HYBRID ANALYTICAL-SIMULATION MODEL FOR PERFORMANCE PREDICTION OF DISTRIBUTED SYSTEMS

Teză destinată obţinerii
titlului ştiinţific de doctor inginer
la
Universitatea "Politehnica" din Timişoara
în domeniul CALCULATOARE ŞI TEHNOLOGIA INFORMAŢIEI
de către

Ing. Cosmina CHIŞE

Conducător ştiinţific:	prof.univ.dr.ing Ioan JURCA.
Referenţi ştiinţifici:	prof.univ.dr. Dana PETCU.
	prof.univ.dr.ing. Ioan SALOMIE.
	prof.univ.dr.ing. Vladimir-Ioan CREŢU.

Ziua susţinerii tezei: 4 noiembrie 2011.

Seriile Teze de doctorat ale UPT sunt:

1. Automatică
2. Chimie
3. Energetică
4. Ingineria Chimică
5. Inginerie Civilă
6. Inginerie Electrică
7. Inginerie Electronică şi Telecomunicaţii
8. Inginerie Industrială
9. Inginerie Mecanică
10. Ştiinţa Calculatoarelor
11. Ştiinţa şi Ingineria Materialelor
12. Ingineria sistemelor
13. Inginerie energetică
14. Calculatoare şi tehnologia informaţiei

Universitatea „Politehnica” din Timişoara a iniţiat seriile de mai sus în scopul diseminării expertizei, cunoştinţelor şi rezultatelor cercetărilor întreprinse în cadrul şcolii doctorale a universităţii. Seriile conţin, potrivit H.B.Ex.S Nr. 14 / 14.07.2006, tezele de doctorat susţinute în universitate începând cu 1 octombrie 2006.

România, 300159 Timişoara, Bd. Republicii 9,
tel. 0256 403823, fax. 0256 403221
e-mail: editura@edipol.upt.ro

Cuvânt înainte

Teza de doctorat a fost elaborată pe parcursul activităţii mele în cadrul Departamentului de Calculatoare al Facultăţii de Automatică şi Calculatoare din cadrul Universităţii „Politehnica” din Timişoara, activitate desfăşurată concomitent cu profesia de inginer software în cadrul companiei Alcatel-Lucent Romania.

Mulţumiri deosebite se cuvin conducătorului de doctorat prof.dr.ing. Ioan Jurca, pentru îndrumare şi facilitarea accesului la materiale din domeniul ingineriei performanţei software. Dl. prof. Jurca a avut încredere în mine chiar şi atunci când am întâmpinat greutăţi în abordarea tematicii tezei. Datorită experienţei sale, a avut răbdare şi a reuşit să evidenţieze cu claritate valoarea activităţii de cercetare pe care am desfăşurat-o, activitate cu atât mai importantă cu cât este prima abordare a acestui domeniu de actualitate în cadrul departamentului. În plus faţă de noua abordare teoretică propusă, partea practică a presupus o documentare temeinică, abilităţi de implementare a unor algoritmi specifici şi a unui model de simulare, precum şi interfaţarea celor două metode.

Un rol important în formarea mea ca om l-au avut părinţii mei, Maria şi Pavel. Din fericire, am avut în tatăl meu un exemplu demn de urmat, prin activitatea de cercetare desfăşurată de-a lungul multor ani, într-un domeniu diferit, al ştiinţei materialelor, obţinând titlul de dr.ing. când eu eram studentă în primul an de facultate.

Aş dori să mulţumesc de asemenea prietenilor care mi-au rămas alături şi m-au încurajat chiar dacă preocupările de cercetare mi-au ocupat o mare parte din timpul liber. Totodată, adresez mulţumiri colegilor de la locul de muncă pentru apreciere, înţelegere şi susţinere. Colegii din cadrul Departamentului de Calculatoare şi specialiştii întâlniţi la simpozioanele şi conferinţele la care am avut onoarea să particip cu lucrări m-au ajutat sa-mi lărgesc perspectiva ştiinţifică prin discuţiile prolifice şi sugestiile constructive aduse.

Le sunt recunoscătoare tuturor cadrelor didactice care au contribuit la formarea şi dezvoltarea competenţelor mele intelectuale şi profesionale, începând din şcoala primară până la anii liceului şi facultăţii.

Timişoara, noiembrie 2011 Cosmina Chişe

Chişe, Cosmina

Hybrid analytical-simulation model for performance prediction of distributed systems

Teze de doctorat ale UPT, Seria 14, Nr. 3, Editura Politehnica, 2011, 130 pagini, 59 figuri, 12 tabele.

ISSN:2069-8216
ISSN-L:2069-8216
ISBN:978-606-554-373-7

Cuvinte cheie: Performance analysis, distributed systems, hybrid model, solver, simulation, LQN, MVA, UML MARTE, application

Rezumat
Distributed systems are at risk of being useless, since design flaws often lead to critical performance issues at deployment, when usually they are too costly to fix. Software Performance Engineering is concerned with modeling and prediction of performance parameters early in the software development process. This thesis proposes a new hybrid analytical-simulation approach for performance model solving. The UML model of the analyzed system is transformed into a performance model hierarchically decomposed into layers, in order to be able to address submodels. Simulation results for these submodels are later used in the higher level submodels by the analytical solver. The proposed approach, implemented by the author in a tool called PHYMSS (Performance Hybrid Model Solver and Simulator), is proven to be faster than simulation and more accurate than analytical calculus.

Contents

List of Figures

List of Tables

List of Abbreviations

A-MVA	Approximate-MVA (Bard-Schweitzer estimation)
AADL	Architecture Analysis Description Language
AATC	Airport Air-Traffic Control system
ADL	Architectural Description Language
AMG	Analysis Model Generator
AQN	Augmented Queueing Network
C2P	CSM to Performance model transformation
CAPPLES	CApacity Planning and Performance analysis method for the migration of LEgacy Systems
CHAM	CHemical Abstract Machine
CLISSPE	CLient/Server SPE
CQN	Closed Queueing Network
CSM	Core Scenario Model
CTMC	Continuous Time Markov Chain
DES	Discrete Event Simulation
DESMO-J	Discrete Event Simulation and Modeling in Java
DOM	Domain Object Model
DSA	Deterministic Service Approximation
E-PAD	Early Performance Aware Development
EG	Execution Graph
EIA/CDIF	Electronic Industries Association/CASE Data Interchange Format
EQN	Extended Queueing Network
EQNM	Extended Queueing Network Model
FAQ	Frequently Asked Questions
FP	Flight Plan

GQAM	Generic Quantitative Analysis Modeling
GreatSPN	GRaphical Editor and Analyzer for Timed and Stochastic Petri Nets
GRM	Generic Resource Modeling
GSMP	Generalized Semi-Markov Process
GSPN	Generalized Stochastic Petri Net
GUI	Graphical User Interface
IM	Intermediate Model
IPM	Intermediate Performance Model
JS	Java Script
KIB	Knowledge Interchange Broker
KLAPER	Kernel LAnguage for PErformance and Reliability analysis
LAN	Local Area Network
LGSPN	Labeled Generalized Stochastic Petri Net
LQML	LQN Modeling Language
LQN	Layered Queueing Network
LQNS	Layered Queueing Network Solver
LRT	Left and Right Truncated
LT	Left Truncated
LTS	Labeled Transition System
MARTE	Modeling and Analysis of Real Time and Embedded systems
MDA	Model Driven Architecture
MDD	Model Driven Development
MIL	Model Implementation Layer
MML	MetaModeling Layer
MOF	Meta Object Facility
MOSES	Modeling Software and platform architEcture in UML 2 for Simulation-based performance analysis
MPC	Model Predictive Control
MSC	Message Sequence Charts

MTL	(Meta Transformation Language
MVA	Mean Value Analysis
NFP	Non-Functional Property
NICE	Naval Integrated Communication Environment
NUMA	Non-Uniform Memory Access
OMG	Object Management Group
OQN	Open Queueing Network
OR/MS	Operations Research/Management Science
PAM	Performance Analysis Modeling
PCM	Palladio Component Model
PEPA	Performance Evaluation Process Algebra
PHYMSS	Performance Hybrid Model Solver and Simulator
PMIF	Performance Model Interchange Format
PUMA	Performance by Unified Model Analysis
QN	Queueing Network
QNM	Queueing Network Model
QoS&FT	Quality of Service and Fault Tolerance
QVT	Query/View/Transformation
RPC	Remote Procedure Calls
RSA	Rational Software Architect
RT	Right Truncated
S-PMIF	Software Performance Model Interchange Format
SCM	Supply Chain Management
SDL	Standard and Description Language (Telelogic Tau G2)
SimML	Simulation Modeling Language
SLA	Service Level Agreement
SMG	Simulation Model Generator
SPA	Stochastic Process Algebra

SPE	Software Performance Engineering
SPN	Stochastic Petri Net
SPT	Schedulability, Performance and Time
TL	Tool Layer
TVL	Tag Value Language
U2C	UML to CSM transformation
UCD	Use Case Diagram
UCM	Use Case Map
UCM2LQN	UCM to LQN transformation
UCMNav	UCM Navigator
UML	Unified Modeling Language
UML-PSI	UML Performance Simulator ($UML - \Psi$)
UML-RT	UML-Real-Time
UPE	Unified Performance Engineering
VSL	Value Specification Language
WPF	Windows Presentation Foundation
XMI	XML Metadata Interchange
XML	Extensible Markup Language
XSLT	Extensible Stylesheet Language Transformations

1. INTRODUCTION

1.1. Role of Performance Analysis in Distributed Systems Development

The fast expansion of computer networks during the last two decades led to unprecedented possibilities in the development of larger, more powerful applications that rely on resources belonging to several interconnected machines. Such distributed systems need to be designed for performance and verified before actual deployment. If not anticipated properly, the inherent communication delays for remote calls, the access policies for shared resources and the real traffic conditions (including high load periods) may lead to early decommissioning of the application. Experience has shown that most problems come from application design, but by the time the application is finished it is too late to fix them because the costs are too high.

Performance prediction based on Model-Driven Development (MDD) has an important role, helping developers make informed design decisions. In the last decade several research directions have been pursued, covering the entire performance prediction process: from input languages, and model transformation techniques, to model solvers or simulators and interpreters of performance results, providing feedback about the design and even suggestions of improvement. Current performance prediction tools rely on analytical or simulation models. Analytical models are mathematical representations of the system, solved in order to obtain performance parameters, such as response time, throughput and resource utilization. The same parameters can also be measured when using simulation models, executable systems, having a similar behavior to the original system. While analytical models can be solved fast, they usually need simplifying assumptions and thus cannot be applied to systems with complex behavior. On the other hand, simulation models can be derived from any kind of system, regardless of the complexity, the drawback being the large number of iterations that need to be performed to obtain sufficient measurements from which relevant mean values of parameters can be computed.

