Omega \rightarrow q)\}$

Here, places are represented by P_i, d_i and P_Ω. Transitions are represented by *do* and *exit,* where *do* is a timed transition and *exit* is an immediate transition. Initially place P_i contains one token, place d_i contains no token and place P_Ω contains **q** (where **q** > 0) tokens, which define the upper bound of the execution of the threads in parallel by the physical node Ω and the timed transition *do* will fire only, when there is a token available in both the place P_i and P_Ω. The place P_Ω will again get back it's token after firing of the timed transition *do* indicating that the node is ready to execute incoming threads. SRN model of the collaboration role is graphically represented by the following way:



6) Evaluate the model: We focus on measuring the throughput of the system from the developed SRN model. Before deriving formula for throughput estimation we consider several assumptions. Firstly, if more than one service component deploy on a network node the processing power of the network node will be utilized among the multiple threads to complete the parallel processing of that node. There must be an upper bound of the execution of parallel threads by a network node. Secondly, when communication between two processes of two interconnected nodes occur it follows the rendezvous synchronization. Moreover, all the communications among the interconnected nodes occur in parallel. Finally, the communications between interconnected nodes will be started following the completion of all the processing inside each physical node. By considering the all the assumption we define the throughput as function of total expected number of jobs, E(N) and cost of the network, C_{Net}. The value of E(N) is calculated by solving the SRN model using SHARPE [15]. The value of C_{Net} is evaluated by considering a subnet, which is performance limiting factor of the whole network i.e., which posses maximum cost with respect to its own execution cost, communication cost with other subnet and cost for running background processes. Assume cost of the network, C_{Net} is defined as follows (where c_subnet_i = cost of the i^th subnet, where i = 1,…, n; that comprises the whole network, f_{c_m} = execution cost of the m^{th} component of $subnet_i$, where m=1….n; which defines the total number of collaboration roles in the i^{th} subnet, j= 1…n; which defines the total number of collaborations in the i^{th} subnet and $I_j = 0$ in this case as k_j internal to a node):

$$c\_subnet_i = \max \{ f_{c_m} + (I_j f_{k_j} + f_{B_j}) \}$$

$$= \max \{ f_{c_m} + f_{B_j} \} \qquad (3)$$

Now we evaluate the cost between each pair of subnet ($sbunet_a$ and $subnet_b$; where $(a,b) \in N$, $a \neq b$) with respect to the subnet's own processing cost, cost for running background process and the cost associated with the communication with other subnet in the network. Cost of a subnet pair, $C\_subnetp_y$ is defined as (where j= 1…n; which defines the total number of collaborations between $subnet_a$ and $subnet_b$, y= 1…n; which defines the total number of subnet pair in the network and $I_j = 1$ as k_j external to nodes):

$$C\_subnetp_y = \max \{\max \{c\_subnet_a, c\_subnet_b\} + (I_j f_{k_j} + f_{B_j})\} \qquad (4)$$

$$C_{Net} = \max \{C\_subnetp_1,…,… C\_subnetp_n\}; \qquad (5)$$

$$\text{Throughput} = \frac{E(N)}{C_{Net}} \tag{6}$$

3 Application example

As a representative example, we consider the scenario originally from Efe dealing with heuristically clustering of modules and assignment of clusters to nodes [13]. This scenario is sufficiently complex to show the applicability of our proposed framework. The problem is defined in our approach as a service of collaboration of $E = 10$ components or collaboration roles (labeled $C_1 \ldots C_{10}$) to be deployed and $K = 14$ collaborations between them depicted in Figure 4. We consider four types of requirements in this specification. Besides the execution cost, communication costs and cost for running background process, we have a restriction on components C_2, C_7, C_9 regarding their location. They must be bound to nodes $\boldsymbol{n_2}$, $\boldsymbol{n_1}$, $\boldsymbol{n_3}$, respectively.

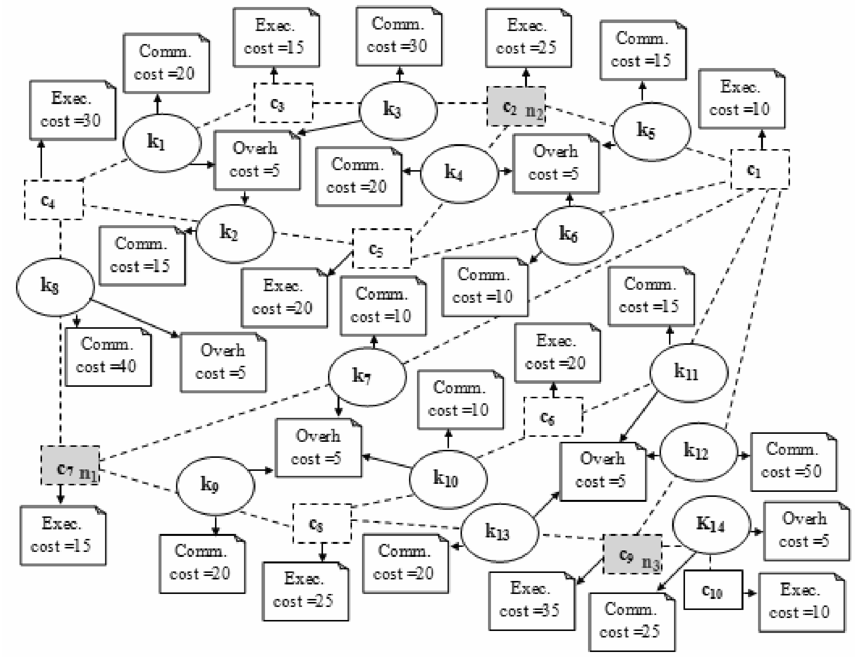


Figure 4. Collaborations and components in the example scenario

The internal behavior of the collaboration K_i of our example scenario is realized by the call behavior action through the same UML activity model already mentioned in Figure 2(b). The composition of the collaboration role C_i is realized through UML activity diagram shown in Figure 5. The initial node (●) indicates the starting of the activity. The activity is started at the same time from the entire participants C_1 to C_{10}. After being activated, each participant starts its processing of request, which is mentioned by call behavior action P_i (Processing of the i^th^ service component). Completions of the processing by the participants are mentioned by the call behavior action d_i *(*Processing done of the ith service component). After completion of the processing, the responses are

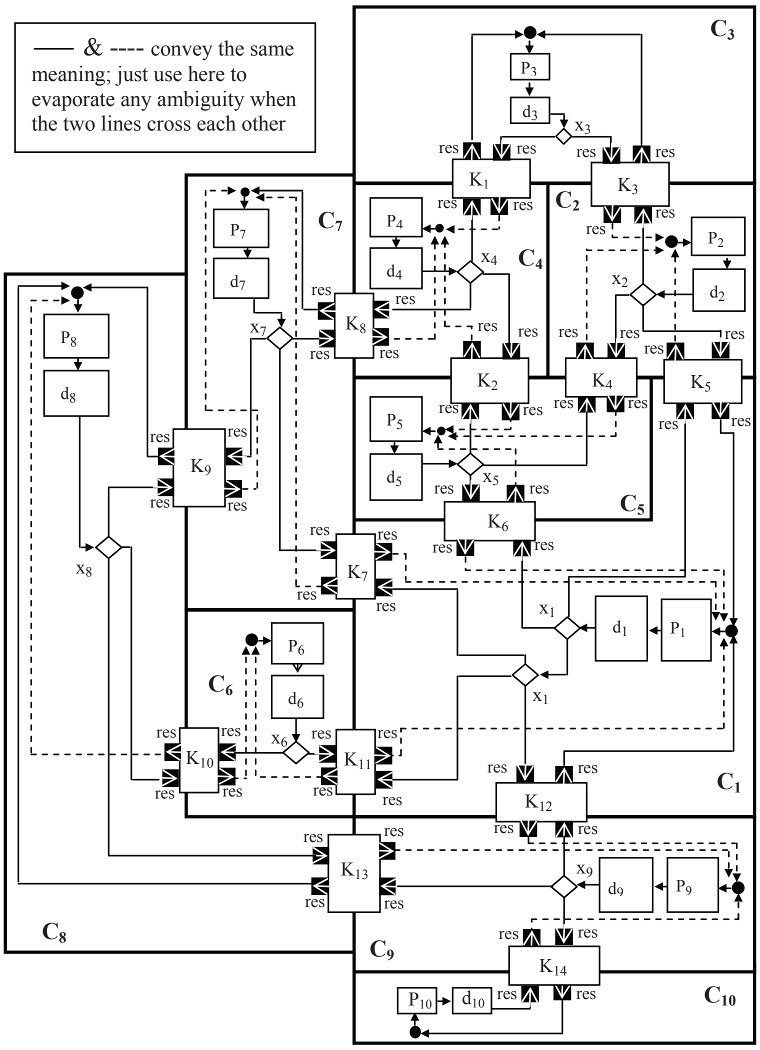


Figure 5. Detailed behavior of the event of the collaboration using activity for our example scenario

delivered to the corresponding participants indicated by the streaming pin *res*. When any participant is associated with more than one participant through collaborations the result of the processing of that participant is passed through a decision node and only one flow is activated at the certain time instance. For example, after completion of the processing of participant C_2 the response will be passed through the decision node x_2 and only one flow (flow towards C_1 or C_3 or C_5) will be activated.

In this example, the target environment consists only of N = 3 identical, interconnected nodes with a single provided property, namely processing power and with infinite communication capacities depicted in Figure 6 (a). The optimal deployment mapping can be observed in Table 1. The lowest possible deployment cost, according to (2) is 17 + (100 + 70) = 187.

Table. 1. Optimal deployment mapping in the example scenario

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}
∑ cost			17	100

To annotate the UML diagram in Figure 5 and 6(a) we use the stereotypes *SaStep, ComputingResource, Scheduler* and the tagged values *execTime, deadline* and *schedPolicy* [4]. *SaStep* is a kind of step that begins and ends, when decisions about the allocation of system resources are made. The duration of the execution time is mentioned by the 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. A *ComputingResource* represents either virtual or physical processing devices capable of storing and executing program code. Hence, its fundamental service is to compute. A *Scheduler* is defined as a kind of ResourceBroker that brings access to its brokered ProcessingResource or resources following a certain scheduling policy tagged by *schedPolicy.* Collaboration K_i is associated with two instances of *deadline* (Figure 6(b)) as collaborations in example scenario are associated with two kinds of cost: communication cost and cost for running background process.

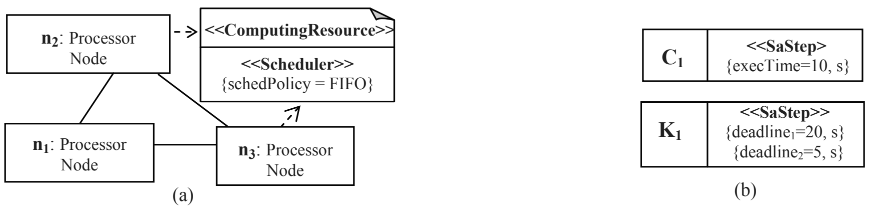


Figure 6. (a)The target network of hosts
(b) annotated UML model using MARTE profile

By considering the above deployment mapping and the transformation rule the analogous SRN model of our example scenario is depicted in Figure 7. The states of the SRN model are derived from the call behavior action of the corresponding collaboration role and collaboration among them. While generating the SRN model of the system if more than one service component deploy on a network node the processing power of the network node will be utilized among the multiple threads to complete the parallel processing of that node. This can be achieved through marking dependency firing rate defined as the following way in SRN model:

$$\lambda_i \ / \ \sum_{i=1}^{n} (\#\,(P_i)) \tag{7}$$

Where λ_i = processing rate of the ith service component deploys in a network node and i=1…n that means total n service components deploy on a network node. (# (P_i)) returns the number of tokens in the place P_i.

Initially, there will be a token in the place p_1 to p_{10}. For generating the SRN model firstly we will consider the collaboration roles deploy on the processor node $\boldsymbol{n_1}$, which are C_4, C_7 and C_8. Here, components C_7 are connected with C_4 and C_8. The communication cost between the components is zero but there is still some cost for execution of the background process. So according to rule 2, after the completion of the state transition 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 transition k_8 and k_9 to generate the SRN model. Collaboration roles C_2, C_3 and C_5 deploy on the processor node $\boldsymbol{n_2}$. Likewise, after the completion of the state transition from p_2 to d_2 (states of component C_2), from p_3 to d_3 (states of component C_3) and from p_5 to d_5 (states of component C_5) the states d_2, d_3 and d_2, d_5 are connected by the timed transition k_3 and k_4 to generate the SRN model according to rule 2. Collaboration roles C_6, C_1, C_9 and C_{10} deploy on the processor node $\boldsymbol{n_3}$. In the same way, after the completion of the state transition from p_1 to d_1 (states of component C_1), from p_6 to d_6 (states of component C_6), p_9 to d_9 (states of component C_9) and from p_{10} to d_{10} (states of component C_{10}) the states d_1, d_6; d_1, d_9 and d_9, d_{10} are connected by the timed transition k_{11}, k_{12} and K_{14} to generate the SRN model following rule 2. To generate the system level SRN model, we need to combine the entire three SRN model generated for three processor nodes by considering the interconnection among them. To compose the SRN models of processor node $\boldsymbol{n_1}$ and $\boldsymbol{n_2}$, states d_4 and d_3 connect by the timed transition k_1 and states d_4 and d_5 connect by the timed transition k_2 according to rule 2. Likewise, to compose the SRN models of processor node $\boldsymbol{n_2}$ and $\boldsymbol{n_3}$, states d_2 and d_1 connect by the timed transition k_5 and states d_5 and d_1 connect by the timed transition k_6 according to rule 2. To compose the SRN models of processor node $\boldsymbol{n_1}$ and $\boldsymbol{n_3}$, states d_7 and d_1 connect by the timed transition k_7, states d_8 and d_6 connect by the timed transition k_{10} and states d_8 and d_9 connect by the timed transition k_{13} according to rule 2. By the above way, the system level SRN model is derived. According to rule 3, to define the upper bound of the execution of parallel threads by a network node we introduce three places PP_1, PP_2 and PP_3 in the SRN model for the three network nodes and initially, these three places will contain $\boldsymbol{q}$ ($\boldsymbol{q}$ = 1, 2, 3,…….) tokens, where $\boldsymbol{q}$ will define the maximum number of the threads that will be handled by a network node at the same time. To ensure the upper bound of the parallel processing of a network node $\boldsymbol{n_1}$, we introduce arcs from place PP_1 to transition t_4, t_7 and t_8. That means components C_4, C_7 and C_8 can start their processing if there is token available in place PP_1 as the firing of transitions t_4, t_7 and t_8 not only depend on the availability of the token in the place p_4, p_7 and p_8 but also depend on the availability of the token in the place PP_1. Likewise, to ensure the upper bound of the parallel processing of a network node $\boldsymbol{n_2}$ and $\mathbf{n_3}$ we introduce arcs from place PP_2 to transition t_2, t_3 and t_5 and from place PP_3 to transition t_1, t_6, t_9, t_{10}.

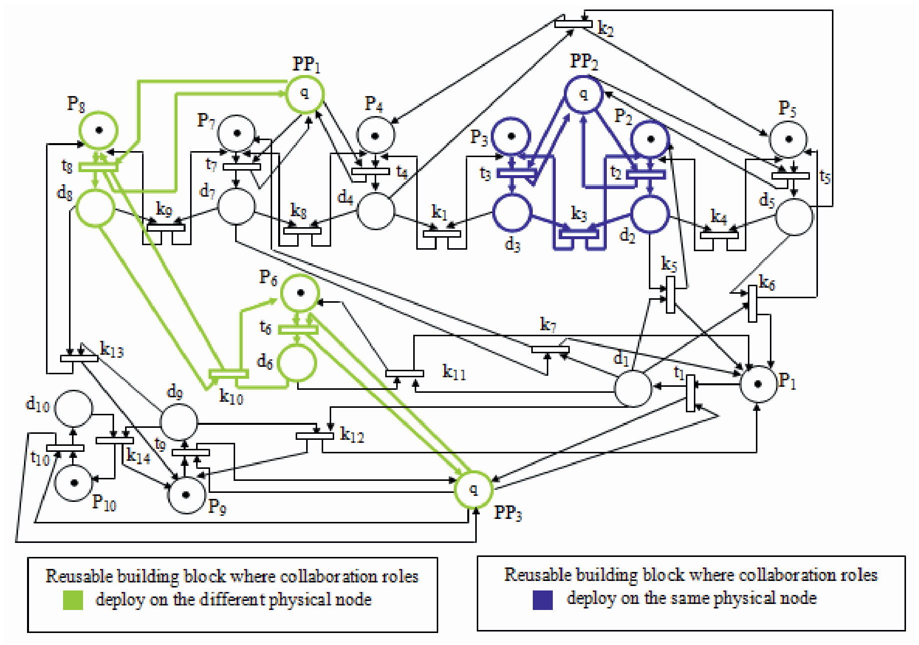

PP1
PP2
PP3
P1
P2
P3
P4
P5
P6
P7
P8
P9
P10
t1
t2
t3
t4
t5
t6
t7
t8
t9
t10
d1
d2
d3
d4
d5
d6
d7
d8
d9
d10
k1
k2
k3
k4
k5
k6
k7
k8
k9
k10
k11
k12
k13
k14
q
Reusable building block where collaboration roles deploy on the different physical node
Reusable building block where collaboration roles deploy on the same physical node

Figure 7. SRN model of our example scenario

The throughput calculation according to (6) for the different deployment mapping including the optimal deployment mapping is shown in Table 2. The throughput is $0.107s^{-1}$ while considers the optimal deployment mapping, where E(N) = 6.96 (calculated using SHARPE [15]) and C_Net = 65s. The optimal deployment mapping presented in Table 1 also ensures the optimality in case of throughput calculation. We

Table 2. Optimal deployment mapping in the example scenario

Node	Components	Possible 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	0.107
$\{n_1, n_2, n_3\}$	$\{\{c_4, c_6, c_7, c_8\}, \{c_2, c_3, c_5\}, \{c_1, c_9, c_{10}\}\}$	218	0.106
$\{n_1, n_2, n_3\}$	$\{\{ c_4, c_7\}, \{c_2, c_3, c_5, c_6,\}, \{c_1, c_8, c_9, c_{10}\}\}$	232	0.102
$\{n_1, n_2, n_3\}$	$\{\{c_5, c_7, c_8\}, \{c_2, c_3, c_4\}, \{ c_1, c_6, c_9, c_{10}\}\}$	227	0.086
$\{n_1, n_2, n_3\}$	$\{\{ c_3, c_7, c_8\}, \{c_2, c_4, c_5\}, \{c_1, c_6, c_9, c_{10}\}\}$	252	0.084
$\{n_1, n_2, n_3\}$	$\{\{ c_1, c_6, c_7, c_8\}, \{c_2, c_3, c_5\}, \{ c_4, c_9, c_{10}\}\}$	257	0.083
$\{n_1, n_2, n_3\}$	$\{\{c_1, c_6, c_7, c_8\}, \{c_2, c_3, c_4\}, \{c_5, c_9, c_{10}\}\}$	247	0.075
$\{n_1, n_2, n_3\}$	$\{\{c_4, c_7, c_8\}, \{ c_1, c_2, c_3, c_5\}, \{ c_6, c_9, c_{10}\}\}$	217	0.073
$\{n_1, n_2, n_3\}$	$\{\{c_3, c_6, c_7, c_8\}, \{c_1, c_2, c_4, c_5\}, \{c_9, c_{10}\}\}$	302	0.072
$\{n_1, n_2, n_3\}$	$\{\{c_6, c_7, c_8\}, \{ c_1, c_2, c_4, c_5\}, \{c_3, c_9, c_{10}\}\}$	288	0.071

present here the throughput calculation of some of the deployment mappings of the software artifacts but obviously the approach presented here confirms the efficiency in both deployment mapping and throughput calculation for all the cases.

4 Conclusion

We present a novel approach for model based performance evaluation of distributed system, which spans from capturing the system dynamics through UML diagram as reusable building block to efficient deployment of service components in a distributed manner by capturing the QoS requirements. System dynamics is captured through UML collaboration and activity oriented approach. Furthermore, quantitative analysis of the system is achieved by generating SRN performance model from the UML specification style. The transformation from UML diagram to corresponding SRN elements like states, different pseudostates and transitions is proposed. Performance related QoS information is taken into account and included in the SRN model with equivalent timing and probabilistic assumption for enabling the evaluation of performance prediction result of the system at the early stage of the system development process. In addition, the logic, as it is presented here, is applied to provide the optimal, initial mapping of components to hosts, i.e. the network is considered rather static. However, our eventual goal is to develop support for run-time redeployment of components, this way keeping the service within an allowed region of parameters defined by the requirements. As the results with our proposed framework show our logic will be a prominent candidate for a robust and adaptive service execution platform.