This thesis presents a hybrid approach: it combines analytical solving with simulation in order to benefit from the strengths of both prediction methods. The performance model is based on a simulation model that can be treated as a Layered Queueing Network (LQN) model [125] during the analysis process. The approach relies on the hierarchical decomposition used by LQNS, a popular LQN solver presented in [48], in order to be able to address system submodels. Simulation is inserted from a certain level downwards, and yields performance results for submodels or groups of submodels, results that can be later used in the higher level submodels by the analytical solver.

1.2. Performance Analysis Automation. Standardization of Distributed Systems Modeling Languages

Performance evaluation assumes obtaining performance parameters, such as response time, throughput and resource utilization, based on preliminary information about the system. Since most developers do not have performance analysis knowledge, performance analysis automation via tools was addressed, mostly after year 2000, when the most important analysis techniques had already been theoretically defined. These tools extract the performance model from the system description annotated with performance information and provide performance results. Complete information on the system is not available, since the development process is ongoing, but performance analysis does not need detailed models. Some information (like actual business specific parameters of client requests) would be meaningless from the performance point of view, the only important input parameters are the number of requests, the arrival rate and the performance parameters for steps needed to provide service, such as demand from the host resource, and the probability of execution.

Automation is an important step further in the direction of integrating performance analysis into the development lifecycle. The tools developed so far usually cover part of the process and are not too robust. They have been developed only as proof-of-concept tools in order to support various modeling and performance evaluation techniques and to verify their effectiveness. Another problem is that each tool usually relies on its own specific input formalism, which is an impediment in using these tools. Automation is meant to relieve software developers from learning performance specific methods and formalisms. The entire analysis process should be transparent to the developer, requiring a minimal understanding effort in providing the input parameters and interpreting the results.

Standardization of description languages for performance oriented system models is still ongoing. UML (Unified Modeling Language) [84] has been widely adopted as a universal modeling language, and extensions for specific purposes have been defined. Performance related annotations are part of two different UML profiles, SPT (Schedulability, Performance and Time) [91] and QoS&FT (Quality of Service and Fault Tolerance) [90], unified in the more comprehensive MARTE (Modeling and Analysis of Real Time and Embedded systems) profile [89]. AADL (Architecture Analysis Description Language) was adopted in 2004 for time critical systems definition and analysis, but it requires a detailed system design. Since 2007, MARTE is under development by OMG as a new UML profile. It is preferred because it does not require too many design details, so the analysis can be performed at a higher level, and thus earlier in the project lifecycle [85].

Performance Hybrid Model Solver and Simulator (PHYMSS) is the tool developed by the author of this thesis, implementing both the proposed hybrid method and a simulation approach. It is developed in C#, using Microsoft .NET Framework 3.5. Several input models consisting in deployment, use case, and activity diagrams have been built using the diagram editors Papyrus UML [5] with MARTE extensions for performance annotations. Since support for complete definition of sequence diagrams was not available in Papyrus UML, Rational Software Architect (RSA) v7.0 with a plug-in for MARTE [3] has been used for models based on deployment and sequence diagrams. These models are evaluated by the hybrid solver, the simulator and by pure analytical calculus. The results are compared from different perspectives: duration of processing and precision.

1.3. Thesis Objectives

The thesis intends to accomplish the following:

- From the methodological point of view, defining a new hybrid approach to performance modeling and analysis:
  - New layered hybrid analytical/simulation model to be easily transformed to a simulation model or a Queueing Network (QN) model.
  - Enhance and extend each component approach: the simulation model and the Mean Value Analysis (MVA) [104] analytical approach.
- As an application, building a new performance analysis tool, Performance Hybrid Model Solver and Simulator:
  - Parsing UML input models, with MARTE performance annotations (or SPT where needed).
  - Implementing both the proposed hybrid approach and a multi-threaded simulator.
- Validation of the proposed hybrid analysis method:
  - Finding and/or building diverse case studies, whose models are to be analyzed.
  - Comparison of the new method to pure analytical and simulation by running PHYMSS: the hybrid approach is expected to be faster than simulation and more precise than analytical calculus.

1.4. Organization

Chapter 2 reviews main research directions in Software Performance Engineering (SPE), from performance models to interpreting results, and classifies hybrid approaches.

Chapter 3 covers the multitude of input formalisms used in performance analysis to the present date and tools that rely on them.

Chapter 4 proposes innovative alternatives to building a hybrid model, and a novel solving method that processes and integrates data from two types of models: simulation and Queueing Networks.

Chapter 5 states the rules defined by the thesis author to extract the hybrid model from a UML input model, annotated using the MARTE profile. Several sets of diagrams are considered, since system behavior can be specified either by activity or by sequence diagrams.

Chapter 6 presents the tool which implements the new analysis technique, Performance Hybrid Model Solver and Simulator.

In order to compare performance of the method to real measurements and to pure simulation and show how the novel approach is applied step by step, chapter 7 includes case studies that cover several types of input models.

Chapter 8 concludes the thesis by summarizing the contributions and establishing future work directions.

2. SOFTWARE PERFORMANCE ENGINEERING APPROACHES

2.1. Software Performance Models and Solution Procedures

The first approach to integrating performance analysis into the software development process was described in [111], by Smith, who set the basis for SPE. System performance descriptions use two models: software execution model and system execution model.

The software execution model shows system behavior and uses Execution Graphs (EGs) to represent workload scenarios. EGs describe control flow among functional components. Static analysis is performed by solving such a software model and obtaining information on the response time and resource requirements of the software. The analysis has simplifying assumptions; it does not consider multiple users or resource contention. If the results are satisfactory, model refinement should be performed by building the system execution model, otherwise it is clear that the system is not feasible.

The system execution model describes system structure, represented as Queueuing Networks (QNs). This dynamic model takes into account the influence of multiple users, therefore the results are more accurate; resource requirements obtained by solving the software model are input parameters of the system model. Solving this system model leads to identification of bottleneck resources and helps in evaluating the effects of workload changes, software changes and hardware upgrades on performance.

Research until year 2000 focused on SPE methodology, the main elaborated algorithms being reviewed in [14]. Bernardo uses the TwoTowers tool which builds a Markov chain that is numerically solved. However, most approaches derive QN or EQN (Extended Queueing Network) models from various input system descriptions, detailed in chapter 3.

The most difficult part of the SPE process is considered by Cortellesa [37] to be the definition of a software model, because platform models can be extracted by other tools. The proposed performance model is based on QNs and can be solved by existing tools which implement numerical solutions. In order to evaluate the performance model, certain input parameters are obtained from the environment specification, others are present in the EG, such as classes of requested jobs, their service demands, routing probabilities among network centers. The steps to solve the model are presented below:

- Assign environment parameters to the Extended Queueing Network Model (EQNM).
- Apply reduction analysis techniques to the EG to obtain software parameters and assign them to the EQNM.
- Solve the EQNM with simulation-based and/or analytical methods.

Smith and Williams [115] present SPE best practices, concerning project management, modeling and measurement. The performance modeling recommendations are the following:

- Use performance models to evaluate architecture and design alternatives before committing to code.

- Start with the simplest model that identifies problems with the system architecture, design, or implementation plans; then add details as your knowledge of the software increases.
- Use best- and worst-case estimates of resource requirements to establish bounds on expected performance and manage uncertainty in estimates.
- Establish a configuration management plan for creating baseline performance models and keeping them synchronized with changes to the software.
- Use performance measurements to gather data for constructing SPE models and validating their results.

A comprehensive survey on performance models is [10]. The most important and widely used performance models are the following:

- Execution Graphs (EG) [36]
- Queueing Networks (QN) [36, 12], and their extensions:
  - Extended Queueing Networks (EQN) [37]
  - Layered Queueing Networks (LQN) [96, 101, 128]
- Stochastic Petri Nets (SPN) [63, 17, 71, 2]
- Stochastic Process Algebras (SPA)
  - Performance Evaluation Process Algebra (PEPA) [118].

2.1.1 Intermediate Performance Models

Since several different performance models can be derived from the same specifications, the idea of having a common representation occurred. Then, this common intermediate representation could be transformed into any kind of analytical model. A pivot language, also known as intermediate or bridge language, can be used as an intermediary for translation in cases where many source languages are translated to many target languages.

An important intermediate model is the Core Scenario Model (CSM) [97, 98], which extracts relevant performance information stored implicitly and explicitly in UML diagrams and SPT Profile data. The desired performance model is obtained by two transformations: UML model to CSM (U2C) and CSM to Performance model (C2P). The CSM meta-model provides explicit representation for entities that are required in order to build performance models, as shown in figure 2.1. Scenario flows are described as ordered sequences of steps and PathConnection objects which connect each pair of steps. Two consecutive steps are connected by a Sequence object, when there are several alternative steps (OR-fork, OR-join) Branch and Merge objects are used, while for parallel activities (AND-fork, AND-join) there are Fork and Join objects. Each PathConnection object has m source steps and n target steps, n and m depend on the particular object subtype. Messages will be used related to network communication, but they are not currently supported in UML SPT. Each step is executed by an active resource, which may be a device (ProcessingResource) or an operation provided by

an external subsystem (ExternalService). Steps can use passive resources, including processes or threads in an operating system, hosted by ProcessingResources.

Several step types are emphasized, such as Start, End, ResourceAcquire and ResourceRelease; the first step of a scenario (Start) may be associated to a workload. This kind of subtyping improves model checking and performance model generation.

The first phase of the model transformation, U2C, will be detailed in chapter 3. Regarding the second phase, transformation of CSM to a performance model, a systematic algorithm to convert a scenario model Use Case Map (UCM) (almost identical to CSM) into a LQN model is described in [94, 95]. This algorithm has been implemented in the UCM2LQN tool. Other examples of intermediate models are graph models (or kinds of EGs) in order to build queueing models.

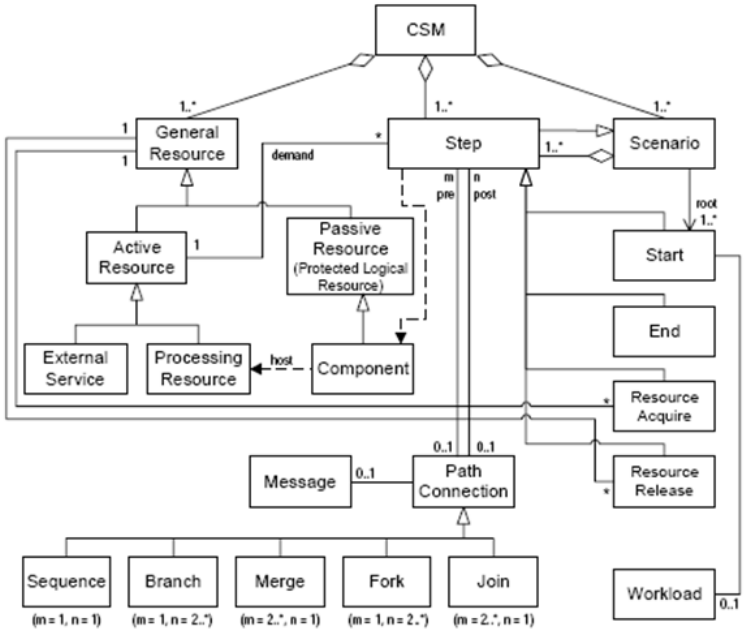

Figure 2.1: CSM meta-model class diagram [97]

Another approach in transforming annotated UML models into LQN models uses an intermediate model (IM) represented as an XML tree [56]. This model was developed in parallel with CSM, so it has similar features, but unlike CSM, it is a task-based model: steps are grouped into tasks, rather than into scenarios.

Since the intermediate model CSM can also be converted to other performance models, it is the center of a framework, called Performance by Unified Model Analysis (PUMA) [126]. Three types of performance models are addressed: Layered Queueing Networks (LQNs), Petri Nets, and Queueing Networks (QNs).

Deriving LQNs from CSMs is based on the algorithm for UCMs, described in [95], as mentioned earlier. The LQN model is represented in an XML syntax called LQML, for input to the LQN editor, solver, and simulator.

The translation to Generalized Stochastic Petri Nets (GSPNs) can be implemented using Labeled Generalized Stochastic Petri Nets (LGSPNs). Fragments of a CSM model have direct representations as fragments of LGSPN, with labels that direct the composition of the fragments into a full model. Using the compositional properties of LGSPN, the fragments are composed through several stages until the LGSPN representing the whole scenario is generated. For each class in the CSM meta-model there is a LGSPN pattern, parameterized by the attributes of the CSM class.

The translation into ordinary (not extended) Queueing Networks can be carried out by applying the workload reduction technique, first described by Smith in [111], treating the steps in the CSM model as steps in Execution Graphs. Automation is not discussed, only manual conversion is described in the paper. Ordinary QNs do not describe simultaneous resource possession, so this model ignores logical resources such as process threads and buffers and their Resource Acquire/Release Steps.

Two performance model formats have been defined in order to standardize notations regarding software models and system models: Performance Model Interchange Format (PMIF 2.0) and Software Performance Model Interchange Format (S-PMIF).

Performance Model Interchange Format (PMIF 2.0) [116, 112] is used for system performance models that represent computer platforms and network interconnections with a network of queues and servers.

Key requirements for an appropriate representation technique for PMIF:

- Expressive power - covering a wide range of models:
  - from a small number of servers to very large numbers of servers;
  - from one to many workloads;
  - both open and closed models;
  - solved using either analytic or simulation solution techniques.
- Extendibility - initially a format for a subset of QNMs (that may be solved using efficient, exact analytic techniques) will be defined and then extensions to cover additional facets of QNMs will be added.
- Compatibility with existing tools and theory.
- Visual QNM representation - in addition to the QNM details required to solve the model, a picture of the model could be exchanged among tools that support this.
- Model results - exchanging results derived from the modeling tool should also be possible.
- Ease of translation - it must be easy to generate the format for a model, and easy to translate the PMIF into an internal representation of a model.
- Tool support - the format should lend itself to a standard lexical analyzer and parser that could be used by all tools that wish to support the PMIF.

Several tools and QNM notations were considered in order to define a common language, and then the format to express the resulting meta-model was chosen to be EIA/CDIF (Electronic Industries Association/CASE Data Interchange Format). These standards define a transfer

format that allows tools with different internal databases and storage formats to exchange information. An exchange takes place via a file and internal tool information is translated to and from the file's transfer format. The QN meta-model, depicted in figure 2.2, is used to define the transfer format that enables the exchange of information specified in the meta-model between tools that support the format.

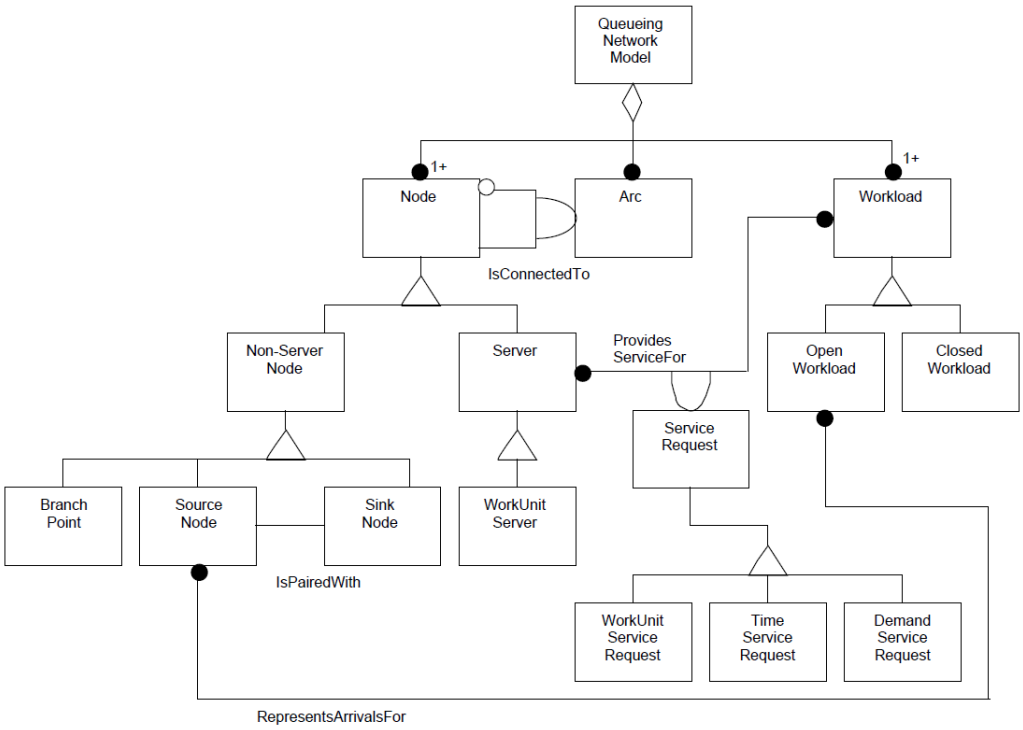

Figure 2.2: Performance Model Interchange Format notation: QN meta-model [116]

To export models with PMIF, tools provide all data they have that is specified in the meta-model. Tools must provide default values for the essential data in the PMIF meta-model if other values are not available. To import models in the PMIF format, tools use the data provided, discard data items they do not need, and make assumptions about data items they require that are not in the basic meta-model.

For software models a new notation is defined: Software Performance Model Interchange Format (S-PMIF), an XML notation based on an updated SPE meta-model [113]. The use of S-PMIF provides the following benefits for SPE tasks:

- Export of software system design to SPE tools where performance models can be constructed automatically.
- The model transformation can be used during system design evolution to check that the resulting processing details are those intended by the UML specification.

- Data available to developers is captured in the development tool, while other specific data can be added by performance specialists in the SPE tool.
- Rapid production of models makes data available for supporting design decisions in a timely fashion, thus allowing study of architecture and design tradeoffs before committing to code.
- Developers do not need detailed knowledge of performance modeling.

The paper also proves how the two notations, S-PMIF and PMIF, can be used together for model data interchange between different analysis tools; the process is illustrated in figure 2.3. Since the approach involves two phases, the UML model is converted to an S-PMIF model, by using a tool such as XPRIT [36], and then the S-PMIF model can be transformed into a PMIF model in order to perform system performance model analysis.

2.1.2 LQN Methodology and Model Solvers

Essentially, an LQN is made of client requests, tasks and hardware devices [102]. Each task runs on a processor, has entries that provide different classes of service to clients, and at the same time issues service requests to lower level tasks or hardware devices; an example is presented in figure 2.4.

Since LQNs are extensions of Queueing Networks on several layers, the following paragraph covers the basics of queueing theory, then an LQN solving method and optimizations are described; also, a language for defining LQN components is presented.

Queueing Theory Basics

A queueing system is usually made of one or more servers (processors) and their queues; for a queueing model to work, jobs are inserted into the system. Queueing Networks (QN) are obtained by interconnecting several queueing systems. [47] There are two types of QN models, depending on the workload (system customers) characteristics [68]:

- Open models: with an infinite stream of arriving customers (transaction workloads), described by their arrival rate λ.
- Closed models: where customers "re-circulate" (batch or terminal workloads) and are defined by the population N and think time Z.

Models consisting of both types of customer classes (open and closed) are referred to as mixed.

Fortier [47] explains that, because of the complexity of classical queueing system analysis methods, three computational alternatives have emerged:

- Central server model [23]: iterative solution proposed for closed QN based on finding the normalization constant for the solution of certain product form networks.
- Mean value analysis (MVA) [104]: applies to closed queuing networks that have a product form solution for the state probabilities.