References

1. F. A. Kramer, R. Bræk, P. Herrmann, "Synthesizes components with sessions from collaboration-oriented service specifications", SDL 2007, V-4745, LNCS, 2007.
2. OMG 2009, "UML Superstructure", Version-2.2
3. M. Csorba, P. Heegaard, P. Herrmann, "Cost-Efficient Deployment of Collaborating Components", Proceedings of the DAIS, LNCS, pp. 253–268, Springer, 2008
4. OMG 2009, "UML Profile for MARTE: Modeling and Analysis of Real-Time Embedded Systems", V – 1.0
5. K. S. Trivedi, "Probability and Statistics with Reliability, Queuing and Computer Science application", Wiley- Interscience publication, ISBN 0-471-33341-7
6. J. P. Lopez, J. Merseguer, J. Campos, "From UML activity diagrams to SPN: application to software performance engineering", ACM SIGSOFT software engineering notes, NY, 2004
7. S. Distefano,M. Scarpa, A. Puliafito, "Software Performance Analysis in UML Models", FIRB-PERF, 2005
8. A. D'Ambrogio, "A Model Transformation Framework for the Automated Building of Performance Models from UML Models", ACM SIGSOFT software engineering notes, NY, 2005
9. R. H. Khan, P. E. Heegaard, "Translation from UML to SPN model: A performance modeling framework", Proceedings of the EUNICE, Springer, Norway, 2010

10. R H Khan, P Heegaard, “Translation from UML to SPN model: Performance modeling framework for managing behavior of multiple session and instance” Proceedings of the ICCDA, IEEE computer society, China, 2010
11. F. A. Kramer, “ARCTIS”, Department of Telematics, NTNU, http://arctis.item.ntnu.no
12. Rendezvous synchronization, http://book.opensourceproject.org.cn/embedded/cmprealtime/opensource/5107final/lib0091.html, retrieved June, 2010
13. Efe, K., “Heuristic models of task assignment scheduling in distributed systems”, Computer (June 1982)
14. R H Khan, P Heegaard, “ A Performance modeling framework incorporating cost efficient deployment of collaborating components”, Proceedings of the ICSTE, IEEE computer society, USA, 2010
15. K. S. Trivedi, R Sahner, “Symbolic Hierarchical Automated Reliability / Performance Evaluator (SHARPE)”, Duke University, Durham, NC, 2002

Paper 6

Derivation of Stochastic Reward Net (SRN) from UML specification considering cost efficient deployment management of collaborative service components

Razib Hayat Khan, Poul E. Heegaard

Published in the International Journal of New Computer Architectures and Their Applications (IJNCAA), Volume – 1, Issue – 3, 2011, p. 721-742, 2011

Derivation of Stochastic Reward net (SRN) from UML specification considering cost efficient deployment management of collaborative service components

Razib Hayat Khan, Poul E. Heegaard

Department of Telematics
Norwegian University of Science and Technology (NTNU)
7491, Trondheim, Norway
{rkhan, poul.heegaard }@item.ntnu.no

Abstract- Performance evaluation of a distributed system is always an intricate undertaking, where system behavior is distributed among several components those are physically distributed. Bearing this concept, we delineate a performance modeling framework for a distributed system that proposes a transformation process from high level UML notation to SRN model and solves the model for relevant performance metrics. To capture the system dynamics through our proposed framework we outline a specification style that focuses on UML collaboration and activity as reusable specification building blocks, while deployment diagrams identify the physical components of the system and the assignment of software artifacts to identified system components. Optimal deployment mapping of software artifacts on the available physical resources of the system is investigated by deriving the cost function. The way to deal with parallel thread processing of the network nodes by defining the upper bound is precisely mentioned to generate the SRN model. The proposed performance modeling framework provides transformation rules of UML elements into corresponding SRN representations and also the prediction result of a system such as throughput. The applicability of our proposed framework is demonstrated in the context of performance modeling of a distributed system.

1 Introduction

Distributed system poses one of the main streams of information and communication technology arena with immense complexity. Designing and implementation of such complex systems are always an intricate endeavor. Likewise, performance evaluation is also a great concern of such complex system to evaluate whether the system meets the performance related system requirements. Hence, modeling phase plays an important role in the whole design process of the system for qualitative and quantitative analysis. However, in a distributed system, system behavior is normally distributed among several objects. The overall behavior of the system is composed of the partial behavior of the distributed objects of the system. So it is obvious to capture the behavior of the distributed objects for appropriate analysis to evaluate the performance related factors of the overall system. We therefore, adopt UML collaboration and activity oriented approach as UML is the most widely used modeling language, which models both the system requirements and qualitative behavior through different notations [2]. Collaboration and activity diagram are utilized to demonstrate the overall system behavior by defining both the structure of the partial object behavior as well as the interaction between them as reusable specification building blocks and later on, this UML specification style is applied to generate the SRN model by our proposed performance modeling framework. UML collaboration and activity provides a tremendous modeling framework containing several interesting properties. Firstly, collaborations and activity model the concept of service provided by the system very nicely. They define structure of partial object behavior, the collaboration roles and enable a precise definition of the overall system behavior. They also delineate the way to compose the services by means of collaboration uses and role bindings [1].

The modeling framework considers system execution architecture to realize the deployment of the service components. Abstract view of the system architecture is captured by the UML deployment diagram, which defines the execution architecture of the system by identifying the system components and the assignment of software artifacts to those identified system components [2]. Considering the system architecture to generate the performance model resolves the bottleneck of system performance by finding a better allocation of service components to the physical nodes. This requires an efficient approach to deploy the service components on the available hosts of distributed environment to achieve preferably high performance and low cost levels. The most basic example in this regard is to choose better deployment architectures by considering only the latency of the service. The easiest way to satisfy the latency requirements is to identify and deploy the service components that require the highest volume of interaction onto the same resource or to choose resources that are connected by links with sufficiently high capacity [3].

It is indispensable to extend the UML model to incorporate the performance related quality of service (QoS) information to allow modeling and evaluating the properties of a system like throughput, utilization, and mean response time. So the UML models are annotated according to the *UML profile for MARTE: Modeling & Analysis of Real-Time Embedded Systems* to include quantitative system parameters [4]. Thus, it helps to maintain consistency between system design and implementation with respect to requirement specification.

Markov models, stochastic process algebras, stochastic petri net and stochastic reward net (SRN) are probably the best studied performance modeling techniques [5]. Among all of them, we will focus on the stochastic reward net (SRN) as the performance model generated by our proposed framework due to its increasingly popular formalism for describing and analyzing systems, its modeling generality, its ability to capture complex system behavior concisely, its ability to preserve the original architecture of the system, to allow marking dependency firing rates and reward rates defined at the net level, to facilitate any modification according to the feedback from the performance evaluation and the existence of analysis tools.

Several approaches have been followed to generate the performance model from system design specification. Lopez-Grao *et al.* proposed a conversion method from annotated UML activity diagram to stochastic petrinet model [6]. Distefano *et al.* proposed a possible solution to address software performance engineering that evolves through system specification using an augmented UML notation, creation of an intermediate performance context model, generation of an equivalent stochastic petri net model whose analytical solution provides the required performance measures [7]. D'Ambrogio proposed a framework for transforming source software models into target performance models by the use of meta-modeling techniques for defining the abstract syntax of models, the interrelationships between model elements and the model transformation rules [8]. However, most existing approaches do not highlight more on the issue that how to optimally conduct the system modeling and performance evaluation. The framework presented in this work is the first known approach that introduces a new specification style utilizing UML behavioral diagrams as reusable specification building block, which is later on, used for generating performance model to produce performance prediction result at early stage of the system development process. Building blocks describe the local behavior of several components and the interaction between them. This provides the advantage of reusability of building blocks, since solution that requires the cooperation of several components may be reused within one self-contained, encapsulated building block. In addition, the resulting deployment mapping provided by our framework has great impact with respect to QoS provided by the system. Our aim is to deal with vector of QoS properties rather than restricting it in one dimension. Our presented deployment logic is surely able to handle any properties of the service, as long as we can provide a cost function for the specific property. The defined cost function is flexible enough to keep pace with the changing size of search space of available host in the execution environment to ensure an efficient deployment of service components. Furthermore, we aim to be able to aid the deployment of several different services at the same time using the same framework. The novelty of our approach also reflected in showing the optimality of our solution with respect to both deployment logic and evaluation of performance metrics.

The objective of the paper is to provide an extensive performance modeling framework that provides a translation process to generate SRN performance model from system design specification captured by UML behavioral diagrams and later on, solves the model for relevant performance metrics to demonstrate performance prediction results at early stage of the system development life cycle. To incorporate the cost function to draw relation between service components and available physical resources permit us identifying an efficient deployment mapping in a fully distributed manner. The way to deal with parallel thread processing of the network node by defining the upper bound is precisely mentioned while generating the SRN model through the proposed framework. The work presented in this paper is the extension of our previous work described in [9] [10] [14], where we presented our framework with respect to the execution of single and multiple collaborative sessions and considered alternatives system architecture candidates to describe the system behavior and evaluate the performance factors.

The paper is organized as follows: Section 2 introduces our proposed performance modeling framework, Section 3 demonstrates the application example to show the applicability of our modeling framework, Section 4 delineates conclusion with future works.

2 Performance modeling framework

Our performance modeling framework utilizes the tool suite Arctis, which is integrated as plug-ins into the eclipse IDE [11]. The proposed framework is composed of 6 steps shown in Figure 1, where steps 1 and 2 are the parts of Arctis tool suite.

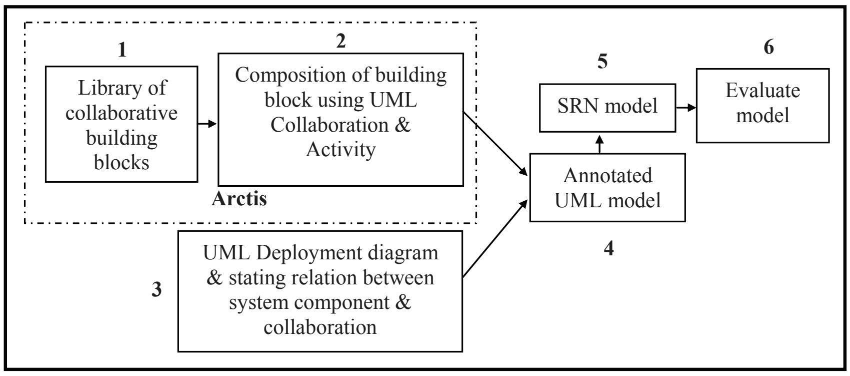

1
Library of collaborative building blocks
2
Composition of building block using UML Collaboration & Activity
Arctis
3
UML Deployment diagram & stating relation between system component & collaboration
4
Annotated UML model
5
SRN model
6
Evaluate model

Figure 1. Performance modeling framework

Arctis focuses on the abstract, reusable service specifications that are composed form UML 2.2 collaborations and activities. It uses collaborative building blocks as reusable specification units to create comprehensive services through composition. To support the construction of building blocks consisting of collaborations and activities, Arctis offers special actions and wizards. In addition, a number of inspections ensure the syntactic consistency of building blocks. A developer first consults a library to check if an already existing collaboration block or a collaboration of several blocks solves a certain task. Missing blocks can also be created from scratch and stored in the library for later reuse. The building blocks are expressed as UML models. The structural aspect, for example 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. They express the local behavior of each of the service components as well as their necessary interactions in a compact and self-contained way using explicit control flows [11]. Moreover, the building blocks are combined into more comprehensive service by composition. For this composition, Arctis uses UML 2.2 collaborations and activities as well. While collaborations provide a good overview of the structural aspect of the composition, i.e., which sub-services are reused and how their collaboration roles are bound, activities express the detailed coupling of their respective behavior [11].

The steps are illustrated below:

1) Construction of collaborative building block: The framework utilizes collaboration as main specification units. The specifications for collaborations are given as coherent, self-contained reusable building blocks. The structure of the building block is described by

UML 2.2 collaboration. The building block declares the participants (as collaboration roles) and connection between them. The internal behavior of building block is

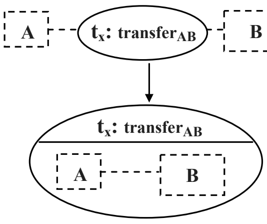


Figure 2. Structure of the building block using collaboration diagram

described by UML activity. It is declared as the classifier behavior of the collaboration and has one activity partition for each collaboration role in the structural description. For each collaboration use, the activity declares a corresponding call behavior action refereeing to the activities of the employed building blocks. For example, the general structure of the building block *t* is given in Figure 2, where it only declares the participants A and B as collaboration roles and the connection between them is defined as collaboration use t_x (x=1…n_{AB} (number of collaborations between collaboration roles A & B)). The internal behavior of the same building block is shown in Figure 3(b). The activity $transfer_{ij}$ (where ij = AB) describes the behavior of the corresponding collaboration. It has one activity partition for each collaboration role: A and B. Activities base their semantics on token flow [1]. The activity starts by placing a token, when there is a response (indicated by the streaming pin *res*) to transfer by either participant A or B. After completion of the processing by the collaboration role A and B the token is transferred from the participant A to participant B and from participant B to Participant A, which is represented by the call behavior action *forward.*

2) Composition of building block using UML collaboration and activity: To generate the performance model, the structural information about how the collaborations are composed is not sufficient. It is necessary to specify the detailed behavior of how the different events of collaborations are composed so that the desired overall system behavior can be obtained. For the composition, UML collaborations and activities are used complementary to each other; UML collaborations focus on the role binding and structural aspect, while UML activities complement this by covering also the behavioral aspect for composition. For this purpose, call behavior actions are used. Each sub-service is represented by call behavior action referring 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 activity parameter nodes to represent them on the frame of a call behavior action. Arbitrary logic between pins may be used to synchronize the building block events and transfer data between them. By connecting the individual input and output pins of the call behavior actions, the events occurring in different collaborations can be coupled with each other. Semantics of the different kinds of pins are given in more detailed in [1].

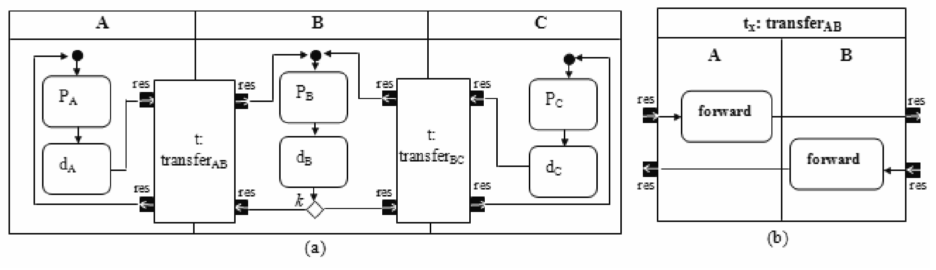


Figure 3. System activity to couple the collaboration

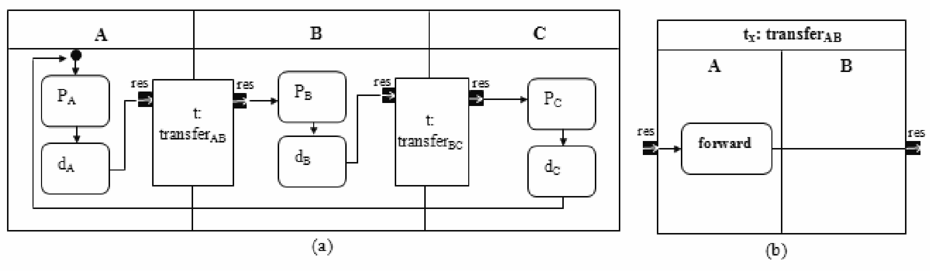


Figure 4. System activity to couple the collaboration when there is an order in which collaboration roles are selected for completing the processing

To delineate the overall system behavior we will consider two sorts of activity diagram, where activities base their semantics on the token flow. In first case, each collaboration role contains one token and the processing realized by the collaboration role is independent of each other and in second case one token will be passed through the each collaboration role to realize the processing done by the collaboration role, which symbolizes the dependency among the execution of collaborations roles' activity as there is an order in which collaboration roles are selected for completing the execution of their activity. For example, the detailed behavior and composition of the collaboration for the first case is given in Figure 3(a).The initial node (●) indicates the starting of the activity. The activity is started at the same time from each participant. After being activated, each participant starts its processing of the request, which is mentioned by call behavior action P_i (Processing$_i$, where i = A, B, and C). Completion of the processing by the participants are mentioned by the call behavior action d_i (Processing_done$_i$, i = A, B, and C). After completion of the processing, the responses are delivered to the corresponding participants indicated by the streaming pin *res*. When the execution of the task by the participant B completes the result is passed through a decision node *k* and only one flow is activated at the certain time instance. The response of the collaboration role A and C are forwarded to B and the response of collaboration role B is forwarded to either A or C, which is mentioned by collaboration *t: transfer$_{ij}$* (where ij = AB or BC). In the above way, the detailed behavior and composition of the collaboration as well as the internal behavior of the collaboration for the second case can be illustrated, which are portrayed in Figure 4(a) and 4(b).

3) Designing UML deployment diagram and stating relation between system components and collaborations: Our deployment logic is launched with the service model enriched with the requirements specifying the search criteria and with a resource profile of the hosting environment specifying the search space. In our view, however, the logic we develop is capable of catering for any other types of non-functional requirements too, as long as a suitable cost function can be provided for the specific QoS dimension at hand. In this paper, costs in the model are constant, independent of the utilization of underlying hardware [3]. Furthermore, we get benefit from using collaborations as design elements as they incorporate local behavior of all participants and all interactions between them. That is, a single cost value can describe communication between component instances, without having to care about the number of messages sent, individual message sizes, etc.

We model the system as collection of N interconnected nodes shown in Figure 5. Our objective is to find a deployment mapping for this execution environment for a set of service components C available for deployment that comprises service. Deployment mapping can be defined as M: $C \rightarrow N$ between a numbers of service components instances c, onto nodes n. A components $c_i \in C$ can be a client process or a service process, while a node, $n \in N$ is a physical resource. Generally, nodes can have different responsibilities, such as providing services (*S1*), relaying traffic (*R1*), accommodating clients (*C1*), or a mixture of these (*SC1*). Components can communicate via a set of collaborations. We consider four types of requirements in the deployment problem.

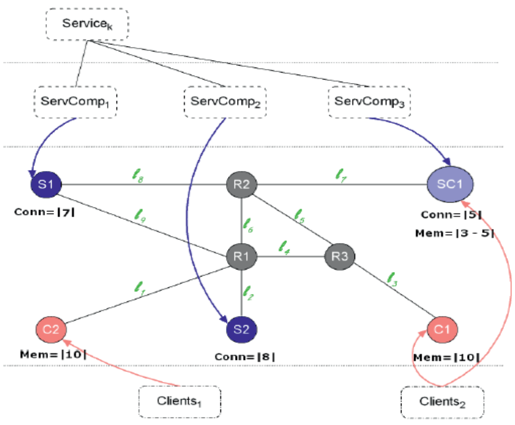


Figure 5. Components mapping example

Components have execution costs, collaborations have communication costs and costs for running of background process and some of the components can be restricted in the deployment mapping to specific nodes, which are called bound components. Furthermore, we consider identical nodes that are interconnected in a full-mesh and are capable of hosting components with unlimited processing demand. We observe the processing load that nodes impose while hosting the components and also the target balancing of load between the nodes available in the network.