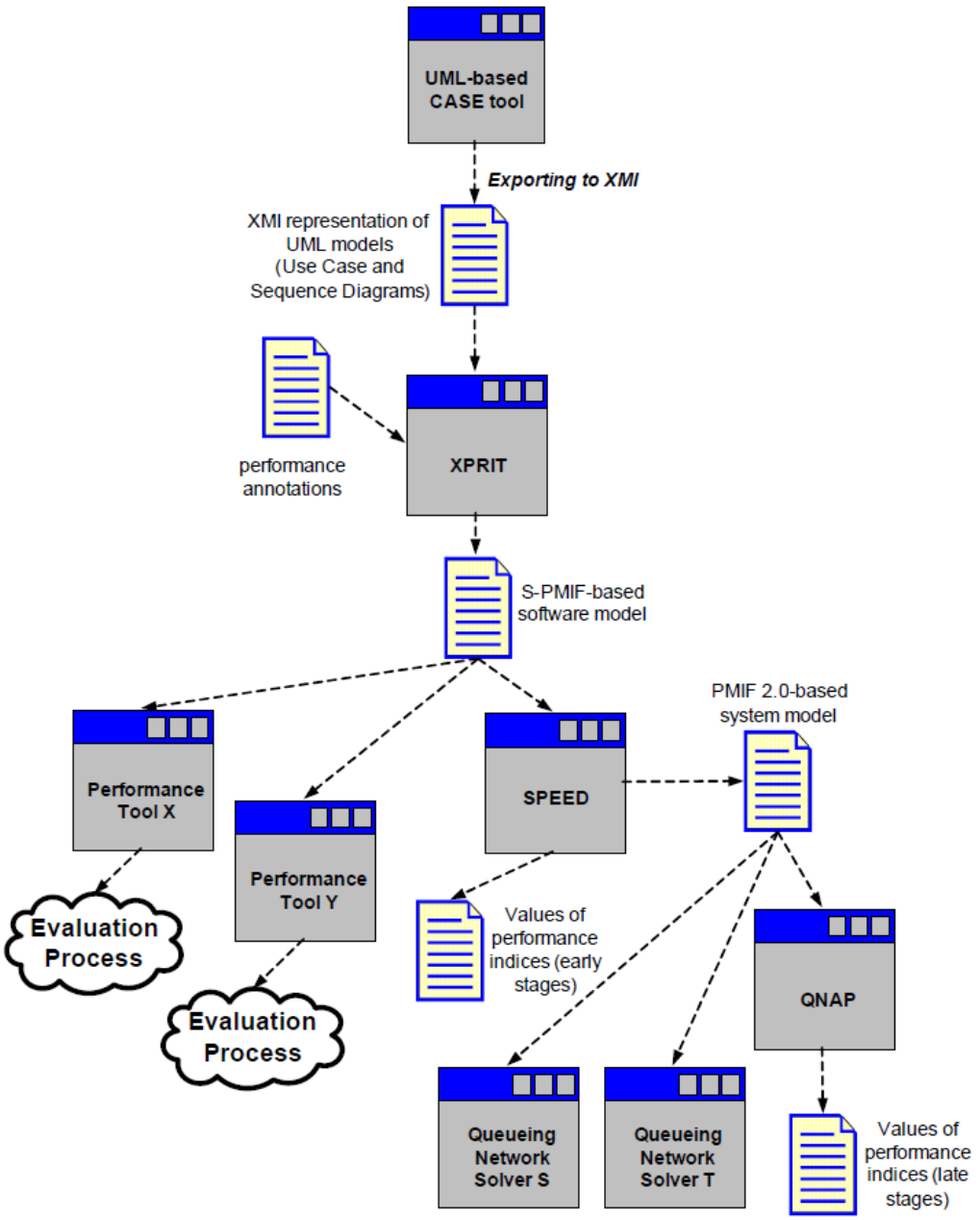

Figure 2.3: SPE interchange process: PMIF and S-PMIF [113]

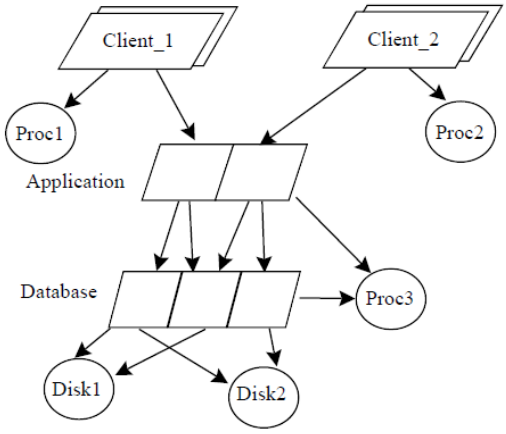

Figure 2.4: Layered Queueing Network example [102]

- Operational analysis [42]: based on observation (for a finite period) of basic, measurable quantities that can be combined into operational relationships; the technique can be used for open and closed networks, but the network under observation must be operationally connected (no server may be idle during the entire observation period).

The MVA solutions are based on the assumption that a customer, arriving at any queue in a closed system that is at steady state, experiences a wait in the queue that is equivalent to the steady-state wait time for that queue with the arriving customer removed. This assumption leads to an iterative algorithm where the steady-state performance characteristics for the system with $n+1$ customers are derived from the characteristics with n customers, which are derived from a system with $n-1$ customers, and so on down to one customer.

Since MVA is a powerful and elegant algorithm, it has been the focus of further research and several extensions have been developed [77]:

- Multi-class networks.
- Networks with load dependent servers.
- Networks with open and closed classes of customers.

LQN Solving

Franks defines in [48] a methodology to assess performance for client-server systems that cannot be solved by simple Mean Value Analysis [104]. Among the various types of LQNs, Closed Queueing Networks (CQN) are the ones considered for solving; there are several classes of requests that drive the behavior of the system, and each one is defined by two parameters: population or number of requests (N) and think time or external delay (Z). The population of requests is closed, as opposed to open requests characterized by arrival rate, that have no limit on the number of requests, but open requests never return after being serviced. In CQN, each client issues requests repeatedly; after a request is serviced and leaves the system, the client waits for Z time units and then sends a new request to the system.

A new solver, LQNS [49], is developed. It relies on model decomposition into submodels and also includes improvements to previous solving methods: forwarding is supported in server requests, by further delegating to lower level servers, leading to simultaneous resource possession; early replies can be modeled by using multi-phase tasks, so that each service can be divided into phases and the response is provided as soon as possible, while the execution continues on the server with a second phase, until the operation is completed. Second phases are usually performance optimizations, executed in parallel, for transaction cleanup or logging. Activities are defined as the unit of modeling and tasks support both homogeneous (multiservers) and heterogeneous (fork-join operations and asynchronous RPC) threads.

LQNS applies fixed-point iteration to submodels, obtaining delay and resource utilization values. The steps to solve a client-server Queueing Network model using LQNS are:

- Read the input and construct an object database: tasks, processors, entries and the calls between them.
- Perform processor, forwarding and think-time transformations.
- Generate the layer submodels, and build MVA submodels from the layer submodels.
- Solve the MVA submodels. The inputs to and the outputs from each submodel are extracted from or saved to the object database. This step is repeated until the waiting time results converge for each layer.
- Write the results out.

Several layering strategies are evaluated, in order to perform the topological sort of servers (establishing their nesting level): strict, loose, batched and squashed layering. Batched layering is considered the most reasonable one, after performing sets of tests, during which other methods fluctuated in performance (strict, loose), while squashed layering proved inferior by far, since it duplicates intermediate level servers.

Submodels are solved one at a time, propagating the results until they reach model level, and then the process is repeated, until results are convergent. There is a two-way dependency between submodels on consecutive layers.

Service times are propagated upwards - they are computed from waiting times of servers from lower level submodels:

$$s_{ml} = \Sigma_{i \in L, i \neq l} \Sigma_{j \in S_i} w_{mj} \tag{2.1}$$

where L is the set of all submodels, l denotes the current submodel, s_{ml} is the service time of task m of submodel l, S_i is the set of servers from submodel i, w_{mj} is the waiting time of server j in providing service to task m.

Think times of clients for each submodel are propagated downwards - they are computed from the parameters of the server in the immediate higher level:

$$Z_{i,l} = N_{i,l} \cdot \frac{1 - \rho_{i,l-1}}{\lambda_{i,l-1}} \tag{2.2}$$

where $Z_{i,l}$ is the waiting time of client i in submodel l, $N_{i,l}$ is the number of requests from client i in submodel l, $\rho_{i,l-1}$ is the utilization of task i when it is acting as a server in submodel $l-1$, $\lambda_{i,l-1}$ is the throughput of task i when it is acting as a server in submodel $l-1$.

In order to consider both dependencies, iteration at model level consists of pairs of passes, one in each direction: top-down and bottom-up. In the first step, submodels are solved starting from the highest level, in order to propagate think time values along request chains; and in the second step, the submodels are solved starting from the lowest level, to propagate upwards service time values for tasks.

An optimization regarding solving LQN including replicated subsystems is described in [83]. A specific notation is defined for replicated servers, as shown in figure 2.5. The proposed solution relies on the hierarchical approach in solving LQNs and defines an âĂIJinner loopâĂİ that handles the replicated subsystems. The replicas are solved once and the results are replaced in the higher level submodels in order to solve the whole system.

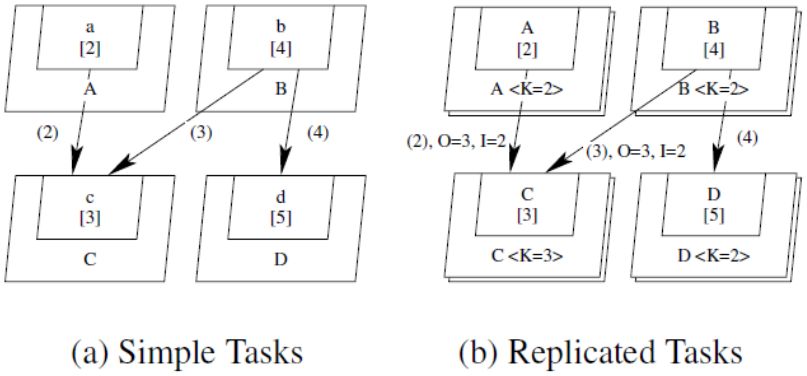

Figure 2.5: Notation for Layered Queueing Network replicated servers [83]

LQN Components

LQN input description language is extended in [76] to support declaration of component classes, also called reusable submodels. They are parameterized and can be instantiated inside LQN models, an example is shown in figure 2.6. Parameterization of instances leads to flexibility regarding multithreaded processing (number of supported threads) or other possible internal differences, while maintaining the common interface for the component class.

Component instances replace a given task, while component entries and requests, as well as processors are bound to existing items from the model. The component interfaces and their bindings are defined by a component assembly model [127], as illustrated in figure 2.7.

2.1.3 Simulation Models and Simulators

While most generic performance models presented earlier can be subject to both analytical solving and simulation, there are models specifically built for simulation. Simulation modeling assumes converting the system design into executable form and obtaining performance results by running the simulation. Simulation models are more accurate since they usually embed more details on the behavior and parameters of the original system. Performance analysis is also more precise than analytical solvers, because there are no restrictions on system parameters and no simplifying assumptions are required. The drawback is that the simulation may be

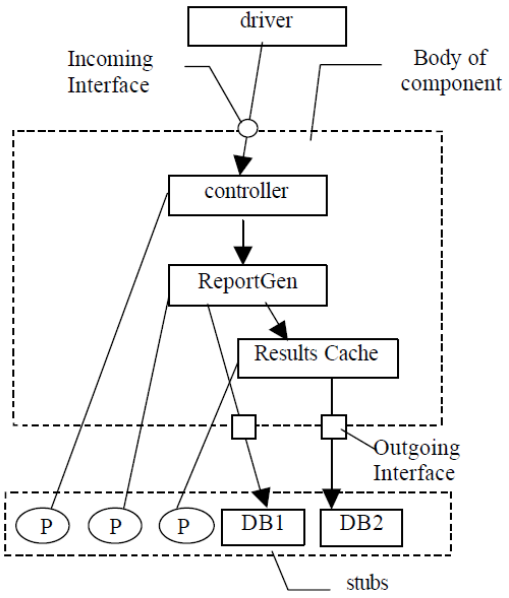

Figure 2.6: Layered Queueing Network component model: application server [127]

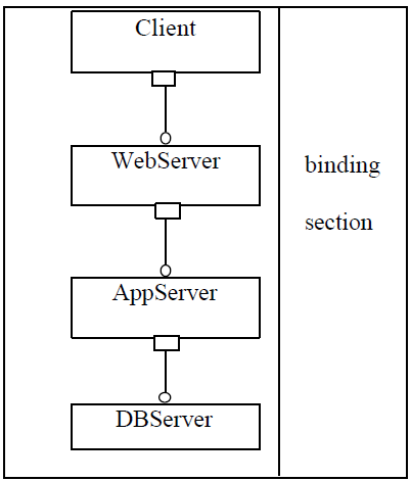

Figure 2.7: Layered Queueing Network component assembly model example [127]

time-consuming until the results converge and the modeler may create a rather complex simulation model, since there are no restrictions in building it, other than to behave similar to the system being analyzed.

Most simulators are based on Discrete Event Simulation (DES) [15], meaning that the system state changes are caused by events that occur at discrete moments in time. There are two approaches: event-oriented and process-oriented. An event-oriented simulation follows a sequence of ordered events, executing the corresponding handlers. In process-oriented simulation, the system is represented by a set of interacting processes. Most visual simulation systems provide a process-oriented view.

In order to assist building simulation programs out of system design, a generic framework has been developed, called SimML (Simulation Modeling Language) [8, 9]. It has been implemented as a Java package, so that simulation programs would be written in Java. They are executed in the JavaSim simulation environment. SimML contains generic simulation components that can be used to generate process-based event-oriented simulation programs. A tool is necessary in order to provide the user with the means to enter the system design as UML diagrams (class and sequence) and also define simulation parameters. An internal simulation model is created by the tool, each component exposing a write method which generates the corresponding part of the JavaSim program. UML and SimML data persistency is obtained via XML files.

More realistic results can be obtained if the communication resources are also simulated. In [58], Hennig uses the OMNET++ discrete-event simulator, along with pre-modeled modules that simulate JBoss and even the TCP protocol. He makes use of Use Case, Sequence and Deployment diagrams in order to provide network topology and system behavior information to the simulator.

Another popular network simulator is OPNET, used by De Miguel et al. in [81] in order to generate simulation models from extended UML diagrams (to express temporal requirements and resource usage, the target being hard real-time systems). UML extensions are suggested in order to address real-time systems modeling aspects, such as load timing distribution, resource usage, time constraints, and scheduling. These extensions are organized as constraints, stereotypes and tagged values, as a response to the UML SPT Profile RFP (Request for Proposal). Two tools are mentioned in the paper: AMG (Analysis Model Generator) and SMG (Simulation Model Generator). SMG uses the dynamic library of OPNET Ema (External Model Access) and generates OPNET models by reusing generic models that represent different metaclasses of UML (operation, classifier, node, etc.). These generic models are customized with application specific information; the application simulation model is obtained by combining submodels generated for each application element. This model is the input for OPNET which evaluates system performance; the approach provides feedback, performance results being inserted as tagged values.

Hennig has also studied alternative system performance evaluation techniques in [57]: benchmarking, simulation, prototyping and load testing.

Benchmarking means measuring how many small standardized activities a given system can execute per second. It is useful for evaluating hardware performance, but it is not relevant for software systems.

Simulation implies building an abstract model of the infrastructure (hardware and middleware), the behavior of the business logic as well as the expected load from internal sources and users.

Prototyping consists of developing small modules of the system whose functionality and interoperability are tested; they can also be subject to load-testing. Because manually developing prototypes is time-consuming and results are restricted, this approach is rarely used.

Load-testing simulates expected user behavior and can be used for acceptance testing or during development to test performance aspects of individual modules or prototypes. Load-testing a finished system is the least predictive method: it is predictive only in terms of the anticipated user behavior, which might vary greatly from the behavior of real users.

The author suggests use of automated performance prototyping, using UML diagrams as input, by following the process depicted in figure 2.8.

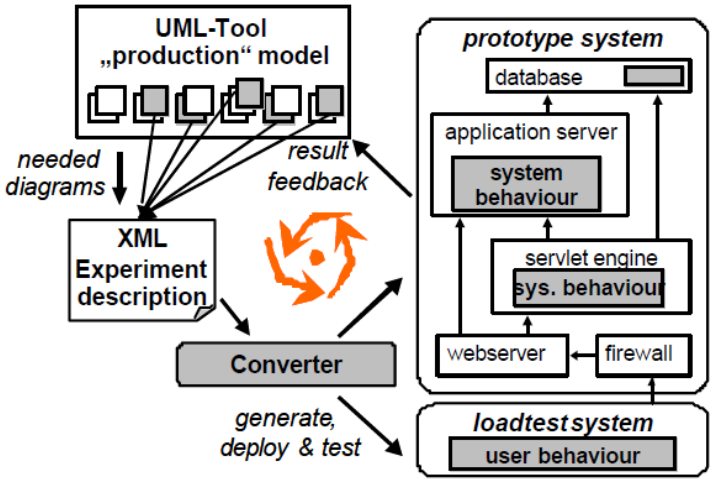

Figure 2.8: Performance prototyping process [57]

As it can be seen, the required information is extracted from the UML-Tool into an XML experiment description, which in turn is converted into prototype parts. For user behavior simulation, a commercial load-testing tool is used. The prototypes are implemented in JSP (JavaServer Pages) and uploaded into JSP-directories of the respective servlet-engines.

An important simulation tool for software systems, developed by Marzolla as the application part of the PhD thesis [73], is $UML-\Psi$ (UML Performance SImulator), described in detail in [74]. The functionality provided by the tool is illustrated through case studies, such as NICE (Naval Integrated Communication Environment) [73] and an e-commerce application [74]. UML Performance Simulator is a tool that builds a simulation model of the system being analyzed and runs the model in order to collect performance information. The overall structure of the tool is presented in figure 2.9.

The tool relies on a C++ library, *libcppsim*, which provides basic functionalities for process-oriented simulation modeling and simulation output data analysis [72].

Tool input consists in XMI (XML Metadata Interchange) files exported by ArgoUML [1] (or the commercial version Poseidon), a visual tool for defining UML diagrams. The diagrams are annotated with performance information according to UML SPT Profile [91]. $UML-\Psi$ extracts relevant information from the XMI input file (Use Case, Deployment and

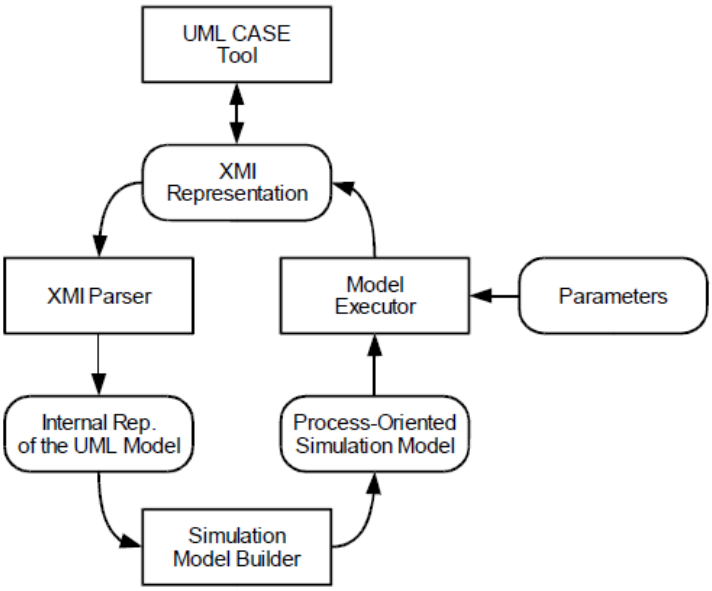

Figure 2.9: UML-PSI overview [73]

Activity diagrams) and generates a performance process-oriented simulation model of the system. The simulation model is based on three main types of entities, corresponding to the actions of activity diagrams, the resources of the software system and the workloads.

The simulation model is executed by using information from a configuration file [75]. The configuration file can be an arbitrary fragment of Perl code, which usually defines simulation parameters, such as simulation duration and desired accuracy of results, and also provides values for unbounded variables in the UML model. The file name is specified as the value of the *paramFileName* tag associated to the whole UML model. After the code in this file has been parsed, the Perl interpreter environment (modified by every declaration contained in the configuration file) is used to parse the tagged values, attached to elements of the UML model. Hence, the configuration file may be used to define Perl variables which are used inside tagged values. The user may then explore different performance behaviors for different values of these variables by simply changing the configuration file, without affecting the UML model.

Performance results are inserted into the software model as tagged values for relevant UML elements. The feedback mechanism is immediate since there is a clear correspondence between the software model and the simulation model, as shown in figure 2.10.

During the life cycle of a software system it is customary to carry out capacity planning and performance analysis, not only to evaluate the system's environment but also to plan its operation. CAPPLES, a CApacity Planning and Performance analysis method for the migration of LEgacy Systems [110], has been developed to address the particularities of performance evaluation during migration of a software system. Simulation models are used, since analytical models usually require simplifications that may not be adopted for mission critical systems.

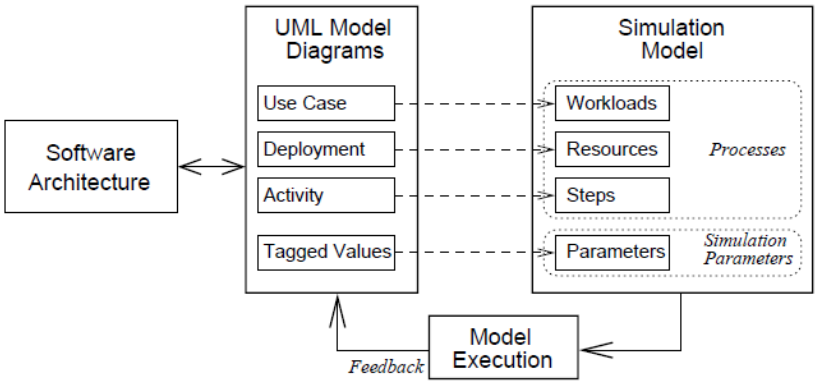

Figure 2.10: UML-PSI simulation model [13]

The simulation process for performance evaluation of a non-operational target system subject to migration is shown in the activity diagram in figure 2.11. The focus in the process is on obtaining a valid model. After the model is built (Modelling), the model is executed in the Simulation activity, producing simulation results. The Validation activity is based on a comparison of the simulation results with the results produced in the Experimentation activity. If the simulation and experimental results are similar enough, the model can be classified as valid. Otherwise, the Modelling refinement activity may be required to produce a more accurate simulation model. Once the model is validated, the Prediction activity is executed producing the simulation results that describe the predicted behavior of the target system.