By balancing the load the deviation from the global average per node execution cost will be minimized. Communication costs are considered if collaboration between two components happens remotely, i.e. it happens between two physical nodes [3]. In other words, if two components are placed onto the same physical node the communication cost between them will not be considered. The cost for executing the background process for conducting the communication between the collaboration roles is always considerable no matter whether the collaboration roles deploy on the same or different physical nodes. Using the above specified input, the deployment logic provides an optimal deployment architecture taking into account the QoS requirements for the components providing the specified services. We then define the objective of the deployment logic as obtaining an efficient (low-cost, if possible optimum) mapping of components onto the nodes that satisfies the requirements in a reasonable time. The deployment logic providing optimal deployment architecture is guided by the cost function F(M). The evaluation of cost function F(M) is mainly influenced by our way of service definition. Service is defined in our approach as a collaboration of total E components labeled as c_i (where i = 1…. E) to be deployed and total K collaboration between them labeled as k_j, (where j = 1 … K). The execution cost of each service component can be labeled as 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 only observe the total load ($\widehat{l}_n$, n = 1…N) of a given deployment mapping at each physical node. We will strive for an optimal solution of equally distributed load among the processing nodes and the lowest cost possible, while taking into account the execution cost 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 load can be calculated as $\sum_{i=1}^{|E|} f_{c_i}$. This way, the logic can be aware of the target load [6]: $T = \frac{\sum_{i=1}^{|E|} f_{c_i}}{|N|}$ (1)

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 [21]:

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

This means that $q_0(M,c)$ returns the node n that host component in the list mapping M. Let collaboration $k_j = \left(c_1, c_2\right)$. The communication cost of k_j is 0 if components c_1 and c_2 are collocated, i.e. $q_0(M,c_1) = q_0(M,c_2)$, and the cost is f_{k_j} if components are otherwise (i.e. the collaboration is remote). Using an indicator function *I(x)*, which is expressed as $I(q_0(M,c_1) \neq q_0(M,c_2)) = 1$, if the collaboration is remote and 0 otherwise. To determine which collaboration k_j is remote, the set of mapping M is used. Given the indicator function, the overall communication cost of service, $F_k(M)$, is the sum [21]

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

Given a mapping M = {m_n} (where m_n is the set of components at 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 (where $I_j = 1$, if k_j external or 0 if k_j internal to a node):

$$F(M) = \sum_{n=1}^{|N|} |\hat{l}_n - T| + F_K(M) + \sum_{j=1}^{|K|} f_{B_j} \qquad (2)$$

4) Annotating the UML model: Performance information is incorporated into the UML activity diagram and deployment diagram according to the *UML profile for MARTE: Modeling & Analysis of Real-Time Embedded Systems* [4] for evaluating system performance by performance model solver.

5) Deriving the SRN model: Since SRN model is based on a Petri net; the introduction of Petri net is described in brief [5]. A Petri net is represented by a bipartite directed graph with two types of node: places and transitions. Each place may contain zero or more tokens in a marking. Marking represents the state of the Petri net at a particular instant. A transition is enabled if all of its input places have at least as many tokens as required by the multiplicities of the input arcs. A transition may fire, when it is enabled and according to the multiplicities of the arcs, tokens in each input place are removed and new tokens are deposited in each output place. In a SPN, each transition has firing time that represents the time to fire the transition after it is enabled. Generalized stochastic Petri net (GSPN) extends SPN by introducing the immediate transition, which has zero firing time [5]. A marking in a GSPN is called vanishing if at least one immediate transition is enabled in the marking; otherwise the marking is called tangible. GSPN also introduces inhibitor arcs that disable the transition unless the number of tokens in input place is as many as the multiplicity of the inhibitor arc. An inhibitor arc is represented by a line terminated with a small hollow circle. SRN is based on the GSPN and extends them further by introducing prominent extensions such as guard functions, reward functions and marking dependent firing rates [5]. A guard function is assigned to a transition. It specifies the condition to enable or disable the transition and can use the entire state of the net rather than just the number of tokens in places. Reward function defines the reward rate for each tangible marking of a Petri Net based on which various quantitative measures can be done in the net level. Marking dependent firing rate allows using the number of token in a chosen place multiplied by the basic rate of the transition.

By considering the internal behavior of the reusable building blocks (step 1), composition of different events of the building blocks (step 2), deployment mapping between system components and collaborations (step 3) and annotated UML structure (step 4), probable states and transition rate for triggering the change between states will be found based on which the SRN performance model will be generated. To generate the SRN model of the system, firstly, we generate the SRN model of the individual system components and later on, compose them together to generate the system level SRN model. The rules are based on decomposition of UML collaboration, activity and deployment diagram into basic elements of the SRN model like states as places, timed transition and immediate transition. In addition, the rules are based on the rendezvous synchronization that means, when communication between two processes of two interconnected nodes occur it follows the rendezvous synchronization [12]. Rendezvous provides synchronization between two threads while they communicate. In rendezvous synchronization, a synchronization and communication point called an entry is constructed as a function call. One process defines its entry and makes it public. Any process with knowledge of this entry can call it as an ordinary function call. The process that defines the entry accepts the call, executes it and returns the results to the caller. The issuer of the entry call establishes a rendezvous with the process that defined the entry [12]. SRN model of the collaboration role of a reusable building block is mentioned by the 6-tuple $\{\Phi, T, A, K, N, m_0\}$ [5]:

Φ = Finite set of the places (drawn as circles)
T = Finite set of the transition (drawn as bars)
$A \subseteq \{\Phi \times T\} \cup \{T \times \Phi\}$ is a set of arcs connecting Φ and T,
K: $T \rightarrow$ {Timed (time>0, drawn as transparent bar), Immediate (time = 0, drawn as thin bar)} specifies the type of the each transition
N: $A \rightarrow \{1, 2, 3\ldots\}$ is the multiplicity associated with the arcs in A,
m: $\Phi \rightarrow \{0, 1, 2...\}$ is the marking that denotes the number of tokens for each place in Φ. The initial marking is denoted as m_0.

The rules are following:

Rule 1: The SRN model of the collaboration role of a reusable building block is represented by the 6-tuple in the following way:

$\Phi_i = \{P_i, d_i\}$
$T = \{do, exit\}$
$A = \{\{(P_i \times do) \cup (do \times d_i)\}, \{(d_i \times exit) \cup (exit \times P_i)\}\}$
$K = (do \rightarrow Timed, exit \rightarrow Immediate)$
$N = \{(P_i \times do) \rightarrow 1, (do \times d_i) \rightarrow 1, (d_i \times exit) \rightarrow 1, (exit \times P_i) \rightarrow 1\}$
$m_o = \{(P_i \rightarrow 1\}, (d_i \rightarrow 0)\}$

The Figure 6(a) highlights the SRN model of the collaboration role *A,* where *A* has its own token to start the execution of the SRN model and the Figure 6(b) highlights the SRN model of the collaboration role *A,* where the starting of the execution of the SRN model of *A* depends on the token received from other element.

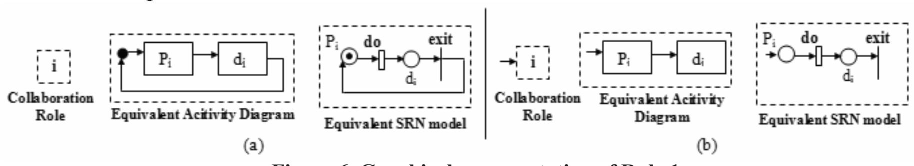


Figure 6. Graphical representation of Rule 1

Rule 2: The SRN model of a collaboration, where collaboration connects only two collaboration roles are represented by the 6-tuple in the following way (In this case, each collaboration role has its own token and the processing realized by the collaboration role is independent of each other):

$\Phi = \{\Phi_i, \Phi_j\} = \{P_i, d_i, P_j, d_j\}$
$T = \{do_i, do_j, t_{ij}\}$
$A = \{\{(P_i \times do_i) \cup (do_i \times d_i)\}, \{(d_i \times t_{ij}) \cup (t_{ij} \times P_j)\}, \{(P_j \times do_j) \cup (do_j \times d_j)\} \{(d_j \times t_{ij}) \cup (t_{ij} \times P_i)\}\}$
$K = (do_i \rightarrow \text{Timed}, do_j \rightarrow \text{Timed}, t_{ij} \rightarrow \text{Timed} \mid \text{Immediate})$
$N = \{(P_i \times do_i) \rightarrow 1, (do_i \times d_i) \rightarrow 1, (d_i \times t_{ij}) \rightarrow 1, (t_{ij} \times P_i) \rightarrow 1, \{\{(P_j \times do_j) \rightarrow 1, (do_j \times d_j) \rightarrow 1, (d_j \times t_{ij}) \rightarrow 1, (t_{ij} \times P_j) \rightarrow 1\}$
$m_o = \{(P_i \rightarrow 1, d_i \rightarrow 0, P_j \rightarrow 1, d_j \rightarrow 0\}$

Here, t_{ij} is a timed transition if the two collaboration roles deploy on the different physical nodes (communication time > 0) or immediate transition if the two collaboration roles deploy on the same physical node (communication time = 0). SRN model of the collaboration is graphically represented in Figure 7.

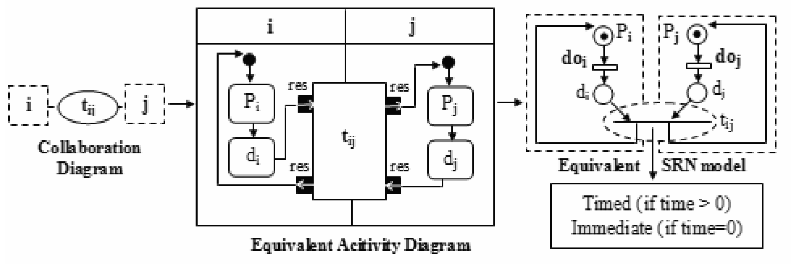


Figure 7. Graphical representation of Rule 2

Rule 3: The SRN model of a collaboration, where collaboration connects only two collaboration roles is represented by the 6-tuple in the following way (In this case, one token will be passed through the each collaboration role to realize the processing done by the collaboration role, which symbolizes the dependency among the execution of collaborations roles activity):

$\Phi = \{\Phi_i, \Phi_j\} = \{P_i, d_i, P_j, d_j\}$
$T = \{do_i, do_j, t_{ij}\}$
$A = \{\{(P_i \times do_i) \cup (do_i \times d_i)\}, \{(d_i \times t_{ij}) \cup ((t_{ij} \times P_i), (t_{ij} \times P_j))\}, \{(P_j \times do_j) \cup (do_j \times d_j)\} \{(d_j \times \text{exit}) \cup (\emptyset)\}\}$
$K = (do_i \rightarrow \text{Timed}, do_j \rightarrow \text{Timed}, t_{ij} \rightarrow \text{Timed} \mid \text{Immediate})$
$N = \{(P_i \times do_i) \rightarrow 1, (do_i \times d_i) \rightarrow 1, (d_i \times t_{ij}) \rightarrow 1, (t_{ij} \times P_i) \rightarrow 1, (t_{ij} \times P_j) \rightarrow 1, (P_j \times do_j) \rightarrow 1, (do_j \times d_j) \rightarrow 1, (d_j \times \text{exit}) \rightarrow 1\}$
$m_o = \{(P_i \rightarrow 1, d_i \rightarrow 0, P_j \rightarrow 1, d_j \rightarrow 0\}$

Here, t_{ij} is an immediate transition if the two collaboration roles deploy on the same physical node (communication time = 0) or timed transition if the two collaboration roles deploy on the different physical nodes (communication time > 0). SRN model of collaboration is represented graphically in Figure 8.

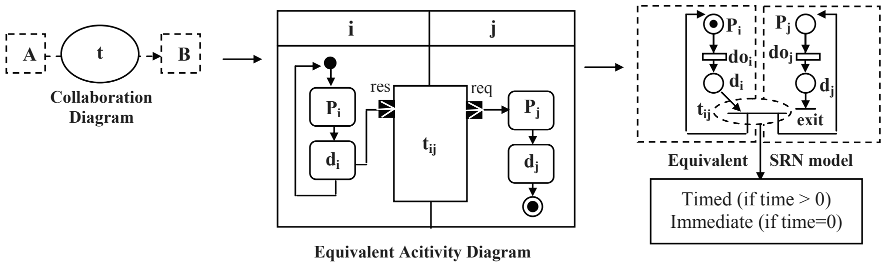


Figure 8. Graphical representation of Rule 3

Rule 4: When the collaboration role of a reusable building block deploys onto a physical node the equivalent SRN model is represented by 6-tuple in following way:

$\Phi_i = \{P_{i,} d_i, P_\Omega\}$
T= {do, exit}
A= $\{\{(P_i \times do) \cup (do \times d_i)\}, \{(P_\Omega \times do) \cup (do \times P_\Omega)\}, \{(d_i \times exit) \cup (exit \times P_i)\}\}$
K= (do → Timed, exit → Immediate)
N= $\{(P_i \times do) \rightarrow 1, (do \times d_i) \rightarrow 1, (P_\Omega \times do) \rightarrow 1, (do \times P_\Omega) \rightarrow 1 (d_i \times exit) \rightarrow 1, (exit \times P_i) \rightarrow 1\}$
$m_o = \{(P_i \rightarrow 1\}, (d_i \rightarrow 0), (P_\Omega \rightarrow q)\}$

Here, place P_Ω contains **q** (where **q** > 0) tokens, which define the upper bound of the execution of the threads in parallel by the physical node Ω and the timed transition *do* will fire only, when there is a token available in both the place P_i and P_Ω. The place P_Ω will again get back it's token after firing of the timed transition *do* indicating that the node is ready to execute other incoming threads. SRN model of the collaboration role deploys onto a physical node is graphically represented in the Figure 9.

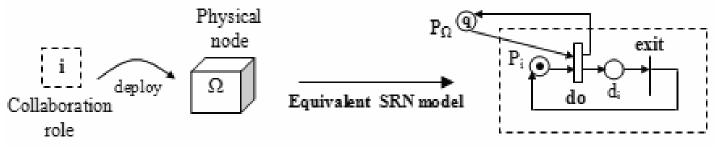


Figure 9. Graphical representation of Rule 4

Rule 5: For a composite structure, if a collaboration role *A* connects with *n* collaboration roles by *n* collaborations like a star graph (where n=2, 3, 4, …..), where each collaboration connects only two collaboration roles, then only one instance of collaboration role *A* exists during the it's basic state transition and the single instance of collaboration role *A* connects with all other collaboration roles by immediate or timed transitions based on their deployment on the same or different physical components to generate the SRN model. This rule can be demonstrated through 6-tuple in the above same way. The graphical representations of the SRN model for composite structures are shown in the Figure 10.

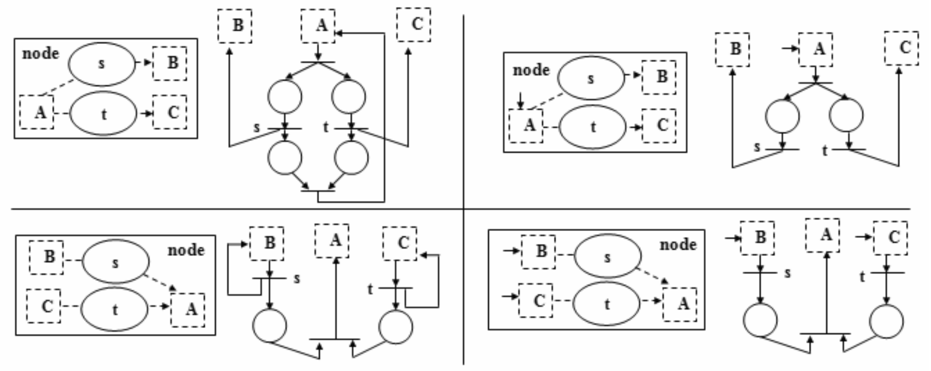


Figure 10. Graphical representation of Rule 5

6) Evaluate the model: We focus on measuring the throughput of the system from the generated SRN model. Before deriving formula for throughput estimation, we consider several assumptions. Firstly, if more than one service component deploy on a network node the processing power of the network node will be utilized among the multiple threads to complete the parallel processing of that node. There must be an upper bound of the execution of parallel threads by a network node. Secondly, when communication between two processes of two interconnected nodes occur it follows the rendezvous synchronization. Moreover, all the communications among the interconnected nodes occur in parallel. Finally, the communications between interconnected nodes will be started following the completion of all the processing inside each physical node. By considering the all the assumption we define the throughput as function of total expected number of jobs, E(N) and cost of the network, C_{Net}. The value of E(N) is calculated by solving the SRN model using SHARPE [15]. The value of C_{Net} is evaluated by considering a subnet, which is performance limiting factor of the whole network i.e., which posses maximum cost with respect to its own execution cost, communication cost with other subnet and cost for running background processes. Assume cost of the network, C_{Net} is defined as follows (where c_subnet_i = cost of the ith subnet, where i = 1,…, n; that comprises the whole network, f_{c_m} = execution cost of the mth component of $subnet_i$, where m=1….n; which defines the total number of collaboration roles in the ith subnet, j= 1…n; which defines the total number of collaborations in the ith subnet and $I_j = 0$ in this case as k_j internal to a node):

$$c\_subnet_i = \max \{ f_{c_m} + (I_j f_{k_j} + f_{B_j} \};$$
$$= \max \{ f_{c_m} + f_{B_j} \};$$

Now we evaluate the cost between each pair of subnet (sbunet$_a$ and subnet$_b$; where $(a,b) \in N$, a $\neq$ b) with respect to the subnet's own processing cost, cost for running background process and the cost associated with the communication with other subnet in the network. Cost of a subnet pair, C_subnetp$_y$ is defined as (where j= 1…n; which defines the total number of collaborations between subnet$_a$ and subnet$_b$, y= 1…n; which defines the total number of subnet pair in the network and I_j = 1 as k_j external to nodes):

$$\text{C\_subnetp}_y = \max \{\max \{\text{c\_subnet}_a, \text{c\_subnet}_b\} + (I_j\, f_{k_j} + f_{B_j})\}$$

$$C_{\text{Net}} = \max \{\text{C\_subnetp}_{1,\dots,}\ \text{C\_subnetp}_n\}$$

$$\text{Throughput} = \frac{\text{E(N)}}{C_{\text{Net}}} \tag{3}$$

Equation (3) for conducting the throughout calculation is considered, when each collaboration role has its own token and the processing realized by the collaboration role is independent of each other. The below equation (4) is considered for throughput calculation, when there is an order in which collaboration roles are selected for completing the execution.

$$\text{Throughput} = \frac{\text{E(N)}}{C_{\text{Net}'}} \tag{4}$$

Value of $C_{\text{Net}'}$ will be derived from equation (5).

$$C_{\text{Net}'} = \sum_{n=1}^{|N|} \hat{l}_n + \sum_{j=1}^{|K|} (I_j f_{k_j} + f_{B_j}) \tag{5}$$

where I_j = 1, if k_j external or 0, if k_j internal to a node.

3 Application example

As a representative example, we consider the scenario originally from Efe dealing with heuristically clustering of modules and assignment of clusters to nodes [13]. This scenario is sufficiently complex to show the applicability of our framework. The problem is defined in our approach as a service of collaboration of E = 10 components or collaboration roles (labeled $C_1 \dots C_{10}$) to be deployed and K = 14 collaborations between them depicted in Figure 11. We consider four types of requirements in this specification. Besides the execution cost, communication costs and cost for running background process, we have a restriction on components C_2, C_7, C_9 regarding their location. They must be bound to nodes $\boldsymbol{n_2}$, $\boldsymbol{n_1}$, $\boldsymbol{n_3}$, respectively. Moreover, collaboration and components in the example scenario are shown in Figure 12 as an order in which components are selected for completing the execution of their activity.

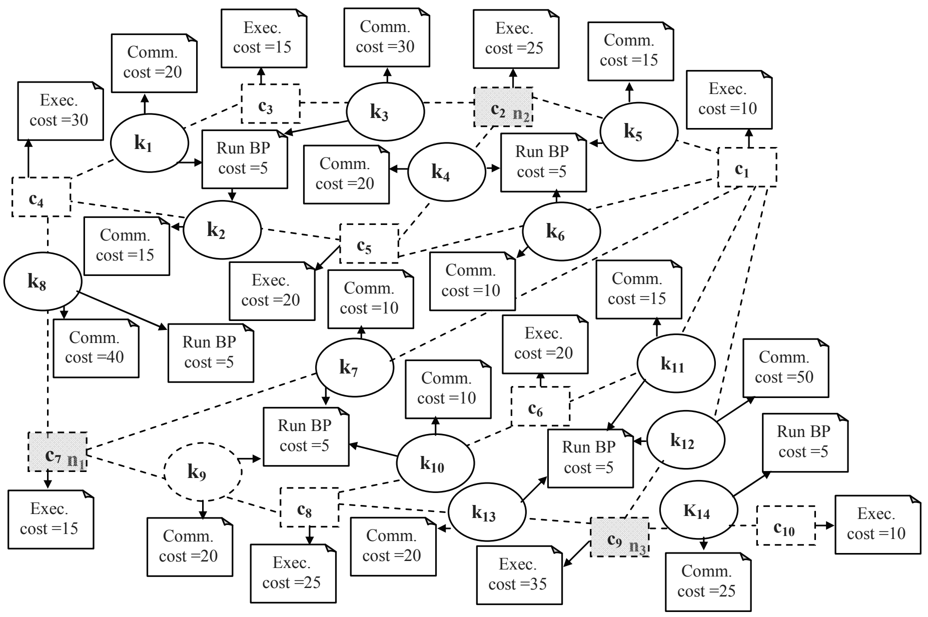


Figure 11. Collaborations and components in the example scenario

The internal behavior of the collaboration K_i of our example scenario is realized by the call behavior action through same UML activity diagram already mentioned in Figure 3(b). The composition of the collaboration role C is realized through UML activity diagram shown in Figure 13. The initial node (●) indicates the starting of the activity. The activity is started at the same time from the entire participants C_1 to C_{10}. After being activated, each participant starts its processing of request, which is mentioned by call behavior action P_i (Processing of the ith service component). Completions of the processing by the participants are mentioned by the call behavior action d_i (Processing done of the ith service component).

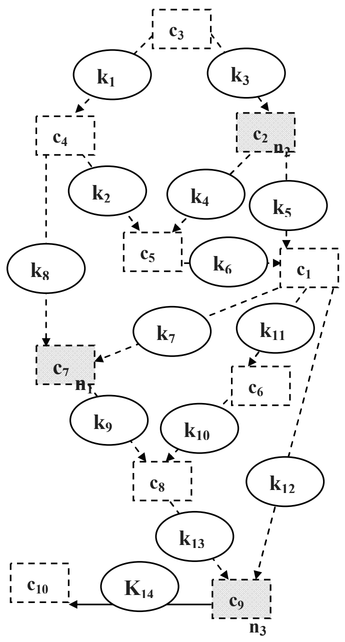


Figure 12. Collaborations and components in the example scenario when there is an order in which components are selected for completing the processing

After completion of the processing, the responses are delivered to the corresponding participants indicated by the streaming pin *res*. When any participant is associated with more than one participant through collaborations the result of the processing of that participant is passed through a decision node and only one flow is activated at the certain time instance. For example after completion of the processing of participant C_2 the response will be passed through the decision node X_2 and only one flow (flow towards C_1 or C_3 or C_5) will be activated. In the same way, the composition of the collaboration role C_i is also realized through UML activity diagram (Figure 14), where there is an order in which collaboration roles are selected for completing the execution of their activity. In this case, the internal behavior of the collaboration K_i of our example scenario is realized by the call behavior action through same UML activity diagram already mentioned in Figure 4(b).

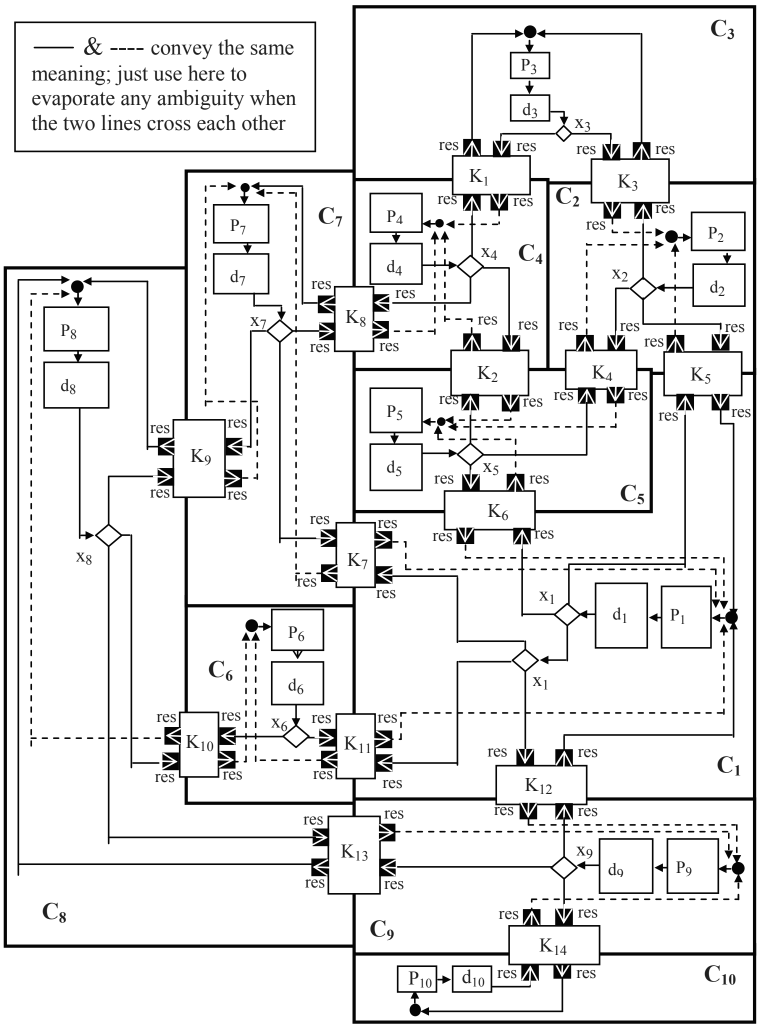


Figure 13. Detailed behavior of the event of the collaboration using activity for example scenario

In this example, the target environment consists only of $N = 3$ identical, interconnected physical nodes with a single provided property, namely processing power and with infinite communication capacities depicted in Figure 15(a). The optimal deployment mapping can be observed in Table 1. The lowest possible deployment cost, according to (2) is 17 + (100 + 70) = 187.

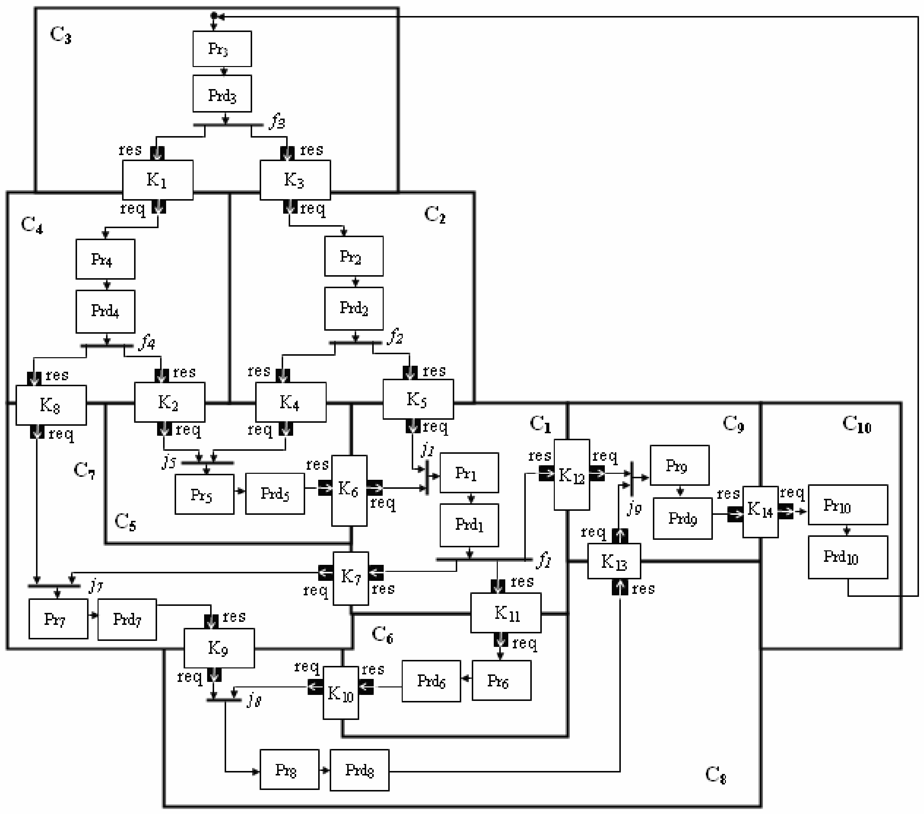


Figure 14. Detailed behavior of the event of the collaboration using activity for our example scenario where there is an order in which collaboration roles are selected for completing the processing

Table 1. Optimal deployment mapping in the example scenario

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

To annotate the UML diagram in Figure 13, 14 and 15(a) we use the stereotypes *SaStep ComputingResource, Scheduler* and the tagged values *execTime, deadline* and *schedPolicy* [4]. *SaStep* is a kind of step that begins and ends, when decisions about the allocation of system resources are made. The duration of the execution time is mentioned by the 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. A *ComputingResource* represents either virtual or physical processing devices capable of storing and executing program code. Hence, its fundamental service is to compute. A *Scheduler* is defined as a kind of ResourceBroker that brings access to its brokered ProcessingResource or resources following a certain scheduling policy tagged by *schedPolicy.* Collaboration K_i is associated with two instances of *deadline* (Figure 15(b)) as collaborations in example scenario are associated with two kinds of cost: communication cost and cost for running background process.

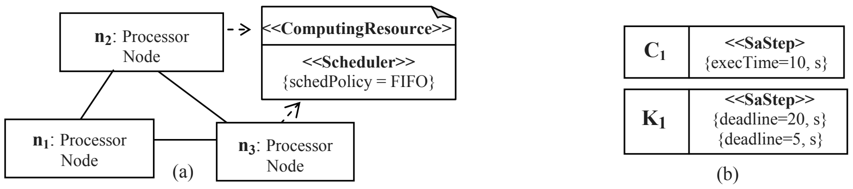


Figure 15. (a)The target network of hosts (b) annotated UML model using MARTE profile

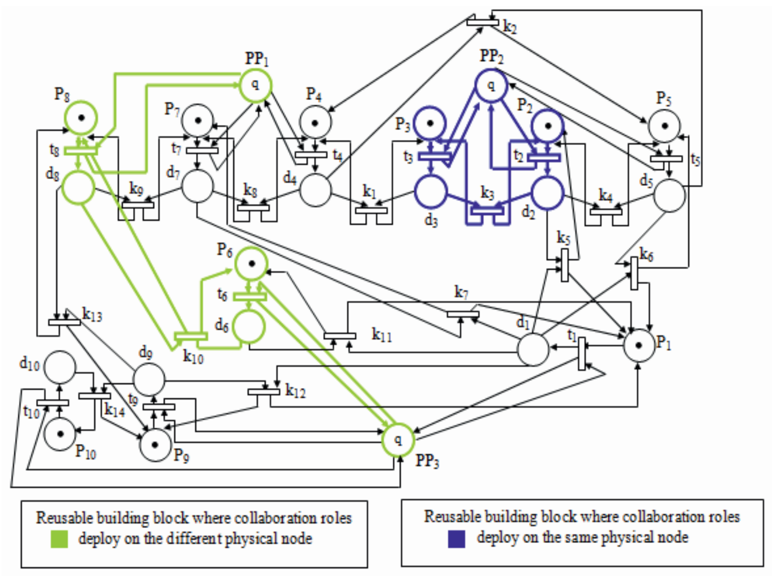
By considering the above deployment mapping and the transformation rules, the analogous SRN model of our example scenario is depicted in Figure 16, where each collaboration role has its own token and the processing realized by collaboration roles is independent of each other. The states of the SRN model are derived from the call behavior action of the corresponding collaboration role and collaboration among them. While generating the SRN model of the system if more than one service component deploy on a network node the processing power of the network node will be utilized among the multiple threads to complete the parallel processing of that node. This can be achieved through marking dependency firing rate defined as the following way in SRN model:

$$\lambda_i = \sum_{i=1}^{n} (\#(P_i)) \qquad (6)$$

Where λ_i = processing rate of the ith service component deploys in a network node and i=1…n defines the number of service components deploy on a network node. $(\#(P_i))$ returns the number of tokens in the place P_i.

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 t_i, which is derived from the annotated UML model. Initially, there will be a token from place p_1 to p_{10}. For generating the SRN model (Figure 16) firstly, we will consider the collaboration roles deploy on the processor node $\boldsymbol{n_1}$, which are C_4, C_7 and C_8. Here, components C_7 are connected with C_4 and C_8. The communication cost between the components is zero but there is still some cost for execution of the background process. So according to rule 2, after the completion of the state transition 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 transition k_8 and k_9

to generate the SRN model. Collaboration roles C_2, C_3 and C_5 deploy on the processor node $\boldsymbol{n_2}$. Likewise, after the completion of the state transition from p_2 to d_2 (states of



k2
PP1
PP2
P8
P7
P4
P3
P2
P5
t8
t7
t4
t3
t2
t5
d8
d7
d4
d3
d2
d5
k9
k8
k1
k3
k4
k5
k6
P6
t6
d6
k7
k13
k10
k11
d1
t1
P1
d10
d9
k12
t10
k14
t9
P10
P9
PP3
q
Reusable building block where collaboration roles deploy on the different physical node
Reusable building block where collaboration roles deploy on the same physical node

Figure 16. SRN model of our example scenario

component C_2), from p_3 to d_3 (states of component C_3) and from p_5 to d_5 (states of component C_5) the states d_2, d_3 and d_2, d_5 are connected by the timed transition k_3 and k_4 to generate the SRN model according to rule 2. Collaboration roles C_6, C_1, C_9 and C_{10} deploy on the processor node $\boldsymbol{n_3}$. In the same way, after the completion of the state transition from p_1 to d_1 (states of component C_1), from p_6 to d_6 (states of component C_6), p_9 to d_9 (states of component C_9) and from p_{10} to d_{10} (states of component C_{10}) the states d_1, d_6; d_1, d_9 and d_9, d_{10} are connected by the timed transition k_{11}, k_{12} and K_{14} to generate the SRN model following rule 2. To generate the system level SRN model we need to combine the entire three SRN model generated for three processor nodes by considering the interconnection among them. To compose the SRN models of processor node $\boldsymbol{n_1}$ and $\boldsymbol{n_2}$, states d_4 and d_3 connect by the timed transition k_1 and states d_4 and d_5 connect by the timed transition k_2 according to rule 2. Likewise, to compose the SRN models of processor node $\boldsymbol{n_2}$ and $\boldsymbol{n_3}$, states d_2 and d_1 connect by the timed transition k_5 and states d_5 and d_1 connect by the timed transition k_6 according to rule 2. To compose the SRN models of processor node $\boldsymbol{n_1}$ and $\boldsymbol{n_3}$, states d_7 and d_1 connect by the timed transition k_7, states d_8 and d_6 connect by the timed transition k_{10} and states d_8 and d_9 connect by the timed transition k_{13} according to rule 2. By the above way, the system level SRN model is derived. According to rule 4, to define the upper bound of the execution of parallel threads by a network node we introduce three places PP_1, PP_2 and PP_3 in the SRN model for the three network nodes and initially these three places will contain $\boldsymbol{q}$ ($\boldsymbol{q} > \boldsymbol{0}$) tokens, where $\boldsymbol{q}$ will define the maximum number of the threads that will be handled by a network node at the same time. To ensure the upper bound of the parallel processing of a network node $\boldsymbol{n_1}$, we introduce arcs from place PP_1 to transition t_4, t_7 and t_8. That means components C_4, C_7 and C_8 can start their processing if there is token available in place PP_1 as the firing of transitions t_4, t_7 and t_8 not only depend on the availability of the token in the 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 of the parallel processing of a network node $\boldsymbol{n_2}$ and $\boldsymbol{n_3}$, 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, t_{10}.

In the same way, by considering the above same deployment mapping and the transformation rules 1, 3 and 5, the analogous SRN model of our example scenario is depicted in Figure 17, where there is an order in which collaboration roles are selected for completing the execution of their activity, which symbolizes the dependency among the execution of collaborations roles' activity.

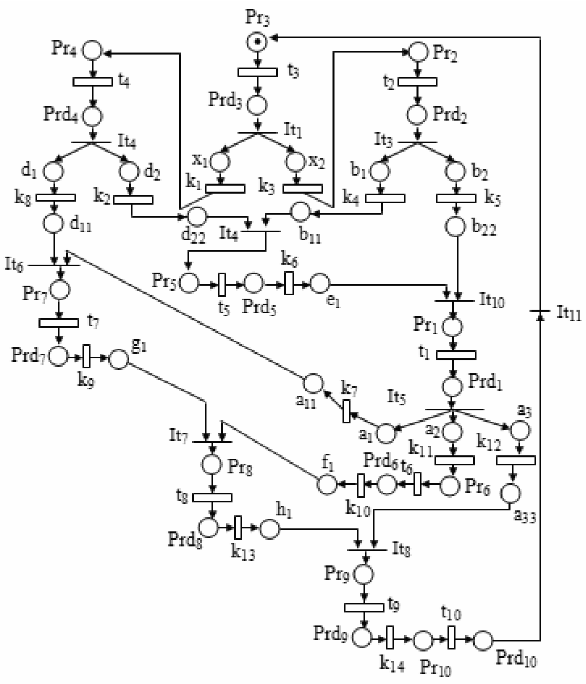


Figure 17. SRN model of our example scenario where there is an order in which components are selected for completing the processing

The throughput calculation according to (3) for the different deployment mapping including the optimal deployment mapping is shown in Table 2. The throughput is $0.107s^{-1}$ while considers the optimal deployment mapping, where E(N) = 6.96 (calculated using SHARPE [15]). The throughput calculation according to (4) for the different deployment mapping including the optimal deployment mapping is shown in Table. 3. The throughput is 2.33×10^{-4} s^{-1} while considers the optimal deployment mapping, where E(N) = 0.0435 (calculated using SHARPE [15]).

Table 2. Optimal deployment mapping in the example scenario

Node	Components	Possible cost (s)	Throughput (s^{-1})
$\{n_1, n_2, n_3\}$	$\{\{c_4, c_7, c_8\}, \{c_2, c_3, c_5\}, \{c_1, c_6, c_9, c_{10}\}\}$	187	0.107
$\{n_1, n_2, n_3\}$	$\{\{c_4, c_6, c_7, c_8\}, \{c_2, c_3, c_5\}, \{c_1, c_9, c_{10}\}\}$	218	0.106
$\{n_1, n_2, n_3\}$	$\{\{c_4, c_7\}, \{c_2, c_3, c_5, c_6,\}, \{c_1, c_8, c_9, c_{10}\}\}$	232	0.102
$\{n_1, n_2, n_3\}$	$\{\{c_5, c_7, c_8\}, \{c_2, c_3, c_4\}, \{c_1, c_6, c_9, c_{10}\}\}$	227	0.086
$\{n_1, n_2, n_3\}$	$\{\{c_3, c_7, c_8\}, \{c_2, c_4, c_5\}, \{c_1, c_6, c_9, c_{10}\}\}$	252	0.084
$\{n_1, n_2, n_3\}$	$\{\{c_1, c_6, c_7, c_8\}, \{c_2, c_3, c_5\}, \{c_4, c_9, c_{10}\}\}$	257	0.083
$\{n_1, n_2, n_3\}$	$\{\{c_1, c_6, c_7, c_8\}, \{c_2, c_3, c_4\}, \{c_5, c_9, c_{10}\}\}$	247	0.075
$\{n_1, n_2, n_3\}$	$\{\{c_4, c_7, c_8\}, \{c_1, c_2, c_3, c_5\}, \{c_6, c_9, c_{10}\}\}$	217	0.073
$\{n_1, n_2, n_3\}$	$\{\{c_3, c_6, c_7, c_8\}, \{c_1, c_2, c_4, c_5\}, \{c_9, c_{10}\}\}$	302	0.072
$\{n_1, n_2, n_3\}$	$\{\{c_6, c_7, c_8\}, \{c_1, c_2, c_4, c_5\}, \{c_3, c_9, c_{10}\}\}$	288	0.071

The optimal deployment mapping presented in Table 1 also ensures the optimality in case of throughput calculation for both the SRN performance model shown in Figure 16 and 17. We present here the throughput calculation of some of the deployment mappings of the software artifacts but obviously the approach presented here confirms the efficiency in both deployment mapping and throughput calculation for all the possible cases.