2.1.4 Methodologies for Component-Based Systems

Special-purpose prediction techniques and tools have been defined for component-based systems in the last few years, they use specific input model notations and their main goal is to facilitate selection and reuse of components, by evaluating different choices regarding component assembly into larger systems.

Performance models for component-based systems were defined as early as 10 years ago. Kahkipuro [62] proposes a framework on UML notation (from which a selective Performance Modeling Language is defined) for describing such performance models based on Augmented Queueing Networks (AQN). Gomaa and Menascé [51] investigate how UML diagrams can be used to define component interconnection patterns, and the diagrams annotated using an XML-type notation are transformed into QN performance models in order to be analyzed.

A reference paper in the field of component-based SPE is [22]: it presents an SPE methodology and a tool that implements it. The proposed technique, also described in [20, 21], shows how to apply SPE analysis to components in the process of component selection and reuse: parameterized evaluation is applied to components and then the global results are obtained by assembling the components step-by-step. The tool receives input from ArgoUML [1] and the system is modeled as EG and QN, according to the SPE methodology [111];

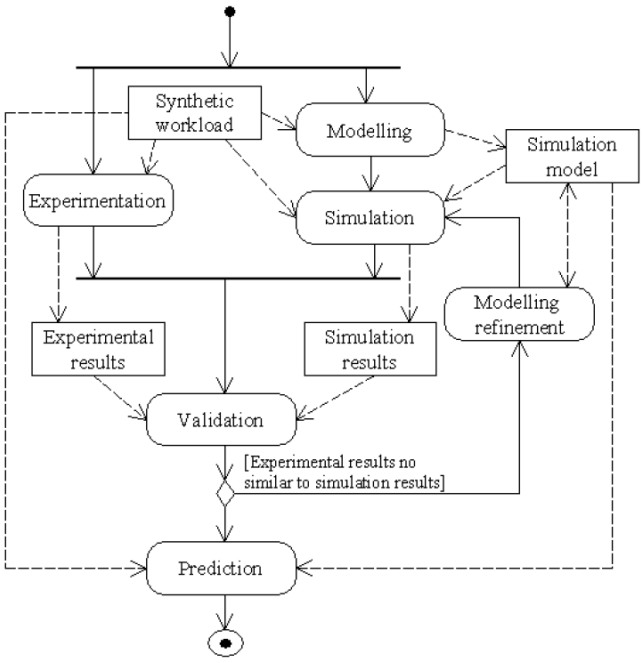

Figure 2.11: The simulation process for CAPPLES [110]

solvers are integrated for both models. The component assembly is expressed using a simple language that allows compositional performance analysis, language described in [52].

KLAPER (Kernel LAnguage for PErformance and Reliability analysis) [54, 53] is an intermediate language for model-driven performance and reliability analysis of component-based systems. The language is used to derive an intermediate model from the UML input model (that uses SPT) and the intermediate KLAPER model is then transformed into an LQN model. The environment will be extended to support many-to-many mappings between input models and output performance models (QN, Markov models). It does not address model solving, it just transforms models.

The Palladio Component Model (PCM) [16, 67] is a domain specific modeling language for component-based software architectures. PCM does not use annotated UML as design model, but defines its own meta-model. This reduces the model to notions necessary for performance prediction and does not introduce the high complexity of arbitrary UML models with a variety of concepts and views. A simulation tool has been implemented based on PCM and used to prove the efficiency of the method.

2.2. Interpretation of Performance Results

Integrating performance analysis within the software lifecycle also assumes interpreting feedback from SPE regarding system architecture or design. A framework that auto-

matically interprets performance output parameters and suggests improvements by identifying performance anti-patterns is described in [34]. The high-level flow chart of the proposed process is shown in figure 2.12.

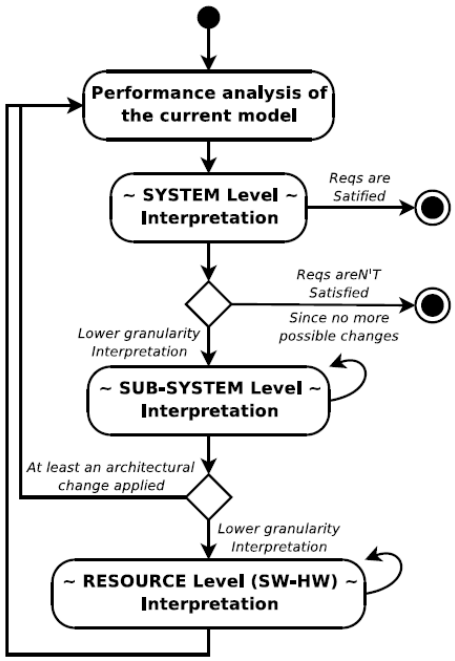

Figure 2.12: Performance results interpretation automated process [34]

The approach goes through two fundamental phases:

- identification phase (or interpretation phase), where the analysis of the performance results helps identify particular scenarios that affect performance;
- construction phase (or generation phase), where several architectural alternatives are constructed, based on the information collected in the previous phase.

Three granularity levels were identified at which software architecture can be analyzed:

- System level - only global indices can be obtained: end-to-end response time (i.e. from input to output), system throughput.
- Subsystem level - an intermediate abstraction level where the system's components and their interactions can be analyzed (the system can be split by applying several criteria).
- Resource level - the finest granularity level for conducting a performance analysis; indices of software or hardware components (that cannot be further split) are obtained at this level.

The approach is exemplified on a LQN model in [35], but it is generic enough to be applied to any kind of performance model.

2.3. Hybrid Approaches

Performance has mostly been predicted by pure analytical or simulation approaches; some tools implement both techniques, so the appropriate one can be chosen, depending on project constraints.

The first paper that defines and classifies hybrid approaches is [108]. According to this paper, there are two distinct approaches:

- Hybrid simulation/analytic models - combining simulation and analytical models of parts of the system and even the solution procedures.
- Hybrid simulation/analytic modeling - development of independent simulation and analytical models of the whole system, and solving of the problem by using the solution procedures together.

Hybrid models are divided into four classes, as shown in figure 2.13:

- CLASS I. A model whose behavior over time is obtained by alternating between using independent simulation and analytic models. The simulation (analytic) part of the model is carried out without intermediate use of the analytic (simulation) part.
- CLASS II. A model in which a simulation model and an analytic model operate in parallel over time with interactions through their solution procedure.
- CLASS III. A model in which a simulation model is used in a subordinate way for an analytic model of the total system.
- CLASS IV. A model in which a simulation model is used as an overall model of the total system, and it requires values from the solution procedure of an analytic model representing a portion of the system for some or all of its input parameters.

Four potential usages of hybrid modeling (figure 2.14) were given in [108]:

- Developing Operations Research/Management Science (OR/MS) theory.
- Gaining insight into system behavior.
- Validating analytic models.
- Performing optimization.

While the previously presented paper emphasizes hybrid models, Ignall and Kolesar [59] encourage use of hybrid modeling.

A more recent paper was published by Sargent in 1994 [105], to conclude on the effects of [108] on models and modeling research: before 1978 and after 1984 the usage of hybrid models and modeling was quite limited. The systematic approaches and classifications of 1978-1984 have not been incorporated into the field and research was still needed.

Hybrid models have usually been implemented for analyzing the behavior of heterogeneous systems or systems that have clearly defined modules, for which one method or the other was considered more appropriate.

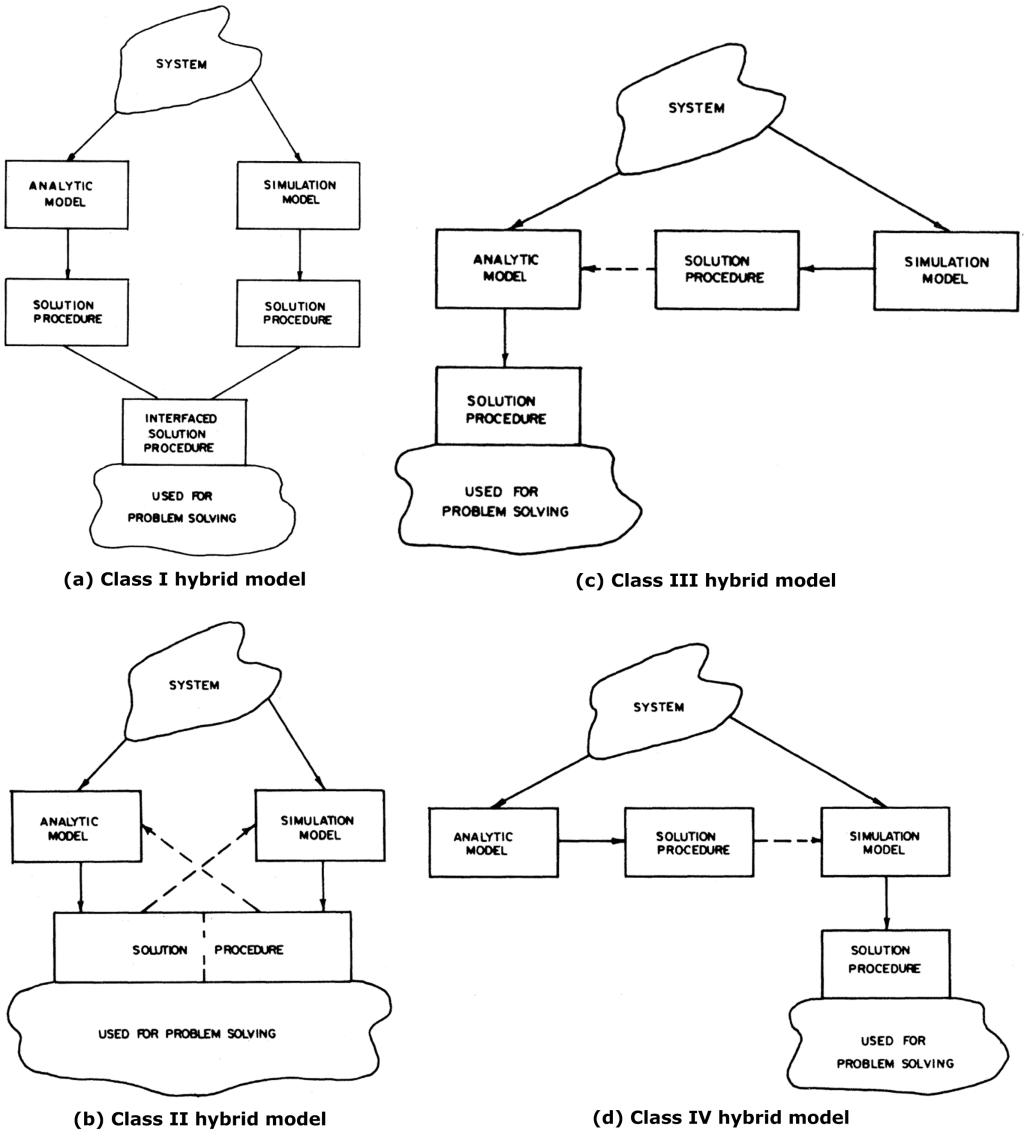

Figure 2.13: Classification of hybrid models [108]