Table. 3. Optimal deployment mapping in the example scenario when there is an order in which components are selected for completing their activity

Node	Components	Possible cost (s)	Throughput (s^{-1})
$\{n_1, n_2, n_3\}$	$\{\{c_4, c_7, c_8\}, \{c_2, c_3, c_5\}, \{c_1, c_6, c_9, c_{10}\}\}$	187	2.33×10^{-4}
$\{n_1, n_2, n_3\}$	$\{\{c_4, c_7, c_8\}, \{c_1, c_2, c_3, c_5\}, \{c_6, c_9, c_{10}\}\}$	217	2.00×10^{-4}
$\{n_1, n_2, n_3\}$	$\{\{c_4, c_6, c_7, c_8\}, \{c_2, c_3, c_5\}, \{c_1, c_9, c_{10}\}\}$	218	1.99×10^{-4}
$\{n_1, n_2, n_3\}$	$\{\{c_5, c_7, c_8\}, \{c_2, c_3, c_4\}, \{c_1, c_6, c_9, c_{10}\}\}$	227	1.92×10^{-4}
$\{n_1, n_2, n_3\}$	$\{\{c_4, c_7\}, \{c_2, c_3, c_5, c_6,\}, \{c_1, c_8, c_9, c_{10}\}\}$	232	1.87×10^{-4}
$\{n_1, n_2, n_3\}$	$\{\{c_4, c_5, c_7, c_8\}, \{c_2, c_3\}, \{c_1, c_6, c_9, c_{10}\}\}$	232	1.87×10^{-4}
$\{n_1, n_2, n_3\}$	$\{\{c_1, c_6, c_7, c_8\}, \{c_2, c_3, c_4\}, \{c_5, c_9, c_{10}\}\}$	247	1.76×10^{-4}
$\{n_1, n_2, n_3\}$	$\{\{c_1, c_6, c_7, c_8\}, \{c_2, c_3, c_5\}, \{c_4, c_9, c_{10}\}\}$	257	1.69×10^{-4}
$\{n_1, n_2, n_3\}$	$\{\{c_6, c_7, c_8\}, \{c_1, c_2, c_4, c_5\}, \{c_3, c_9, c_{10}\}\}$	288	1.51×10^{-4}
$\{n_1, n_2, n_3\}$	$\{\{c_3,c_6, c_7, c_8\}, \{c_1, c_2, c_4, c_5\}, \{c_9, c_{10}\}\}$	302	1.44×10^{-4}

4 Conclusion

We present a novel approach for model based performance evaluation of distributed systems, which spans from capturing the system dynamics through UML diagram as reusable building block to efficient deployment of service components in a distributed manner by capturing the QoS requirements. System dynamics is captured through UML collaboration and activity oriented approach. The behavior of the collaboration and the composition of collaboration to highlight the overall system behavior are demonstrated by utilizing UML activity. Furthermore, quantitative analysis of the system is achieved by generating SRN performance model from the UML specification style. The transformation from UML diagrams to corresponding SRN elements like states, different pseudostates and transitions is proposed. Performance related QoS information is taken into account and included in the SRN model with equivalent timing and probabilistic assumptions for enabling the evaluation of performance prediction result of the system at the early stage of the system development process. In addition, the logic, as it is presented here, is applied to provide the optimal, initial mapping of components to hosts, i.e. the network is considered rather static. However, our eventual goal is to develop support for run-time redeployment of components, this way keeping the service within an allowed region of parameters defined by the requirements. As the results with our proposed framework show our logic will be a prominent candidate for a robust and adaptive service execution platform. However, the size of the underlying reachability set to generate SRN model is major limitation for large and complex systems. Further work includes automating the whole translation process, the way to solve the performance model and to tackle state explosion problems of reachability marking.

References

1. F. A. Kramer, R. Bræk, P. Herrmann, "Synthesizes components with sessions from collaboration-oriented service specifications", SDL 2007, V-4745, LNCS, 2007.
2. OMG 2009, "UML Superstructure", Version-2.2
3. M. Csorba, P. Heegaard, P. Herrmann, "Cost-Efficient Deployment of Collaborating Components", DAIS 2008, LNCS, pp. 253–268.
4. OMG 2009, "UML Profile for MARTE: Modeling & Analysis of Real-Time Embedded Systems", V – 1.0
5. K. S. Trivedi, "Probability and Statistics with Reliability, Queuing and Computer Science application", Wiley- Interscience publication, ISBN 0-471-33341-7
6. J. P. Lopez, J. Merseguer, J. Campos, "From UML activity diagrams to SPN: application to software performance engineering", ACM SIGSOFT software engineering notes, NY, 2004
7. S. Distefano,M. Scarpa, A. Puliafito, "Software Performance Analysis in UML Models", FIRB-PERF, 2005
8. A. D'Ambrogio, "A Model Transformation Framework for the Automated Building of Performance Models from UML Models", WOSP, 2005
9. R. H. Khan, P. E. Heegaard, "Translation from UML to SPN model: A performance modeling framework", EUNICE, 2010

10. R H Khan, P Heegaard, “Translation from UML to SPN model: Performance modeling framework for managing behavior of multiple session & instance” ICCDA 2010
11. F. A. Kramer, “ARCTIS”, Department of Telematics, NTNU, http://arctis.item.ntnu.no
12. Rendezvous synchronization, http://book.opensourceproject.org.cn/embedded/cmprealtime/opensource/5107final/lib0091.html, retrieved June, 2010
13. Efe, K., “Heuristic models of task assignment scheduling in distributed systems”, Computer (June 1982)
14. R H Khan, P Heegaard, “ A Performance modeling framework incorporating cost efficient deployment of collaborating components” ICSTE, 2010
15. K. S. Trivedi, R Sahner, “Symbolic Hierarchical Automated Reliability / Performance Evaluator (SHARPE)”, Duke University, Durham, NC

Paper 7

From UML to SRN: A performability modeling framework considering service components deployment

Razib Hayat Khan, Fumio Machida, Poul E. Heegaard, Kishor S. Trivedi

Presented in International Conference on Networking and Services (ICNS), St. Maarteen, Netherlands Antiles, 2012

Published in the Proceedings of International Conference on Networking and Services (ICNS), p. 118-127, IARIA, 2012

From UML to SRN: A performability modeling framework considering service components deployment

Razib Hayat Khan[1], Fumio Machida[2], Poul E. Heegaard[1], Kishor S. Trivedi[3]

[1]Department of Telematics
Norwegian University of Science and Technology (NTNU)
7491, Trondheim, Norway
{rkhan, poul.heegaard }@item.ntnu.no

[2]Service Platform Research
NEC, Japan
h-machida@ab.jp.nec.com

[3]Department of ECE
Duke University, NC, USA
kst@ee.duke.edu

Abstract- 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 system components due to failure and recovery. This introduces the concept of performability that considers the behavioral changes of the system components due to failures and also reveals how this behavioral change affect the system performance. But, to design a composite model for a distributed system, perfect modeling of the overall system behavior is crucial and sometimes very cumbersome. Additionally, evaluation of the required measures by solving the composite model are also intricate and error prone. Bearing this concept, we delineate a performability modeling framework for a distributed system that proposes an automated transformation process from high level UML notation to SRN model and solves the model to generate various numerical results. In order to capture system dynamics through our framework, we outline a specification style that focuses on UML collaboration and activity as reusable specification building blocks, while deployment diagram identifies the physical components of the system and the assignment of software artifacts to the identified system components. Optimal deployment mapping of software artifacts on the available physical resources of the system is investigated by deriving the cost functions. State machine diagram is utilized to capture state changes of system components such as failure and recovery. Later on, model composition is achieved by assigning guard functions.

1 Introduction

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

However, in a distributed system, system behavior is normally distributed among several objects. The overall behavior of the system is composed of the partial behavior of the distributed objects of the system. So it is obvious to model the behavior of the distributed objects perfectly for appropriate demonstration of the system dynamics. Hence, we adopt Unified Modeling Language (UML) collaboration, state machine and activity oriented approach as UML is the most widely used modeling language, which models both the system requirements and qualitative behavior through different notations [4]. Collaboration and activity diagram are utilized in the performance modeling layer to demonstrate the overall system behavior by defining both the structure of the partial object behavior as well as the interaction between them. State machine (STM) diagram is employed in the dependability modeling layer to capture system components behavior with respect to failure and repair events. Later on, the UML specification styles are applied to generate the Stochastic Reward Net (SRN) model automatically by our modeling framework. SRN models generated in both performance and dependability modeling layer are synchronized by the model synchronization role using guard functions (a special property of the SRN model [5]) to properly model the system performance behavior with respect to any state change in the system due to components failure [1]. The framework considers system architecture to realize the deployment of the service components. Abstract view of the system architecture is captured by the UML deployment diagram, which defines the execution architecture of the system by identifying the system components and the assignment of software artifacts to those identified system components [4]. Considering the system architecture to design the proposed framework resolves the bottleneck of system performance by finding a better allocation of service components to the physical nodes. This needs for an efficient approach to deploy the service components on the available hosts of distributed environment to achieve preferably high performance and low cost levels. Moreover, UML models are annotated according to the *UML profile for MARTE* [7] and *UML profile for Modeling Quality of Service and Fault Tolerance Characteristics and Mechanisms* to include quantitative system parameters [12].

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

Several approaches have been followed to conduct the performability analysis model from system design specification [8] [9] [10] [11]. However, most existing approaches do not highlight more on the issues that how to optimally conduct the system modeling to capture system dynamics and to conduct performability evaluation. The framework presented here is the first known approach that introduces a new specification style utilizing UML behavioral diagrams as reusable specification building block to characterize system dynamics. Building blocks describe the local behavior of several components and the interaction between them. This provides the advantage of reusability of building blocks, since solution that requires the cooperation of several components may be reused within one self-contained, encapsulated building block. This reusability provides the opportunity to design new system's behavior rapidly utilizing the existing building blocks according to the specification rather than starting the design process from the scratch. In addition, the resulting deployment mapping provided by our framework has greater impact with respect to QoS provided by the system. Our aim here is to deal with vector of QoS properties rather than restricting in one dimension. Our presented deployment logic is surely able to handle any properties of the service, as long as we can provide a cost function for the specific property. The cost function defined here is flexible enough to keep pace with the changing size of search space of available hosts in the execution environment to ensure an efficient deployment of service components. Furthermore, we aim to be able to aid the deployment of several different services at the same time using the same framework. Moreover, the introduction of model synchronization activity relinquishes the complexity and unwieldy affects in modeling and evaluation task of large and multifaceted systems. Model synchronization hides the intricacy behind demonstration of composite model behavior by designing guard functions [5]. Guard functions take charge of the proper functioning of the composite model by considering any changes in the dependability model.

The paper is organized as follows: Section 2 introduces our proposed modeling framework, Section 3 depicts UML based model description, Section 4 explains service component deployment issue, Section 5 clarifies model annotation, Section 6 delineates model translation rules, Section 7 introduces the model synchronization mechanism, Section 8 describes the fault tree model, Section 9 demonstrates the application example to show the applicability of our modeling framework and Section 10 delineates the conclusion with future directions.

2 Overview of the performability framework

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

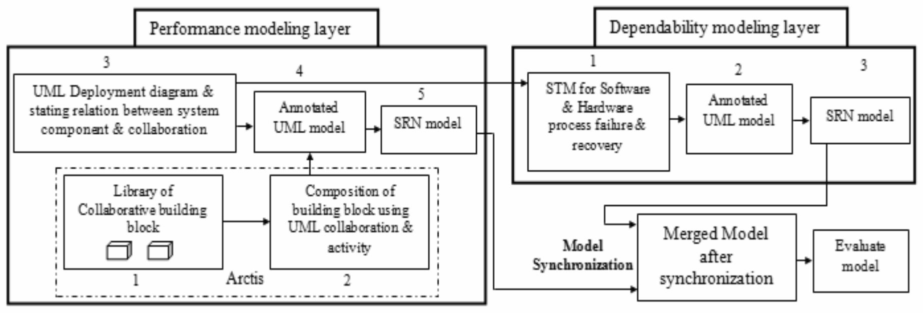

Performance modeling layer
3
UML Deployment diagram & stating relation between system component & collaboration
4
Annotated UML model
5
SRN model
Library of Collaborative building block
1
Composition of building block using UML collaboration & activity
2
Arctis
Dependability modeling layer
1
STM for Software & Hardware process failure & recovery
2
Annotated UML model
3
SRN model
Model Synchronization
Merged Model after synchronization
Evaluate model

Figure 1. Performability modeling framework

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

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

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

3 UML based system description

Construction of collaborative building blocks: The framework utilizes collaboration as main entity. Collaboration is an illustration of the relationship and interaction among software objects in the UML. Objects are shown as rectangles with naming label inside. The relationships between the objects are shown as line connecting the rectangles [4]. The specifications for collaborations here are given as coherent, self-contained reusable building blocks. The structure of the building block is described by UML 2.2 collaboration. The building block declares the participants (as collaboration roles) and connection between them. The internal behavior of building block is described by UML activity. It is declared as the classifier behavior of the collaboration and has one activity partition for each collaboration role in the structural description. For each collaboration, the activity declares a corresponding call behavior action refereeing to the activities of the employed building blocks. For example, the general structure of the building block *t* is given in Figure 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…n_{AB} (*number of collaborations between collaboration roles A and B*))*. The internal behavior of the same building block is shown in Figure 3(b). The activity *transfer$_{ij}$* (where *ij* = AB) describes the behavior of the corresponding collaboration. It has one activity partition for each collaboration role: *A* and *B*. Activities base their semantics on token flow [1]. The activity

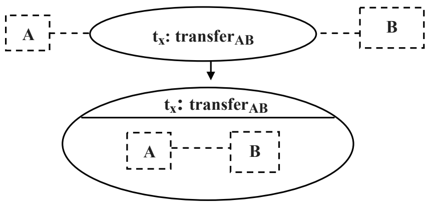


Figure 2. Structure of the building block

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

Composition of building block using UML collaboration and activity: 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 collaboration of the system. Collaboration is represented by connecting their input and output pins. Arbitrary logic between pins may be used to synchronize the building block events and transfer data between them. By connecting the individual input and output pins of the call behavior actions, the events occurring in different collaborations can be coupled with each other. Semantics of the different kinds of pins are given in more detailed in [14]. For example, the detailed behavior and composition of the collaboration is given in following Figure 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 and C)*. Completion of the processing by the participants are mentioned by the call operation action *Prd_i (Processing_done$_i$, where i = A, B and C).* After completion of the processing, the response is delivered to the corresponding participant. When the processing of the task by the participant *A* completes, the response (indicated by streaming pin *res)* is transferred to the participant *B* mentioned by collaboration *t: transfer$_{ij}$* (*where ij = AB)* and participant *B* starts the processing of the request (indicated by streaming pin *req).* After completion of processing participant B transfers the response to the participant *C* mentioned by collaboration *t: transfer$_{ij}$* (*where ij = BC)*. Participant *C* starts the processing after getting the response form *B* and activity is terminated after completion of the processing, which is illustrated by the terminating node (◉).

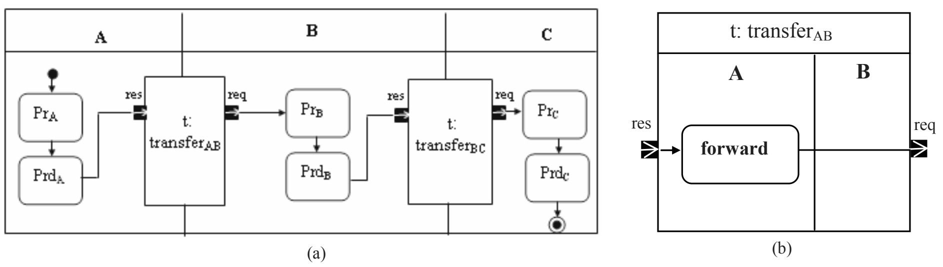


Figure 3(a). Detailed behavior of the event of the collaboration using activity (b). Internal behavior of the collaboration

Modeling failure and repair behavior of software and hardware components using STM: State transitions of a system element are described using 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. In this paper, STM is used to describe the failure and recovery events of software and hardware components. The STM of software process is shown in Figure 4(a). The initial node (•) indicates the starting of the operation of software process. Then the process enters Running state. Running is the only available state in the STM. If the software process fails during the operation, the process enters Failed state. When the failure is detected by the external monitoring service the software process enters Recovery state and the repair operation will be started. When the failure of the process is recovered the software process returns to Running state. The STM of hardware node is shown in Figure 4(b). States of the hardware node start from the Running state. Running is the only available state here. If the node fails during the operation, the node enters Failed state. When the failure is detected the repair operation of the hardware node is started. When the failure of the node is repaired the node returns to Running state. The hardware node operation is terminated by the off operation and enters Stop state. The hardware node starts the operation, when the on command is invoked and the node enters Running state.

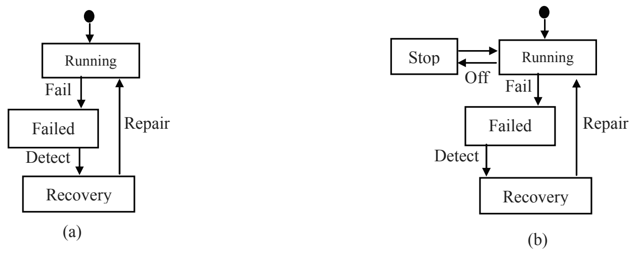


Figure 4(a). STM of software process (b). STM of hardware component

4 Deployment diagram and stating relation between system and service component

We model the system as collection of N interconnected nodes. Our objective is to find a deployment mapping for this execution environment for a set of service components C available for deployment that comprises the service. Deployment mapping can be defined as ($M:C \rightarrow N$) between a numbers of service components instances C, onto nodes N. We consider four types of requirements in the deployment problem: (1) Components have execution costs, (2) collaborations have communication costs and (3) costs for running of background process known as overhead cost (4) some of the components can be restricted in the deployment mapping to specific physical nodes, which are called bound components. We observe the processing cost that nodes impose while host the components and also the target balancing of cost among the nodes available in the network. Communication costs are considered if collaboration between two components happens remotely, i.e., it happens between two nodes [6]. In other words, if two components are placed onto the same physical node the communication cost between them will not be considered. The cost for executing the background process for conducting the communication between the components is always considerable no matter whether the components deploy on the same or different physical nodes. Using the above specified input, the deployment logic provides an optimal deployment architecture taking into account the QoS requirements for the components providing the specified service. We then define the objective of the deployment logic as obtaining an efficient (low-cost, if possible optimum) mapping of components onto the nodes that satisfies the requirements in reasonable time. The deployment logic providing optimal deployment architecture is guided by the cost function F(M). The cost function is designed here to reflect the goal of balancing the execution cost and minimizing the communications cost [6]. This is in turn utilized to achieve reduced task turnaround time by maximizing the utilization of resources while minimizing any communication between processing node. That will offer a high system throughput, taking into account the expected execution and inter-node communication requirements of the service components on the given hardware architectures, which is already highlighted in [13]. The evaluation of cost function F(M) is mainly influenced by our way of service definition. Service is defined in our approach as a collaboration of total E components labeled as c_i (where i = 1…. E) to be deployed and total K collaboration between them labeled as k_j, (where j = 1 … K). The execution cost of each service component can be labeled as 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 only observe the total cost ($\hat{l}_n$, n = 1…N) of a given deployment mapping at every node. We will strive for an optimal solution of equally distributed cost among the processing nodes and the lowest cost possible, while taking into account the execution cost 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 cost T [6]: $T = \frac{1}{|N|}\sum_{i=1}^{|E|} f_{c_i}$ (1)

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 [16]:

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

This means that $q_0(M,c)$ returns the node n that host component in the list mapping M. Let collaboration $k_j = (c_1, c_2)$. The communication cost of k_j is 0 if components c_1 and c_2 are collocated, i.e. $q_0(M,c_1) = q_0(M,c_2)$, and the cost is f_{k_j} if components are otherwise (i.e. the collaboration is remote). Using an indicator function *I(x)*, which is 1 if x is true and 0 otherwise, this expressed as $I(q_0(M,c_1) \neq q_0(M,c_2)) = 1$, if the collaboration is remote and 0 otherwise. To determine which collaboration k_j is remote, the set of mapping M is used. Given the indicator function, the overall communication cost of service, $F_K(M)$, is the sum [16]:

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

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

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

5 Annotation

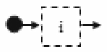
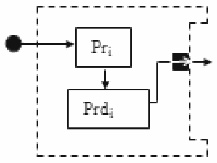
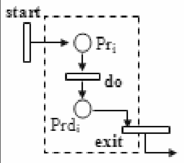
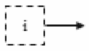
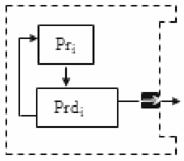
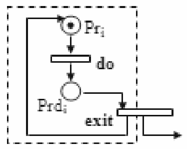
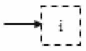
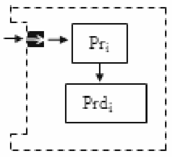
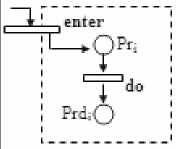
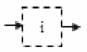
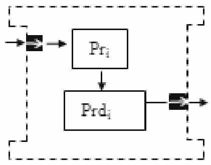
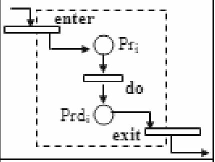
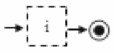
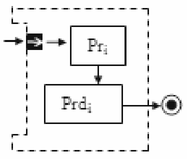
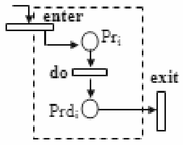
In order to annotate the UML diagram, the stereotypes *SaStep, ComputingResource, scheduler, QoSDimension* and the tagged values *execTime, deadline, mean-time-to-repair, mean-time-between-failures* and *schedPolicy* are used according to the *UML profile for MARTE* and *UML Profile for Modeling Quality of Service and Fault Tolerance Characteristics* [7],[12]. *SaStep* is a kind of step that begins and ends, when decisions about the allocation of system resources are made. The duration of the execution time is mentioned by the 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. A *ComputingResource* represents either virtual or physical processing devices capable of storing and executing program code. Hence, its fundamental service is to compute. A *Scheduler* is defined as a kind of ResourceBroker that brings access to its brokered ProcessingResource or resources following a certain scheduling policy tagged by *schedPolicy.* The ResourceBroker is a kind of resource that is responsible for allocation and de-allocation of a set of resource instances (or their services) to clients according to a specific access control policy [7]. *QoSDimension* provides support for the quantification of QoS characteristics and attributes *mean-time-to-repair* and *mean-time-between-failures* [12]. We also introduce a new stereotype *Transition* and three tagged values *mean-time-to-stop, mean-time-to-start* and *mean-time-to-failure-detect*. *Transition* induces a state transition of a scenario. *mean-time-to-stop* defines the mean time required to stop working of a hardware instance, *mean-time-to-start* states the mean time required to start working of a hardware instance, *mean- time-to-failure-detect* defines the mean time required to detect failures in the system.

6 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 SRN we will give a brief introduction about SRN model. SRN is based on the Generalized Stochastic Petri net (GSPN) [3] and extends them further by introducing prominent extensions such as guard function, reward function and marking dependent firing rate [5]. A guard function is assigned to a transition. It specifies the condition to enable or disable the transition and can use the entire state of the net rather than just the number of tokens in places [5]. Reward function defines the reward rate for each tangible marking of Petri Net based on which various quantitative measures can be done in the Net level. Marking dependent firing rate allows using the number of token in a chosen place multiplying the basic rate of the transition. SRN model has the following elements: Finite set of the place (drawn as circle), Finite set of the transition defined as either timed transition (drawn as thick transparent bar) or immediate transition (drawn as thick black bar), set of the arc connecting places and transition, multiplicity associated with the arcs, and marking that denotes the number of token in each place.

Before introducing the translation rules different types of collaboration roles as reusable basic building blocks are demonstrated with the corresponding SRN model in Table 1 that can be utilized to form the collaborative building blocks.

Table 1. Specification of reusable unites and their SRN model

Type	Representation of Collaboration role	Activity diagram as reusable specification units	Equivalent SRN model
1			
2			
3			
4			
5			

The rules are the following:

Rule1: The SRN model of a collaboration (Figure 5), where collaboration connects only two collaboration roles, is formed by combining the basic building blocks type 2 and type 3 from Table 1. 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 and overhead cost.

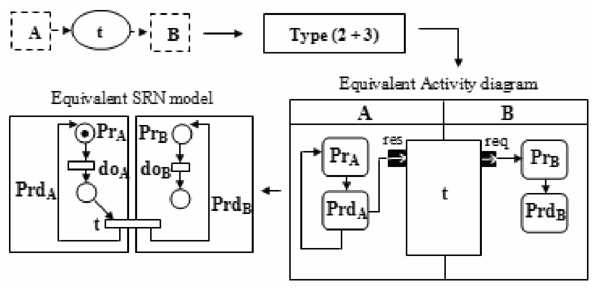


Figure 5. Graphical representation of Rule 1

In the same way, SRN model of the collaboration can be demonstrated, where the starting of the execution of the SRN model of collaboration role A depends on the 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 the utilizing the basic building block of Table 1, which is shown in Figure 6. In the first diagram in Figure 6, if component A contains its own token equivalent SRN model of the collaboration role A will be formed using basic building block type 1 from Table 1. The same applies to the component B and C in the second diagram in Figure 6.

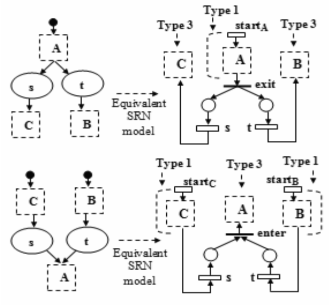


Figure 6. Graphical representation of Rule 2

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

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

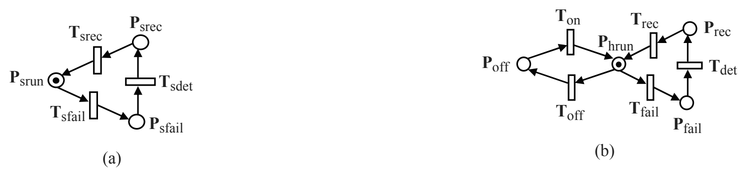


Figure 7(a). SRN of software process (b). SRN of hardware component

7 Model synchronization

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

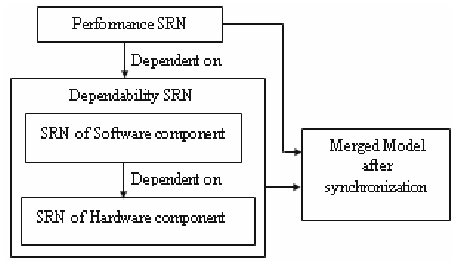

Performance SRN
Dependent on
Dependability SRN
SRN of Software component
Dependent on
SRN of Hardware component
Merged Model after synchronization

Figure 8: Model synchronization hierarchy

The model synchronization is focused in details here:

Synchronization between the dependability SRN models in the dependability modeling layer: SRN model for the software process (Figure 7(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

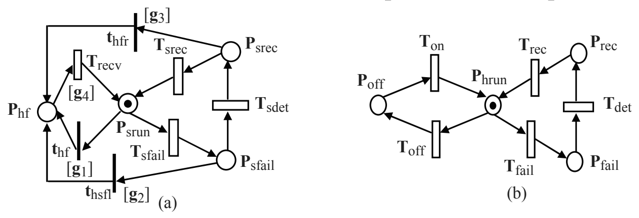


Figure 9. (a) Synchronized transition in the SRN model of the software process with the (b) SRN model of the hardware component

(Figure 9(a)) is 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_{hfl}, t_{hfr} will be enabled only, when the hardware node (in Figure 9 (b)) fails as failure of hardware node will stop the operation of software process. The timed transition T_{recv} will be enabled only, when the hardware node will again start working after being recovered from failure. Four guard functions $\boldsymbol{g_1, g_2, g_3, g_4}$ 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 3.

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

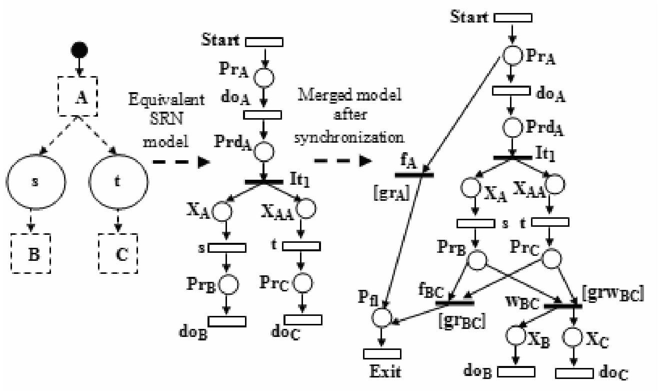


Figure 10. Synchronize the performance SRN model with dependability SRN

Performance SRN model of parallel execution of collaboration roles are expanded by incorporating one additional place P_{fl} and immediate transitions f_{BC}, w_{BC} shown in Figure 10. In our discussion, during the synchronization of the parallel processes it needs to ensure that failure of one process eventually stop providing service to the users. This could be achieved by immediate transition f_{BC}. If software resp. hardware components (Figure 9) 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 inconsistent execution of the whole SRN and produce erroneous result. If both the software and hardware components work fine the timed transition w_{BC} will fire to continue the execution of parallel processes B and C. Guard functions gr_{BC}, grw_{BC} allow the immediate transition $f_{BC,} w_{BC}$ to work consistently with the change of the states of the software and hardware components. The guard function definitions are shown in the Table 3.

8 Hierarchical model for MTTF calculation

It is very demanding and not efficient with respect to execution time to consider behavior of all the hardware components during the SRN model generation. SRN model becomes very cumbersome and inefficient to execute. 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 (Figure 7(b)) rather than considering failure behavior of all the hardware components individually. The below Figure 11 introduces one example scenario of capturing failure behavior of the hardware components using fault tree, where system is composed of different hardware devices such as one CPU, two memory interfaces, one storage device and one cooler. The system will work, when CPU, one of the memory interfaces, storage device, and cooler will run. Failure of both memory interfaces or failure of either CPU or storage device or cooler will result in system unavailability.

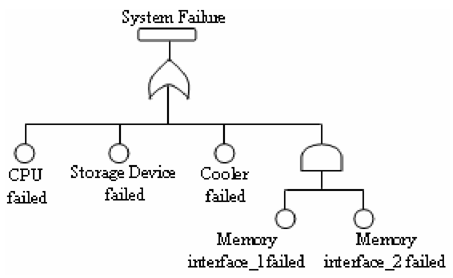


Figure 11. Fault tree model of system failure

9 Case study

As a representative example, we consider the scenario dealing with heuristically clustering of modules and assignment of clusters to nodes [16]. This scenario is sufficiently complex to show the applicability of our 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 depicted in Figure 12. We consider four types of requirements in this specification. Besides the execution cost, communication costs and cost for running background process, we have a restriction on components C_2, C_7, C_9 regarding their location. They must be bound to nodes $\boldsymbol{n_2}$, $\boldsymbol{n_1}$, $\boldsymbol{n_3}$ respectively. In this scenario, new service is generated by integrating and combining the existing service components that will be delivered conveniently by the system. For example, one new service is composed by combining the service components C_1, C_6, C_7, C_8, and $C9$ shown in Figure 12 as thick dashed line.

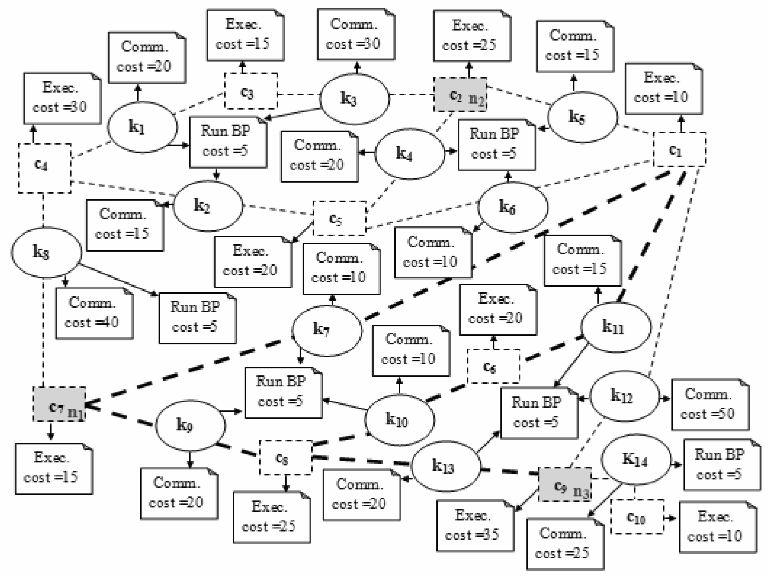


Figure 12. Collaboration & components in the example scenario

The internal behavior of the collaboration K_i is realized by the call behavior actions through UML activity like structure already demonstrated in Figure 3(b). The composition of the collaboration role C_i of the delivered service by the system is demonstrated in Figure 14. The initial node (●) indicates the starting of the activity. After being activated, each participant starts its processing of request, which is mentioned by call behavior action Pr_i (Processing of the ith service component). Completions of the processing by the participants are mentioned by the call behavior action Prd_i (Processing done of the ith service component). The activity is started from the component C_1, where the semantics of the activity is realized by the token flow. After completion of the processing of the component C_1 the response is divided into two flows, which are shown by the fork node f_1. The flows are activated towards component C_7 and C_6. After getting the response from the component C_1, processing of the components C_7 and C_6 will be

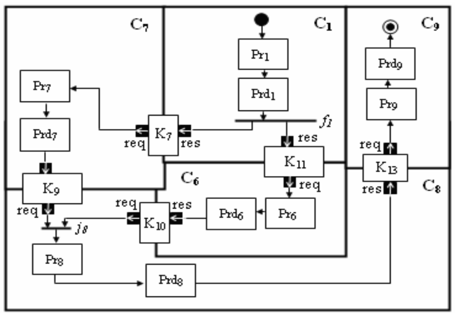


Figure 14. Service composition & detail behavior of the event of the collaboration using activity

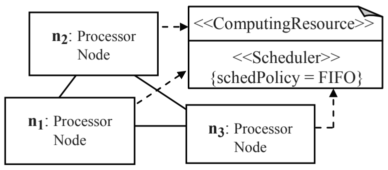


Figure 13. The target network of hosts

started. The response and request are mentioned by the streaming pin *res* and *req*. The processing of the Component C_8 will be started after getting the responses from both component C_7 and C_6, which is realized by the join node j_8. After completion of the processing of component C_8 component C_9 starts its processing and later on, activity is terminated, which is mentioned by the end node (◉).

In this example, the target environment consists of $N = 3$ identical, interconnected nodes with no failure of network link, with a single provided property, namely processing power, and with infinite communication capacities depicted in Figure 13. The optimal deployment mapping can be observed in Table 2. The lowest possible deployment cost, according to equation (4) is: 17 + 100 + 70 = 187.

Table 2. Optimal deployment mapping

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

In order to annotate the UML diagrams in Figure 13 and 14 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 (Figure 14) is associated with two instances of *deadline* as collaborations in example scenario are associated with two kinds of cost: communication cost and cost for running background process (BP). To annotate the STM UML diagram of software process (shown in Figure 15) we use the stereotype *<<QoSDimension>>, <<transition>>* and attributes *mean-time-between-failures, mean-time-to-failure-detect* and *mean-time-to-repair* already mentioned in Section 5. Annotation of the STM of hardware component can be demonstrated in the same way as STM of software process.

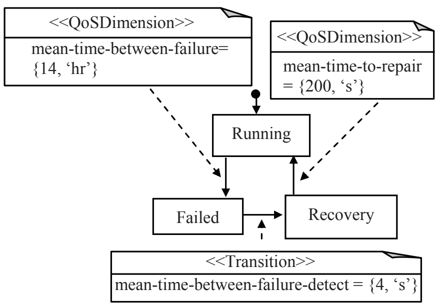

<<QoSDimension>>
mean-time-between-failure= {14, 'hr'}
<<QoSDimension>>
mean-time-to-repair = {200, 's'}
Running
Failed
Recovery
<<Transition>>
mean-time-between-failure-detect = {4, 's'}

Figure 15. Annotated STM diagram of software

By considering the deployment mapping and the transformation rules the analogous SRN model of our example service (in Figure 14) is depicted in Figure 16. In our discussion, we consider M/M/1/n queuing system so that at most n jobs can be in the system at a time [3]. For generating the SRN model, firstly, we will consider the starting node (•). According to rule 1, it is represented by timed transition (denoted as start) and the arc connected to place Pr_1 (states of component C_1). When a token is deposited in place Pr_1, immediately, a checking is done about the availability of both software and hardware components by inspecting the corresponding SRN models (Figure 9). The availability of software and hardware components allow the firing of timed transition t_1 mentioning the continuation of the further execution. Otherwise, immediate transition f_1 will be fired mentioning the ending of the further execution because of software resp. hardware component failure. The enabling of immediate transition f_1 is realized by the guard function $\boldsymbol{gr_1}$. After the completion of the state transition from Pr_1 to Prd_1 (states of component C_1) the flow is divided into two branches (denoted by the immediate transition It_1) according to rule 2. The token will be deposited to place Pr_7 (states of component C_7) and Pr_6 (states of component C_6) after the firing of transitions K_7 and K_{11}. The collaboration K_7 is realized both by the communication cost and cost for running background process as C_1 and C_7 deploy on the two different nodes $\boldsymbol{n_3}$ and $\boldsymbol{n_1}$. According to rule 1, collaboration K_{11} is realized only by the cost for running background process as C_1 and C_6 deploy on the same processor node $\boldsymbol{n_3}$. When a token is deposited into place Pr_7 and Pr_6, immediately, a checking is done about the availability of both software and hardware components by inspecting the corresponding dependability SRN models (Figure 9). The availability of software and hardware components allow the firing of immediate transition w_{76}, which eventually enables the firing of timed transition t_7 and t_6

Table 3. Guard functions definitions

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

mentioning the continuation of the further execution. The enabling of immediate transition w_{76} is realized by the guard function $\boldsymbol{grw_{76}}$. Otherwise, immediate transition f_{76} will be fired mentioning the ending of the further execution because of failure of software resp. hardware component. The enabling of immediate transition f_{76} is realized by the guard function $\boldsymbol{gr_{76}}$. After the completion of the state transition from Pr_7 to Prd_7 (states of component C_7) and from Pr_6 to Prd_6 (states of component C_6) component C_8 starts processing. The merging of result is realized by the immediate transition It_2 after the firing of transitions K_9 and K_{10}. Collaboration K_9 is realized only by the cost for running background process as C_7 and C_8 deploy on the same processor node $\boldsymbol{n_1}$. K_{10} is translated by the timed transition, which is realized both by the communication cost and cost for running background process as C_6 and C_8 deploy on the two different nodes $\boldsymbol{n_3}$ and $\boldsymbol{n_1}$. When a token is deposited in place Pr_8, immediately a checking is done about the availability of both software and hardware components by inspecting the corresponding SRN models (Figure 9). The availability of software and hardware components allow the firing of timed transition t_8 mentioning the continuation of the further execution. Otherwise, immediate transition f_8 will be fired mentioning the ending of the further execution because of software resp. hardware component failure. The enabling of immediate transition f_8 is realized by the guard function $\boldsymbol{gr_8}$. After the completion of the state transition from Pr_8 to Prd_8 (states of component C_8) the token is passed to place Pr_9 by firing of timed transition K_{13}. K_{13} is realized by both communication cost and cost for running background process as C_8 and C_9 deploy on the two different nodes $\boldsymbol{n_1}$ and $\boldsymbol{n_3}$. When a token is deposited in place Pr_9, immediately a checking is done about the availability of both software and hardware components by inspecting the corresponding SRN models (Figure 9). The availability of software and hardware components allow the firing of timed transition t_9 mentioning the continuation of the further execution. Otherwise, immediate transition f_9 will be fired mentioning the ending of the further execution because of software resp. hardware component failure and the ending of the execution of the SRN model is realized by the timed transition $Exit_2$. The enabling of immediate transition f_9 is realized by the guard function $\boldsymbol{gr_9}$. After the completion of the state transition from Pr_9 to Prd_9 (states of component C_9) the ending of the execution of the SRN model is realized by the timed transition $Exit_1$. The definition of guard functions are shown in Table 3 (P_{hrun} and P_{srun} are shown in Figure 9).

We use SHARPE [15] to execute the obtained model and calculate the system's throughput. The throughput of successful jobs can be computed by checking the throughput of the transition $Exit_1$ by SHARPE [15]. The throughput result is summarized in Table 4 and graph in Figure 17 shows throughput variation of the system against the change of failure rate (sec^{-1}) of both hardware and software components.

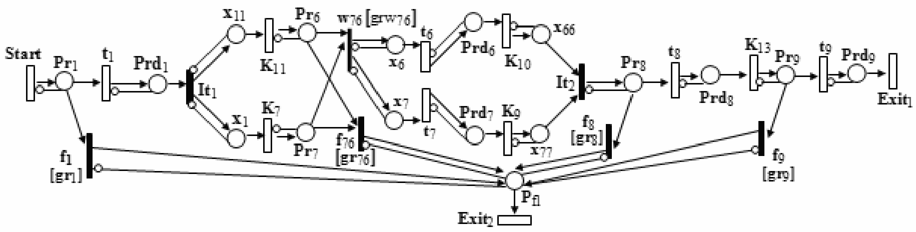


Figure 16. SRN model of the example service

10 Conclusion and future work

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

Table 4. Throughput calculation

	Throughput
Performability model	0.0095
Performance model	0.01385

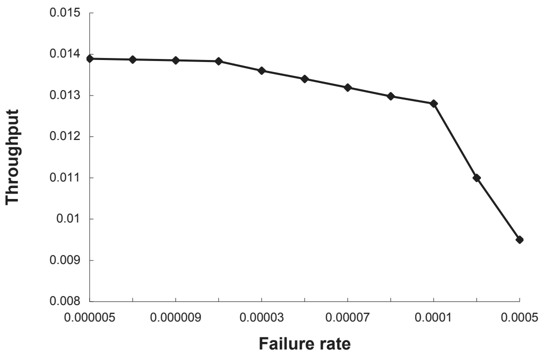


Figure 17. Numerical result of our example scenario

References

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

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

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

[4] OMG 2009, “UML Superstructure”, Version-2.2

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

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

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

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

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

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

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

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

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

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

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

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

Paper 8

A performability modeling framework considering service components deployment

Razib Hayat Khan, Fumio Machida, Poul E. Heegaard, Kishor S. Trivedi

Internal Journal on Advances in Networks and Services, Volume – 5, Number – 3&4, p. 346-366, 2012

A performability modeling framework considering service components deployment

Razib Hayat Khan[1], Fumio Machida[2], Poul E. Heegaard[1], Kishor S. Trivedi[3]

[1]Department of Telematics
Norwegian University of Science and Technology (NTNU)
7491, Trondheim, Norway
{rkhan, poul.heegaard }@item.ntnu.no

[2]Service Platform Research
NEC, Japan
h-machida@ab.jp.nec.com

[3]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.