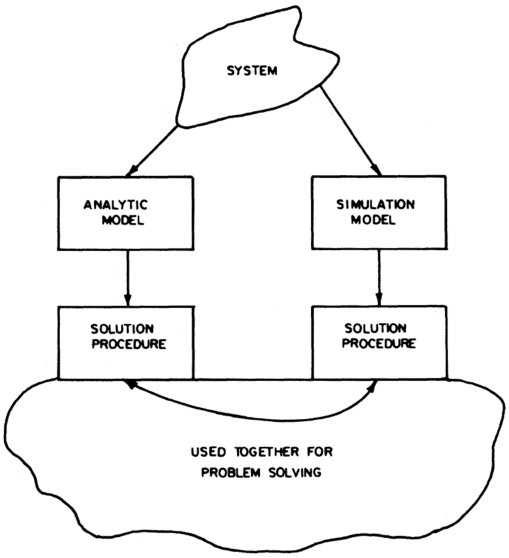

Figure 2.14: Hybrid modeling [108]

Nico van Dijk describes several examples of systems which benefit from hybrid modeling [45]: queueing theory is used to provide rules and directions, narrowing the number of design alternatives, while simulation is run to compare these options and chose the optimal one. A more comprehensive paper on optimization by hybrid modeling is [44], in which simulation is combined with queueing, linear programming, dynamic programming, or heuristic dynamic scheduling, in different applications, such as call centers, airport check-in, or train scheduling.

Hybrid simulation-analytic modeling was used in production-distribution planning [69], as part of Supply Chain Management (SCM). The approach involves obtaining production and distribution rates from the analytical model, and using them as input for the independently developed simulation model; this iterative process is repeated until simulation results show that the rates are feasible in realistic operational conditions. Another hybrid modeling solution for SCM is presented in [107], by combining Model Predictive Control (MPC) with DES. In MPC, the current and historical measurements of a process are used to predict future behavior, by using a control-relevant objective function. A Knowledge Interchange Broker (KIB) is used for model composition, since each of the approaches (DEVSJAVA simulation and MPC MATLAB solver) has its own syntax and semantics. The simulator sends its current output (inventory level) to the MPC solver that estimates future inventory levels, compares them to forecasted demands, sends computed starts of each factory to the simulator, and then the process is repeated.

NUMA (Non-Uniform Memory Access) architectures benefit from hybrid modeling too [122]; Westall and Geist present a system for describing (input modeling) and solving closed queuing network models of such systems. The input model is translated in order to be processed by either a discrete event simulator, or an algorithm derived from value analysis (MVA). The analytical algorithm combines MVA with DSA (Deterministic Service Approxi-

mation), in order to correctly consider the deterministic service times of devices (memories, buses) belonging to NUMA architectures. The fast MVA-DSA solver is used to interpolate between design points computed by simulation.

A few examples of recently applied hybrid modeling approaches are presented in the following paragraph; the methods are separately applied, in an iterative manner, have distinct input and output parameters and they are ordered one after the other. The motivation for these approaches is that analytical models are not able to cover certain lower level parameters that can be measured by simulation. A hybrid approach combines software design evaluation with network specific architecture in [119]. The software model (LQN) and network model (NS-2 [4]) are solved iteratively, thus the analysis results are refined. Another hybrid method is an iterative algorithm that applies alternatively analytical techniques and simulation for network processor design [24].

This thesis presents a hybrid model of class III, according to the classification in [108]: the analytical model uses results from a simulated submodel in the solving algorithm. This approach is different compared to the previously mentioned hybrid modeling approaches, because only part of the system model is simulated. In hybrid modeling, a model of the entire system is simulated, and an independent analytical model is used for optimization or for rapid narrowing of design solution space that is simulated afterwards (analytical results are validated by means of simulation). The main purpose of the proposed approach is similar to optimization approaches: improvement of prediction speed, by limiting the size of the simulated submodel (while in optimization, the number of model design alternatives is limited) and improvement of accuracy, compared to pure analytical models.

2.4. Summary

SPE is focused on integrating performance analysis in the software development lifecycle as early as possible. The main research directions in SPE are reviewed.

First, the most important approaches in modeling are presented, from performance models, such as EG, QN, SPN, to intermediate performance models that allow transformation of different input formalisms into several performance models (CSM) and interchange formats (PMIF, S-PMIF) to facilitate transfer of performance data between analysis tools.

Two kinds of performance analysis methods are covered: analytical and simulation.

Analytical solving techniques for performance models are presented in the context of queueing theory. LQN related methodologies are detailed since these will be the basis for the analytical component of the proposed hybrid approach.

Simulators and simulation frameworks (SimML) are also reviewed, focusing on DES, without omitting alternative dynamic evaluation techniques: benchmarking, prototyping and load testing. $UML - \Psi$ and CAPPLES are two examples of simulators: the former supports UML SPT diagrams as input and has a scenario based internal model, while the latter is applied for optimizing legacy systems.

Component-based systems are modeled using specific formalisms, since they are mostly concerned with selection and reuse of components and their assembly. A few such languages are presented: KLAPER, PCM.

Automation of interpreting performance analysis results is a more recently pursued direction. It provides design alternatives based on anti-pattern recognition techniques.

Hybrid approaches are classified and existing hybrid modeling directions are briefly presented, since hybrid models haven't been incorporated in the field yet. The hybrid meta-model proposed by the thesis author in chapter 4 belongs to class III according to the classification.

3. STANDARDIZATION OF DISTRIBUTED SYSTEMS MODELING LANGUAGES

3.1. Transformation Methodology: Input Models to Performance Models

Several system modeling languages were used to provide a high-level description of systems subject to performance analysis; the formalisms and the transformation procedures until year 2000 are briefly presented in this section.

Bernardo et al. [19] use an architectural description language (ADL) based on SPAs and the TwoTowers tool [18] that supports such input.

However, most approaches consist in deriving QN (or EQN) models from different kinds of system descriptions, which will be mentioned in the following paragraphs.

Gomaa and Menascé created CLISSPE (CLient/Server SPE) program specifications and a compiler that generates a corresponding QN model [50].

Balsamo et al. [11] use the CHAM (CHemical Abstract Machine) formalism [60]; evaluation relies on the analysis of the Labeled Transition System (LTS) representing the dynamic behavior of the CHAM architecture.

Andolfi et al. [6] start from Message Sequence Charts (MSC); the dynamic aspect of the behavior is emphasized by considering the real degree of parallelism (through description of sequences of events).

Aquilani et al. [7] use Labeled Transition Systems (LTS), a finite state representation of system architecture, independent of any ADL; concurrent execution and component interaction are modeled by EQN models.

Williams and Smith [123] obtain the software execution model (QN) from MSC or sequence diagrams; class and deployment diagrams are used to complete the system specification, but are not involved in the conversion. As already stated in section 2.1, EGs describe the software execution model, and resource requirements lead to the system execution model (QN). A tool that accepts such input formalisms has been developed, *SPE•ED*, which also allows simulation of the QN model. After year 2000, tool development started to make significant progress.

A more formal methodology to automatically extract a performance model from UML diagrams, based on the previously mentioned approach of Williams and Smith, is defined in [37]. The system description consists of use case diagrams, sequence diagrams and deployment diagrams, which define the Extended Queueing Network Model (EQNM) and the relationships between software and hardware. The basic steps of building the performance model, as presented in [37], are the following:

- Extract user profile from Use Case Diagram (UCD).
- For each use case in the UCD process the set of Sequence Diagrams to obtain the meta-EG.

- Use the deployment diagram to obtain the EQNM of the hardware platform and to tailor the meta-EG in order to derive an EG instance.
- Combine the EG and EQNM into the system performance model.

More details on the most important approaches are presented by Balsamo and Simeoni [14], where the evolution of the SPE methodology until year 2000 is reviewed.

3.2. Automation of Conversion and Performance Analysis. Input Formalisms for Tools.

The input to performance evaluation can have different degrees of abstraction. A high level description is used in [94, 96], starting from Use Case Maps (UCM) and deriving an LQN model, which is then analyzed by automated solvers. The proposed process is called Early Performance Aware Development (E-PAD) and uses generative programming principles in order to evaluate performance early during software development. The process is illustrated in figure 3.1.

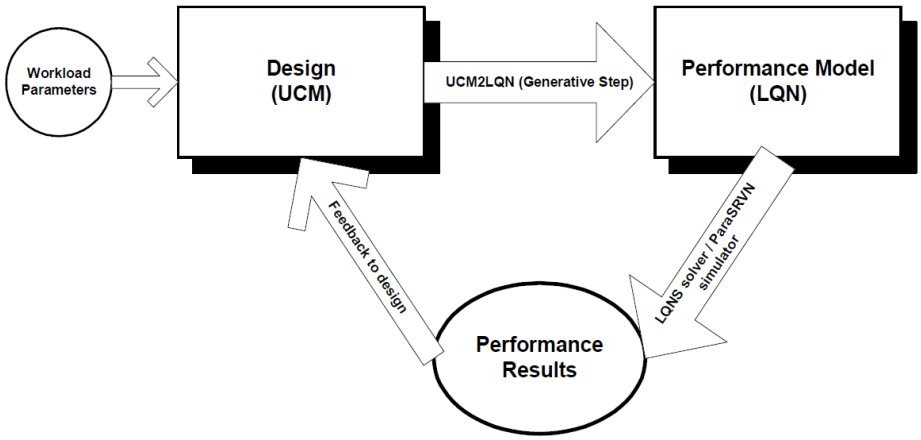

Figure 3.1: Early Performance Aware Development (E-PAD) process [96]

Tools have been developed in order to automate the process. UCM Navigator (UCM-Nav) is a UCM visual editor enhanced by the possibility to define additional information. UCM2LQN converter is a generative tool, it is integrated with UCMNav, takes the internal representation of the specification from UCMNav and converts UCM paths into sequences of LQN elements. The generated LQN models are saved as text files and will be solved by LQNS or ParaSRVN.

More details in system specification lead to more accurate results, so UML diagrams are preferred in order to provide input to performance analysis tools. Most approaches rely on the UML Profile for Schedulability, Performance and Time (SPT) [91], in order to annotate the diagrams in a consistent manner. However, UML diagrams were used for performance analysis even before the SPT Profile existed.