1 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 activities of 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 (SAN) notation [10]. Subsequent effort proposes a methodology for the modeling, verification, and performance evaluation of communication components of 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 2 introduces our performability modeling framework, Section 3 depicts UML model description, Section 4 describes formalization of UML models, Section 5 explains service components deployment issue, Section 6 clarifies UML models annotations, Section 7 delineates model transformation rules, Section 8 introduces the model synchronization mechanism, Section 9 describes the hierarchical method for mean time to failure (MTTF) calculation, Section 10 indicates the tool based support of the modeling framework, Section 11 illustrates the case study, and Section 12 delineates the concluding remarks with future directions.

2 Overview of the performability 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 Figure 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.

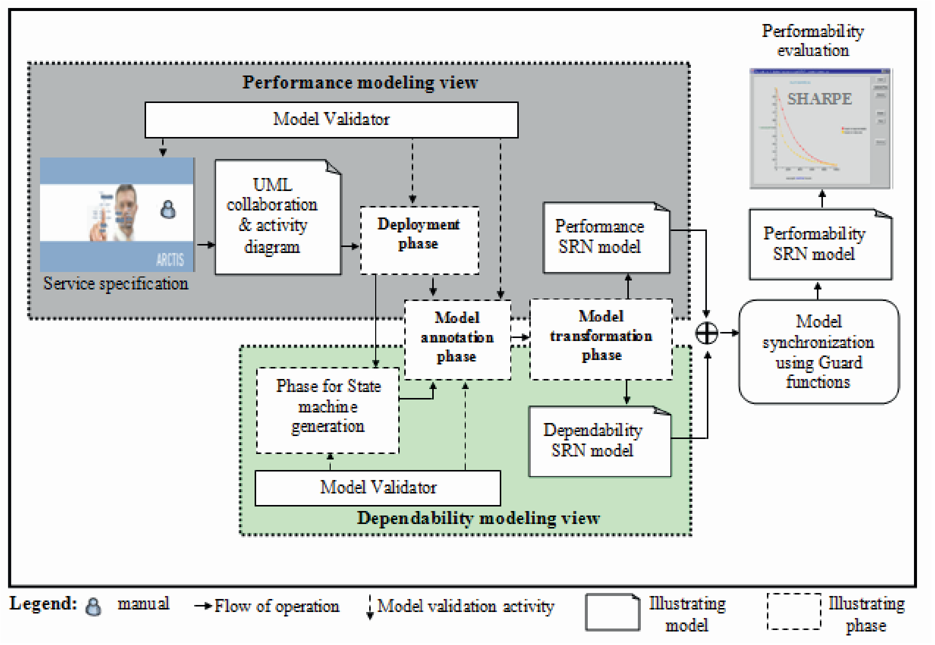

Performability evaluation
SHARPE
Performance modeling view
Model Validator
UML collaboration & activity diagram
Deployment phase
Service specification
Performance SRN model
Performability SRN model
Model annotation phase
Model transformation phase
Model synchronization using Guard functions
Phase for State machine generation
Dependability SRN model
Model Validator
Dependability modeling view
Legend: manual
Flow of operation
Model validation activity
Illustrating model
Illustrating phase

Figure 1. Performability modeling framework

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 Figure 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].

3 UML based system description

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 Figure 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…*n_{AB} *(*number of collaborations between collaboration roles *A & B))*. The internal behavior of the same building block is shown in Figure 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

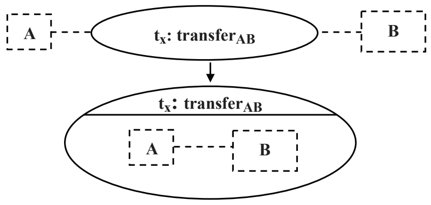


Figure 2. Structure of the building block

(indicated by the streaming pin *res*) to transfer 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

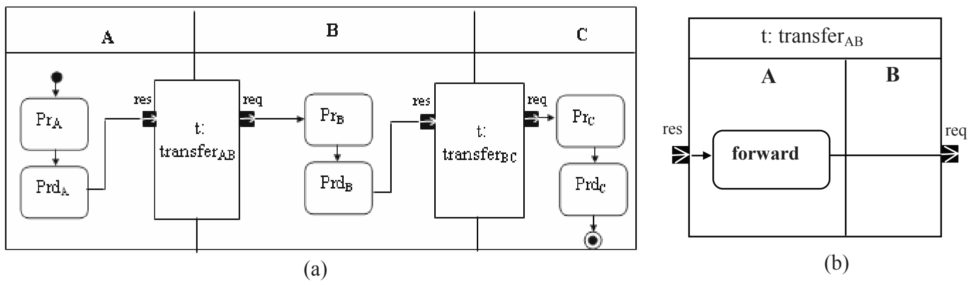


Figure 3. (a) Detailed behavior of the event of the collaboration using activity (b) internal behavior of the collaboration

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 Figure 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 (◉).

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.

The STM of software process is shown in Figure 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 Figure 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.

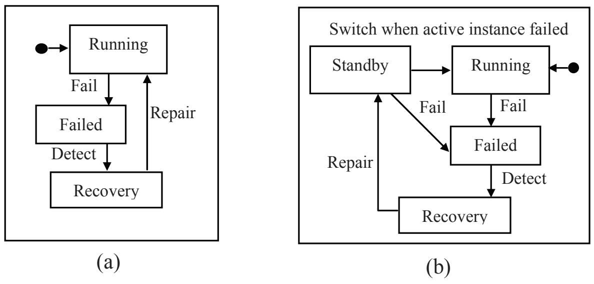


Figure 4. (a) STM of software process (b) STM of hardware component

4 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].

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 at 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].

Formalizing state machine diagram using cTLA: We sketch the cTLA model of STM in Figure 5 by the specification of software process dependability behavior illustrated in Figure 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 state 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 fulfill 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 inserted into the data structure *addEvent*, which is a

```
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 .........
```

Figure 5. cTLA process of software component

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].

5 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]:

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

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]:

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

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]:

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

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 $\hat{l}_n=\sum\nolimits_{c_i\in m_n} f_{c_i}$. 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):

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

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

6 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.

Figure 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

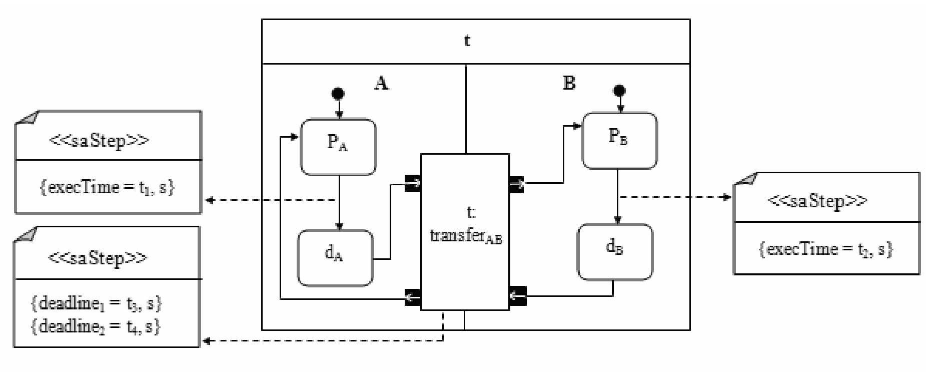


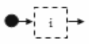
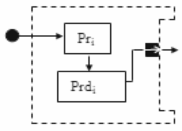
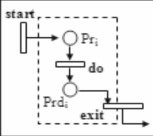
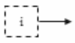
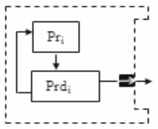
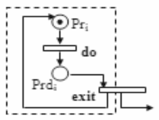
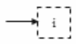
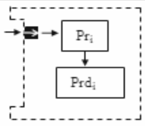
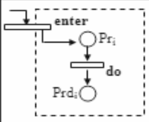
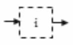
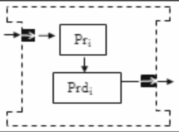
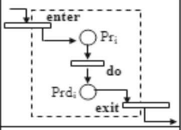
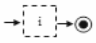
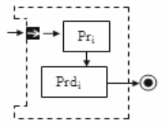
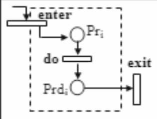
Figure 6. Annotated UML

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* – *deadline₁* defines the communication time and *deadline₂* is for overhead time.

7 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 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.

Table 1. Specification of reusable unites and their SRN model

Type	Representation of Collaboration role	Activity diagram as reusable specification units	Equivalent SRN model
1			
2			
3			
4			
5			

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 (Figure 7), where collaboration connects only two collaboration roles, is formed by combining the basic building blocks type 2 and type 3 from Table 1. 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.

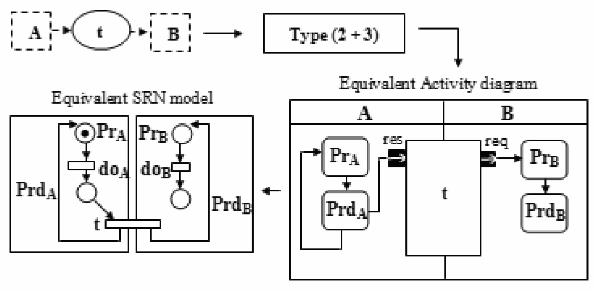


Figure 7. Graphical representation of Rule 1

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 1, which is shown in Figure 8. In the first diagram of Figure 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 1. The same applies to the component B and C in the second diagram in Figure 8.

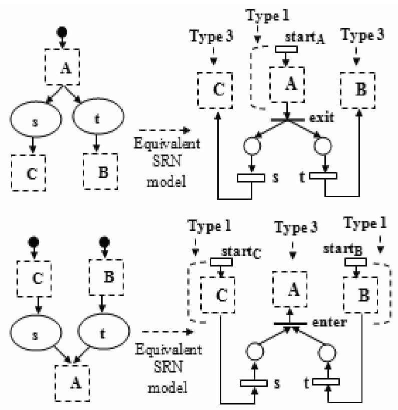


Figure 8. Graphical representation of Rule 2

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 Figure 9.

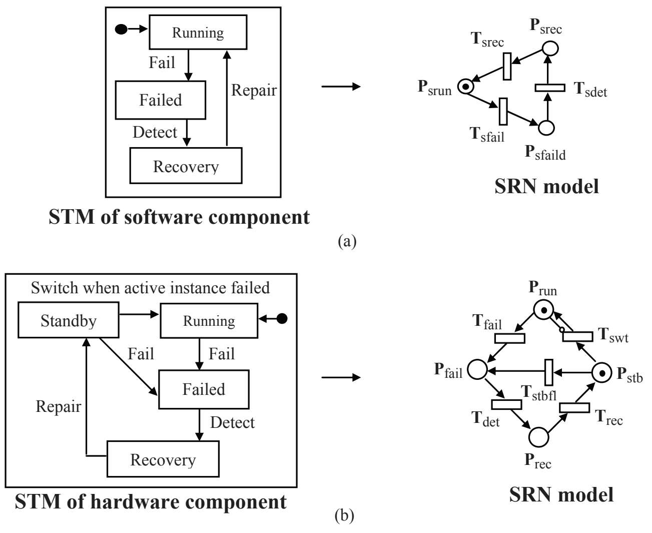


Figure 9 (a) SRN of Software process (b) SRN of hardware component

The SRN model for hardware component is shown in Figure 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.

8 Model synchronization

The model synchronization is achieved hierarchically, which is illustrated in Figure 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].

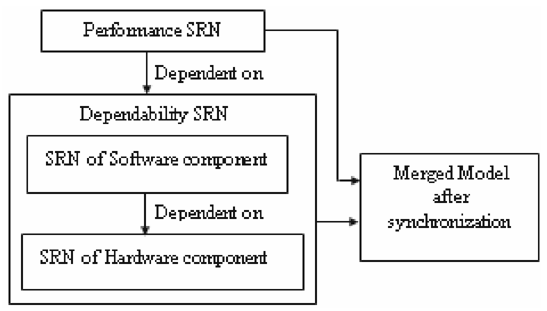

Performance SRN
Dependent on
Dependability SRN
SRN of Software component
Dependent on
SRN of Hardware component
Merged Model after synchronization

Figure 10. Model synchronization hierarchy

The model synchronization is focused in detail below:

Synchronization between the dependability SRN models in the dependability modeling layer: SRN model for the software process (Figure 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

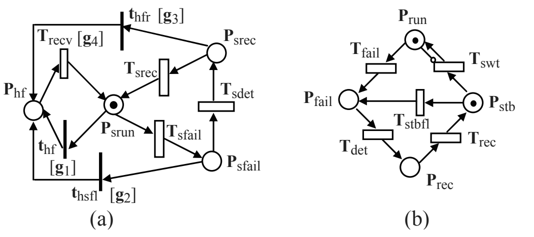


Figure 11. (a) Synchronized transition in the SRN model of the software process with the (b) SRN model of the hardware component

Table 2. Guard functions definitions

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

(Figure 11(a)) is associated with four additional arcs such as (P_{sfail} × t_{hsfl}) ∪ (t_{hsfl} × P_{hf}), (P_{srec} × t_{hfr}) ∪ (t_{hfr} × P_{hf}), (P_{srun} × t_{hf}) ∪ (t_{hf} × P_{hf}) and (P_{hf} × T_{recv}) ∪ (T_{recv} × P_{srun}). The immediate transitions t_{hf}, t_{hsfl}, t_{hfr} will be enabled only, when the hardware node (in Figure 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 2.

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 Figure 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 Figure 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

(Figure 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

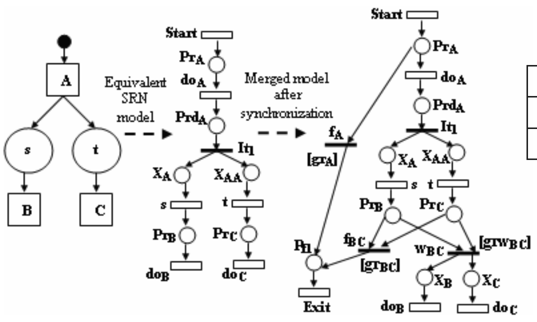


Figure 12. Synchronize the performance SRN model with dependability SRN

Table 3. Guard functions definitions

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

the immediate transition f_{BC}, w_{BC} to work consistently with the change of the states of the software and hardware components. The guard function definitions are shown in the Table 3. Algorithms for model transformation rules and model synchronization process have been mentioned in Appendix A.

9 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 (Figure 9(b)) rather than considering failure behavior of all the hardware components individually. The below Figure 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.

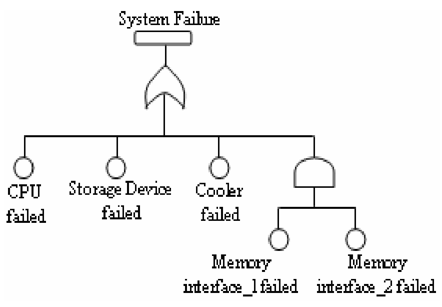


Figure 13. Fault tree model of system failure

10 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 Figure 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

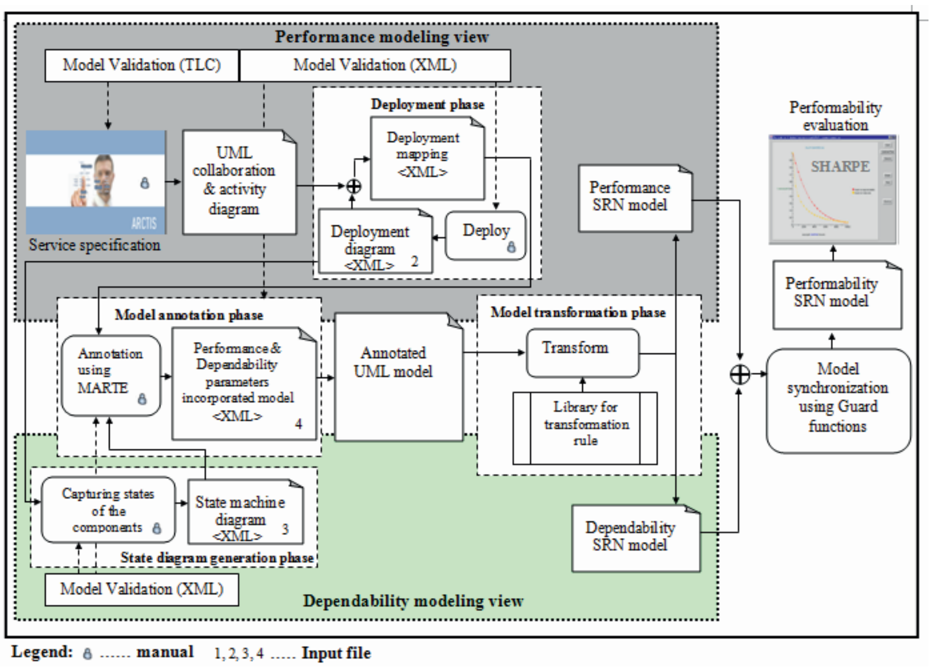

Performance modeling view
Model Validation (TLC)
Model Validation (XML)
Deployment phase
Deployment mapping <XML>
UML collaboration & activity diagram
Service specification
Deployment diagram <XML> 2
Deploy
Performability evaluation
SHARPE
Performance SRN model
Performability SRN model
Model annotation phase
Annotation using MARTE
Performance & Dependability parameters incorporated model <XML> 4
Annotated UML model
Model transformation phase
Transform
Library for transformation rule
Model synchronization using Guard functions
Capturing states of the components
State machine diagram <XML> 3
State diagram generation phase
Dependability SRN model
Model Validation (XML)
Dependability modeling view
Legend: 🔒 ...... manual 1, 2, 3, 4 ..... Input file

Figure 14. Tool support of our performability modeling framework

parameters. From Figure 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.

11 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 . . . C_{10}) to be deployed and K = 14 collaborations between them illustrated in Figure 15. We consider three types of requirements in this specification.

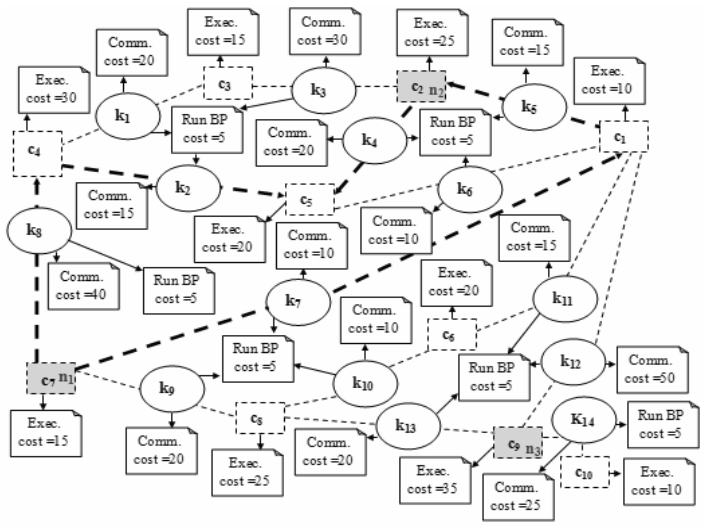


Figure 15. Collaboration & components in the example scenario

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 composed by combining the

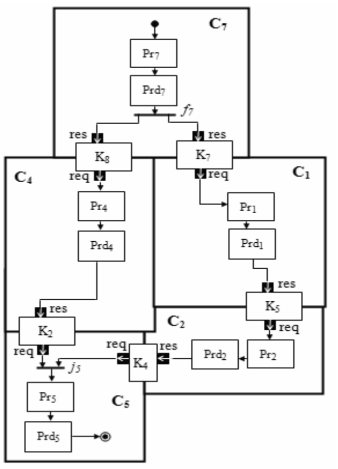


Figure 16. Composition of collaboration

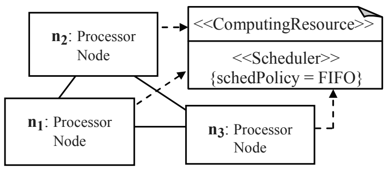


Figure 17. The target network of hosts

$\mathbf{C_1}$	<<SaStep>
	{execTime=10, s}

$\mathbf{K_1}$	<<SaStep>>
	{deadline$_1$=20, s}
	{deadline$_2$=5, s}

Figure 18. Annotated UML model

service components C_1, C_2, C_4, C_5, C_7 shown in Figure 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 Figure 3(b). The composition of the collaboration role C_i of the delivered service by the system is demonstrated in Figure 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 Figure 17. The optimal deployment mapping can be observed in Table 4. The lowest possible deployment cost, according to equation (4) is: $17 + 100 + 70 = 187$.

Table 4. 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

In order to annotate the UML diagrams in Figure 16 and 17, we use the stereotypes *<<SaStep>> <<ComputingResource>>, <<Scheduler>>* and the tagged values *execTime, deadline* and *schedPolicy,* which are already explained in Section 6. Collaboration K_i (Figure 18) is associated with two instances of *deadline* as collaborations in example scenario are associated with two kinds of cost: communication cost and cost for running background process (BP). In order to annotate the STM UML diagram of software process (shown in Figure 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 6. Annotation of the STM of hardware component can be demonstrated in the same way as STM of software process.

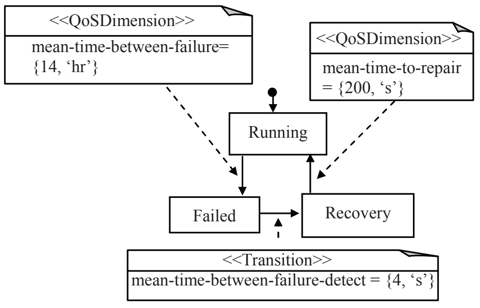

<<QoSDimension>>
mean-time-between-failure= {14, 'hr'}
<<QoSDimension>>
mean-time-to-repair = {200, 's'}
Running
Failed
Recovery
<<Transition>>
mean-time-between-failure-detect = {4, 's'}

Figure 19. Annotated STM diagram of software component

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 Figure 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 Figure 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

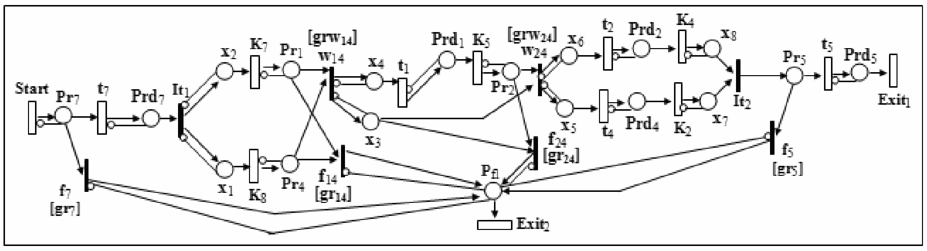


Figure 20. SRN model of the example service

to model transformation rule 2 (Figure 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 4). 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 4). 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 Figure 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 Figure 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 4). 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 4). 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 Figure 11. The availability of software and hardware components allows the firing of timed transition t_5

mentioning 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 5, which is dependent on the execution of the SRN model of the corresponding STM of software and hardware instances illustrated in Figure 11.

Table 4. 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 Figure 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.

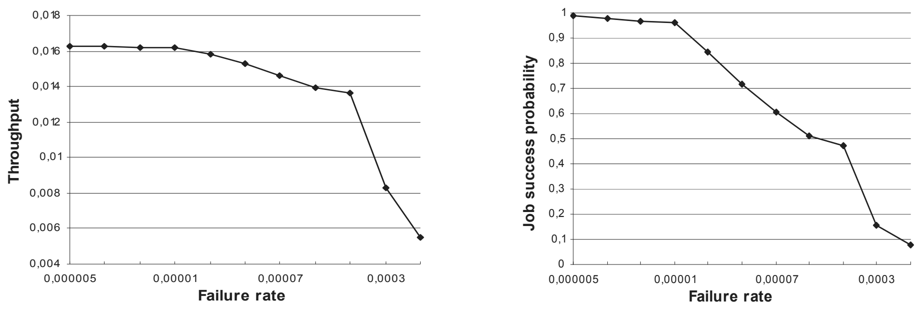


Figure 21. Numerical result of our example scenario

12 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.

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
16   else
17        Timed_Transitions += “$t_B$ ind” + 1/ (overhead cost + communication cost)
18   if CollaborationRoles A and C are deployed on the same node then
19        Timed_Transitions += “$t_C$ ind” + 1/ overhead cost
20   else
21        Timed_Transitions += “$t_C$ ind”+1/ (overhead cost + communication cost)
22   Input_Arcs += “$Pr_A$ $do_A$ 1”
23   Input_Arcs += “$Prd_A$ it 1”
24   Input_Arcs += “$Pr_B$ $do_B$ 1”
25   Input_Arcs += “$Pr_C$ $do_C$ 1”
26   Input_Arcs += “$X_B$ $t_B$ 1”
27   Input_Arcs += “$X_C$ $T_C$ 1”
28   Output_Arcs += “Start $Pr_A$ 1”
29   Output_Arcs += “$do_A$ $Prd_A$ 1”
30   Output_Arcs += “it $X_b$ 1”
31   Output_Arcs += “it $X_c$ 1”
32   Output_Arcs += “$t_B$ $Pr_B$ 1”
33   Output_Arcs += “$t_C$ $Pr_C$ 1”
34   Output_Arcs += “$do_B$ $Prd_B$ 1”
35   Output_Arcs += “$do_C$ $Prd_C$ 1”
36   Inhibitor_Arcs += “$Pr_A$ Start 1”
37   Inhibitor_Arcs += “$Prd_A$ $do_A$ 1”
38   Inhibitor_Arcs += “$X_b$ it 1”
39   Inhibitor_Arcs += “$X_c$ IT 1”
40   Inhibitor_Arcs += “$Pr_B$ $t_B$ 1”
41   Inhibitor_Arcs += “$Pr_C$ $t_C$ 1”
42   Inhibitor_Arcs += “$Prd_B$ $do_B$ 1”
43   Inhibitor_Arcs += “$Prd_C$ $do_C$ 1”
44   Print Places, Immediate_Transitions, Timed_Transitions, Input_Arcs, Output_Arcs, Inhibitor_Arcs
45   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: parallal_process_sync_srn()

```
Places += “$Pr_A$ 0”
Places += “$Prd_A$ 0”
Places += “$Xa_1$ 0”
Places += “$Xa_2$ 0”
Places += “P_fl 0”
Places += “$\mathrm{Pr}_B$ 0”
Places += “$Prd_B$ 0”
Places += “$Pr_C$ 0”
Places += “$Prd_C$ 0”
Places += “$X_B$ 0”
Places += “$X_C$ 0”
Immediate_Transitions += “it ind 1”
Immediate_Transitions += “$f_{BC}$ ind 1 guard sw_up ()”
Immediate_Transitions += “$f'_{BC}$ ind 1 guard sw_down ()”
Timed_Transitions += “Start ind” + 1 / Start transition rate
if CollaborationRoles A and B are deployed on the same node then
    Timed_Transitions += “$T_B$ ind” + 1/ overhead cost
else
    Timed_Transitions += “$T_B$ ind” + 1/ (overhead cost + communication cost)
if CollaborationRoles A and C are deployed on the same node then
    Timed_Transitions += “$T_C$ ind” + 1/ overhead cost
else
    Timed_Transitions += “$T_C$ ind” + 1/ (overhead cost + communication cost)
Timed_Transitions += “End ind” + 1 / End transition rate
Input_Arcs += “$Pr_A$ $do_A$ 1”
Input_Arcs += “$Prd_A$ it 1”
Input_Arcs += “$Xa_1$ $T_B$ 1”
Input_Arcs += “$Xa_2$ $T_C$ 1”
Input_Arcs += “$Pr_B$ $f_{BC}$ 1”
Input_Arcs += “$Pr_B$ $f'_{BC}$ 1”
Input_Arcs += “$Pr_C$ $f_{BC}$ 1”
Input_Arcs += “$Pr_C$ $f'_{BC}$ 1”
Input_Arcs += “P_fl End 1”
Input_Arcs += “$X_B$ $do_B$ 1”
Input_Arcs += “$X_C$ $do_C$ 1”
Output_Arcs += “Start $Pr_A$ 1”
Output_Arcs += “$do_A$ $Prd_A$ 1”
Output_Arcs += “it $Xa_1$ 1”
Output_Arcs += “it $Xa_2$ 1”
Output_Arcs += “$T_B$ $Pr_B$ 1”
Output_Arcs += “$T_C$ $Pr_C$ 1”
Output_Arcs += “$f_{BC}$ $X_B$ 1”
Output_Arcs += “$f_{BC}$ XC 1”
Output_Arcs += “$f'_{BC}$ P_fl 1”
Output_Arcs += “$do_B$ $Prd_B$ 1”
Output_Arcs += “$do_C$ $Prd_C$ 1”
Inhibitor_Arcs += “$Pr_A$ Start 1”
Inhibitor_Arcs += “$Prd_A$ $do_A$ 1”
Inhibitor_Arcs += “$Xa_1$ it 1”
Inhibitor_Arcs += “$Xa_2$ it 1”
Inhibitor_Arcs += “$Pr_B$ $T_B$ 1”
Inhibitor_Arcs += “$Pr_C$ $T_C$ 1”
Inhibitor_Arcs += “P_fl $f'_{BC}$ 1”
Inhibitor_Arcs += “$X_B$ $f_{BC}$ 1”
Inhibitor_Arcs += “$X_C$ $f_{BC}$ 1”
Inhibitor_Arcs += “$Prd_B$ $do_B$ 1”
Inhibitor_Arcs += “$Prd_C$ $do_C$ 1”
Print Places, Timed_Transitions, Immediate_Transitions, Input_Arcs, Output_Arcs, Inhibitor_Arcs
return
```

```
sw_up()

    if place S_run has one token then
        return TRUE
    else
        return FALSE
    return
```

```
sw_down()

    if place S_run has zero token then
        return TRUE
    else
        return FALSE
    return
```

Algorithm 9: basic_bulding_block_srn()

```
    if CollaborationRoles A has a self token generator then
        Places += "$Pr_i$ 1"
    else
        Places += "$Pr_i$ 0"
    Places += "$Prd_i$ 0"
    Timed_Transitions += "do ind " + 1/execution cost for collaboration role i
    if i is getting token from external token generator then

        Timed_Transitions += "Start ind" + 1 / Start rate
        Output_Arcs += "Start $Pr_i$ 1"
        Inhibitor_Arcs += "$Pr_i$ Start 1"
        Inhibitor_Arcs += "$Prd_i$ do 1"
    else if i is getting token from another CollaborationRoles

        Timed_Transitions += "Enter ind" + 1 / cost of the transition
        Output_Arcs += "Enter $Pr_i$ 1"
    else

        Output_Arcs += "Exit $Pr_i$ 1"
        Input_Arcs += "$Pr_i$ do 1"
    if i is passing its token then

        Timed_Transitions += "Exit ind " + 1 / rate for Exit
        Input_Arcs += "$Prd_i$ Exit 1"
        Output_Arcs += "do $Prd_i$ 1"
    Print Places, Timed_Transitions, Input_Arcs, Output_Arcs, Inhibitor_Arcs
    return
```

Paper 9

Software performance evaluation utilizing UML specification and SRN model and their formal representation

Razib Hayat Khan, Poul E. Heegaard

Submitted to a journal for reviewing

Part III

Thesis Appendix

Appendix A

List of related research approaches

Approach	System/ Service specification model	Specifying method for deployment mapping (Yes/No)	Formalization of UML model (Yes/No)	UML model validation (Yes/No)	Annotation of system/service specification model using MARTE/SPT (Yes/No)	Performance/ performability model generation (Yes/No)	Tool support (Yes/No)
Our Approach	Y	Y	Y	Y	Y	Y	Y
[5]	UML	N(d)*	N	N	SPT	Simulation based	UML- Ψ
[24]	UML	N	Y	N	Workload diagram	Queuing network	N
[25]	UML	N	N	N	N	GSPN	SPNP
[26]	UML	N	N	N	N	OPNM	N
[27]	UML	N	N	N	SPT	Markov	N
[28]	UML	N	N	N	N	Stochastic porcess algebra, Markov	N
[29]	UML	N	N	N	SPT	Simulation model	Simmcast
[30]	UML (r)**	N	N	N	N	Simulation	N
[31]	UML	N	N	N	SPT	Simulation based	N
[32]	UML (r)	N	N	N	XML based annotation	Queieng model	N
[33]	UML	N	Y	N	SPT	LGSPN	ArgoUML, GreatSPN
[34]	UML	N	Y	N	SPT	LGSPN	ArgoSPE
[35]	UML	N	N	N	QoS and Fault tolerance, SPT	Fault tree	ArgoUML
[36]	UML	N	N	N	SPT	SPN	ArgoUML, WebSPN
[37]	UML	N	N	N	SPT	LQN, PN	PUMA
[38]	UML	N	Y	N	SPT	LQN	ArgoUML, LQNS
[45]	UML	N	N	N	SPT	SPN	ArgoUML, WebSPN
[40]	UML	N	Y	N	SPT	LGSPN	GreatSPN
[41]	UML	N	N	N	UML 2 Profile	BPEL	N
[42]	UML	N	N	N	MARTE	CTMC	RSA
[43]	SysUML	N	N	N	N	DTMC	PRISM
[44]	UML	N	N	N	SPT	LQN	Tool developed by authors
[45]	UML	N	Y	N	N	LQN	N
[46]	UML	N	N	N	SPT	QN	UML- Ψ
[47]	UML	N	N	N	SPT	GSPN, CTMC	N
[48]	Fuzzy UML	N	Y	N	N	Fuzzy PN	N
[49]	UML	N	N	Y	N	LQN	RRE, LQSim

Approach	System/ Service specification model	Specifying method for deployment mapping (Yes/No)	Formalization of UML model (Yes/No)	UML model validation (Yes/No)	Annotation of system/service specification model using MARTE/SPT (Yes/No)	Performance/ performability model generation (Yes/No)	Tool support (Yes/No)
[50]	UML	N	N	N	SPT	LQN	N
[51]	UML	N	N	N	Defined by author	GSPN	PANDA
[52]	UML	N	N	N	MARTE	Simulation based	Papyrus
[53]	UML	N	N	N	MARTE	Simulation based	Tool developed by authors
[54]	UML (r)	N	N	N	N	Markov/PetriNet (performability)	NetSolve
[55]	UML (r)	N	Y	N	N	LQN	LQComposer
[56]	UML(r)	N	N	N	CoCoMe profile	Palladio performance model	Java2PCM
[57]	UML(r)	N	N	N	N	Transaction graph	N
[58]	UML	N	Y	N	SPT	LQN	Tool developed by authors
[59]	UML	N	N	N	MARTE	LQN,PN	N
[60]	UML(r)	N	N	N	N	Simulation based	PerfENplorer
[61]	UML(r)	N	N	N	N	Simulations based	SimuLink
[62]	UML(r)	N	N	N	N	Simulation based	FMP
[63]	UML	N(d)	N	N	SPT	Petri Nets	N
[64]	UML	N(d)	Y	N	Performance data card	EQN	UML-JMT
[65]	UML	N(d)	N	N	SPT	LQN	N
[66]	UML	N(d)	N	N	SPT	QN	N
[67]	UML	N(d)	N	N	SPT	LQN	N
[68]	UML	N(d)	N	Y(for XML)	XML based annotation	Simulation based	Tangram-II, Mosquito, Omondo
[69]	UML	N(d)	N	N	Defined by author	LQN	N
[70]	UML	N(d)	N	N	SPT	LQN	N
[71]	UML	N(d)	N	N	SPT	GSPN	PUMA
[72]	UML	N(d)	N	N	SPT	Simulation based	ArgoPerformance tool
[73]	UML	N(d)	N	N	Defined by author	Process algebra model	N
[74]	UML	N(d)	N	N	SPT	QN	N
[75]	UML	N(d)	N	N	UML profile for distributed system modeling	Simulation based	Rational Rose
[76]	UML	N	N	N	SPT	Network of timed automata	TIMES
[77]	UML	N	N	N	SPT	Simulation based	N
[78]	UML	N	N	N	SPT	Simulation based	LTSA
[79]	UML	N	N	N	SPT	LQN	PUMA
[80]	UML	N	N	N	SPT	GSPN	GreatSPN

Approach	System/ Service specification model	Specifying method for deployment mapping (Yes/No)	Formalization of UML model (Yes/No)	UML model validation (Yes/No)	Annotation of system/service specification model using MARTE/SPT (Yes/No)	Performance/ performability model generation (Yes/No)	Tool support (Yes/No)
[81]	UML	N	N	N	MARTE	QN	N
[82]	UML	N	N	N	SPT	Coloured Petri Nets	N
[83]	UML	N	N	N	SPT	LQN	ArgoUML, PROGRES
[84]	UML	N	N	N	SPT / MARTE	LQN, PN	PUMA
[85]	UML	N	N	N	UML-TUT profile	Simulation based	Telelogic TAU G2, Koski
[86]	UML	N	N	N	SPT	GSPN	ArgoSPE
[87]	UML	N	N	N	MARTE	Simulation based	Papyrus
[88]	UML	N	N	N	SPT	SPN	ArgoSPE
[89]	UML	N	N	N	MARTE	LQN	IMPACT, mediniQVT
[90]	UML	N	Y(ADL) Rapide	N	SPT	Simulation based	N
[91]	UML	N	Y	N	Defined by authors	Stochastic process algebra	ArgoUML, PEPA Workbench
[92]	UML	N	Y (object-Z)	N	Object-Z	Simulation based	RoZe
[93]	UML	N	Y(OCL)	N	N	N	N
[94]	UML	N	Y (POOSL)	Y	SHE method	Simulation based	SHESim
[95]	UML	N	Y (rt_EFSM)	Y	OCL	N	N
[96]	UML	N	Y (temporal logic)	N	N	N	Developed by authors
[97]	UML	N	Y	N	N	QN Model	N
[98]	UML	N	Y(ordering action)	N	N	PN	N
[99]	UML	N	Y (extended hierarchical automata)	N	N	N	N
[100]	UML	N	Y	Y	N	N	Vuml
[101]	UML	N	Y	N	N	N	N
[102]	UML	N	Y	N	N	N	N
[103]	UML	N	Y (STATEMATE statechart semantics)	partial	N	N	N
[104]	UML	N	Y(CSP)	N	N	N	N
[105]	UML	N	Y	N	N	Petri Nets	N
[106]	UML	N	Y(ASM)	N	N	N	N
[107]	UML	N	Y	N	N	N	N
[108]	UML	N	Y	Y	N	N	N
[109]	UML	N	Y(STATEMATE statechart semantics)	partial	N	N	N
[110]	[UML]	N	N	N	SPT	SPN	TimeNET

Approach	System/ Service specification model	Specifying method for deployment mapping (Yes/No)	Formalization of UML model (Yes/No)	UML model validation (Yes/No)	Annotation of system/service specification model using MARTE/SPT (Yes/No)	Performance/ performability model generation (Yes/No)	Tool support (Yes/No)
[111]	[UML]	N	N	N	N	SPN, Rare event simulation	N
[112]	UML	N	(MODEST)	N	Defined by authors	Simulation based	MOTOR
[113]	UML	N	N	N	Extension of UML defined by authors	GSMP	Rational Rose, DSPN eNpress 2000
[114]	UML	N	N	N	Defined by authors	Simulation based	N
[115]	UML	N	N	N	N	Simulation based	Tool developed by author
[116]	BPEL	N	N	N	N	CTMC (performability)	N
[117]	UML	N	N	N	Extension of UML	SAN (performability)	Mobius
[118]	UML	N	N	N	Worklaod diagram	Simulation based (performability)	ProSPEN
[119]	UML	N	N	N	SPT	Stochastic process algebra (performability)	RSA, PEPA
[120]	Defined by authors	N	N	N	Defined by authors	Simulation based (performability)	N
[121]	Defined by authors	N	N	N	N	Simulation based (performability)	SimPar

*r = reusable software components
**d = deployment diagram