A first approach to the integration of non-functional properties into the software model, which previously included only the functional attributes, describing behavior, is presented in [33]. The proposed framework combines information from system descriptions expressed in XML format using different models, depending on the type of system view, and may provide output for various analyzer tools, which perform validity checks and/or compute output parameters (metrics), as shown in figure 3.2. The various input descriptions are filtered and combined into a common representation, called XML Models Representation, by using a schema for each input description type. The Semantic Relations section translates analysis results (feedback) from one notation to another; it is also based on XML files, each one containing translation rules for a pair of notations corresponding to different analyzers.

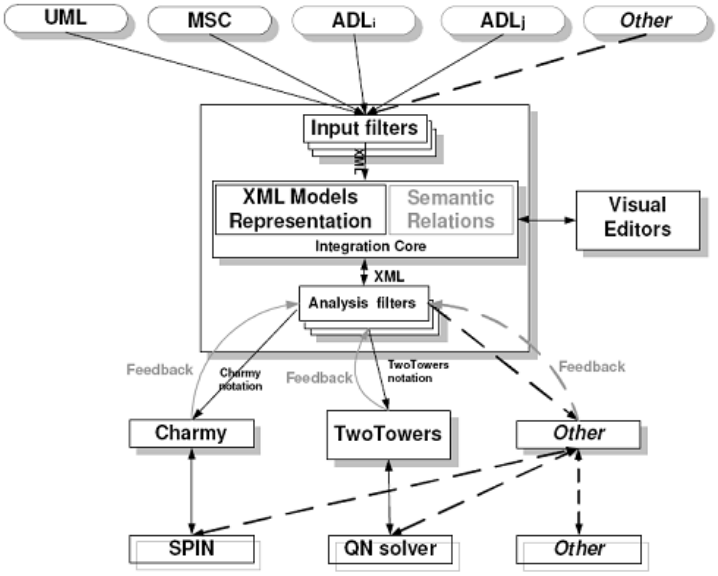

Figure 3.2: NFP integration framework architecture [33]

Starting from the SPE (Software Performance Engineering) approach, defined by C.U. Smith [114], which divides performance analysis in two phases – software execution model analysis and system execution model analysis – a tool called XPRIT (XML-based PRIMA-UML Tool) [36] was built. The tool represents an implementation of the PRIMA-UML methodology [38]. It provides inputs for both phases by parsing annotated UML models. It may generate either EG (as software execution model), or QN (as system execution model), as illustrated in figure 3.3.

Performance models were extracted from UML diagrams even before the SPT Profile was adopted; conversion algorithms and their automation were a main concern. King and Pooley [63] define an intuitive method to obtain a Generalized Stochastic Petri Net (GSPN) from UML collaboration diagrams with embedded statecharts. Bernardi et al. [17] present an algorithm for derivation of Stochastic Petri Nets (SPN) from UML sequence and state diagrams; the statecharts are used to describe the behavior of each participant to an interaction modeled by a sequence. Lindemann et al. [70] derive a Generalized Semi-Markov Process

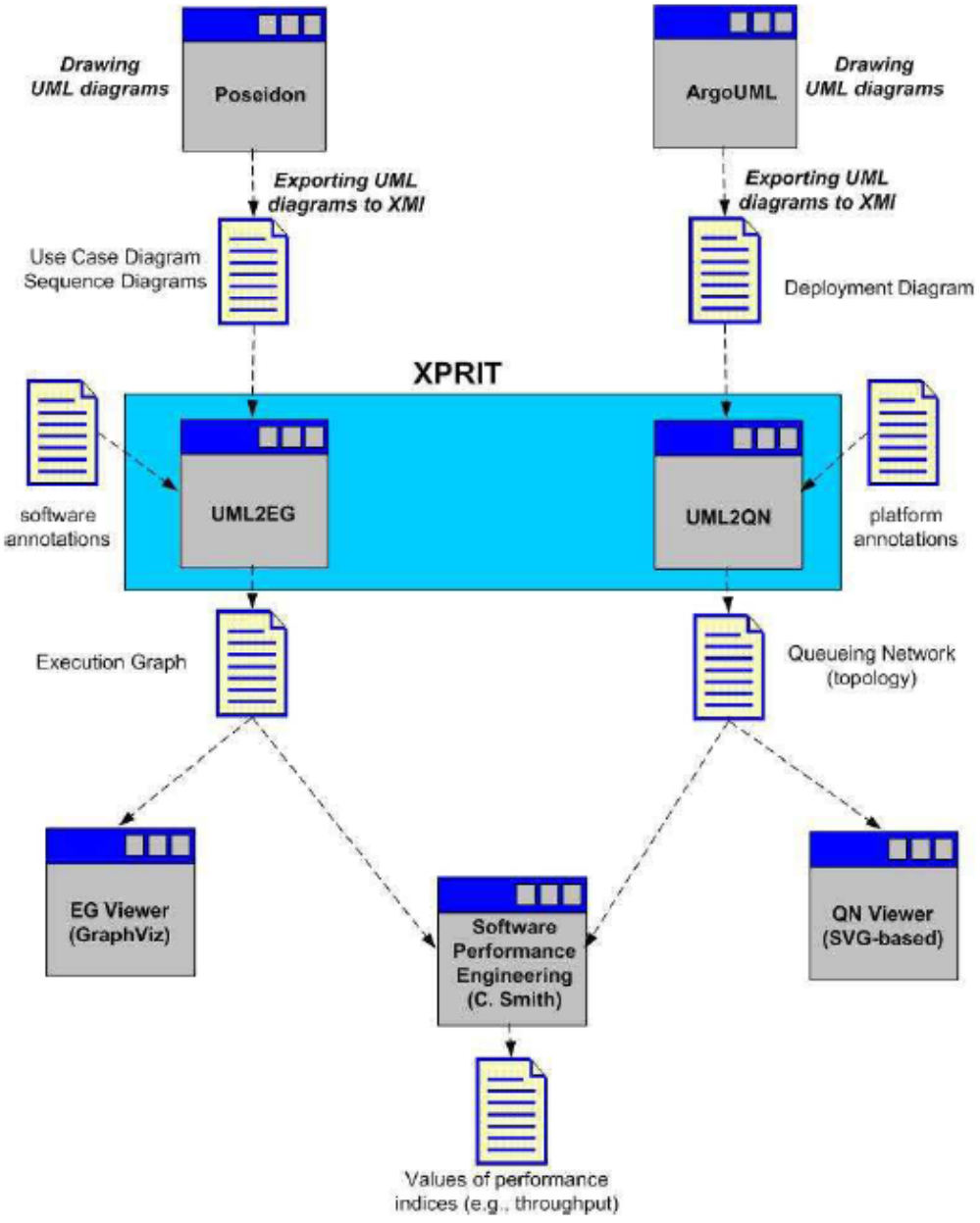

Figure 3.3: XPRIT tool block diagram [36]

(GSMP) from UML state and activity diagrams; several extensions are proposed by the authors to allow the association of events with exponentially distributed and deterministic delays. The approach is implemented in DSPNexpress 2000, which includes an efficient numerical solver of discrete-event stochastic systems underlying UML diagrams and Petri Nets.

During the definition of UML SPT, Petriu and Shen [101] developed a graph-grammar based transformation algorithm from UML SPT annotated diagrams expressed in XMI format [93] to Layered Queueing Network performance models. After UML SPT was officially adopted, Xu et al. [128] illustrate how to analyze and improve system performance using deployment and sequence diagrams with SPT notations; an LQN model is generated and evaluated in order to detect issues and adjust the UML design.

A first attempt to use SPT notations shows how they can be inserted into use case and statechart diagrams [78]. In use case diagrams, each scenario should be associated with an *openLoad* or *closedLoad* and the corresponding performance parameters should be defined (PAoccurence, or PApopulation and PAextDelay respectively). Also, a usage probability is attached to every edge connecting an actor to a use case. A scenario will be considered belonging to PAstep stereotype and will have a tagged value called PAprob, which is computed using the following formula:

$$P(x) = \Sigma_{j=1}^{m} (p_i \cdot P_{ix}) \ , \tag{3.1}$$

where p_i denotes the i-th user frequency of software usage ($\Sigma_{i=1}^{m} p_i = 1$), P_{ix} is the probability that the i-th user makes use of the x-th use case ($\Sigma_{x=1}^{n} P_{ix} = 1$). The SPT profile suggests that each scenario should be decomposed into steps illustrated by activity or collaboration diagrams. The authors have a different perspective, they use statechart diagrams to describe the life of each object in the system; each class with dynamic behavior will be modeled by a statechart. Activities are stereotyped as PAstep, thus being able to model parameters such as response time, demand, repetition, delay. If transitions are also stereotyped as PAstep, the probability attribute can be used to avoid non-determinism among several transitions originating from the same state; network delays can also be modeled in this case.

The previous research was continued in [71] by exploring the role of activity diagrams in order to refine activities in statecharts. Regarding performance annotations, each activity and transition (routing rate) will be stereotyped as PAstep, specifying PAprob (for routing rates) and PArespTime (for action durations); in case probabilities are not mentioned for transitions, they are considered equiprobable (equal probabilities are assigned to transitions originating from the same action). In order to obtain a formal performance model, LGSPNs are derived from each activity diagram and then they are composed into a scenario model or system model. A drawback is that resource capacity is not considered (an infinite amount is assumed). A Java module has been developed to take as input XMI files obtained by defining the statechart and activity diagrams in the ArgoUML tool [1]; a LGSPN model is generated by the module and it can be further analyzed by the GreatSPN (GRaphical Editor and Analyzer for Timed and Stochastic Petri Nets) tool [2] in order to get performance measures.

Balsamo et al. [12] describe an automated algorithm to transform UML SPT diagrams satisfying a few constraints (service demands exponentially distributed only) into a product-form Queueing Network performance model. The transformation is based on the mapping shown in figure 3.4.

The algorithm works according to the following steps:

- Each deployment diagram Node is mapped to a service center.

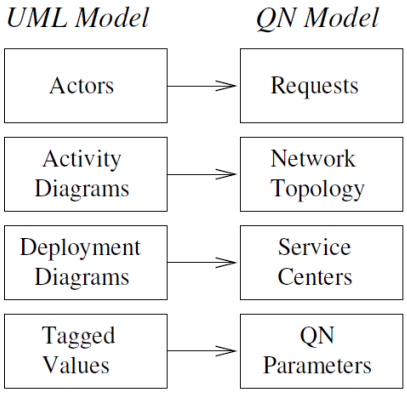

Figure 3.4: UML-QNE model transformation [12]

- Each Actor in use case diagrams is mapped to a class of customers in the QN.
- Each activity diagram associated to an Actor generates the routing matrix for the current class of customers according to the predecessor-successor relationship of Action states. Routing probabilities are obtained from the PAprob tag associated to UML transitions.

Input diagrams are obtained by using the ArgoUML tool [1], UML-QNE generates an internal representation of a QN, as depicted in figure 3.5, which is solved by MVA, one of the popular solving approaches described in section 2.1.2.

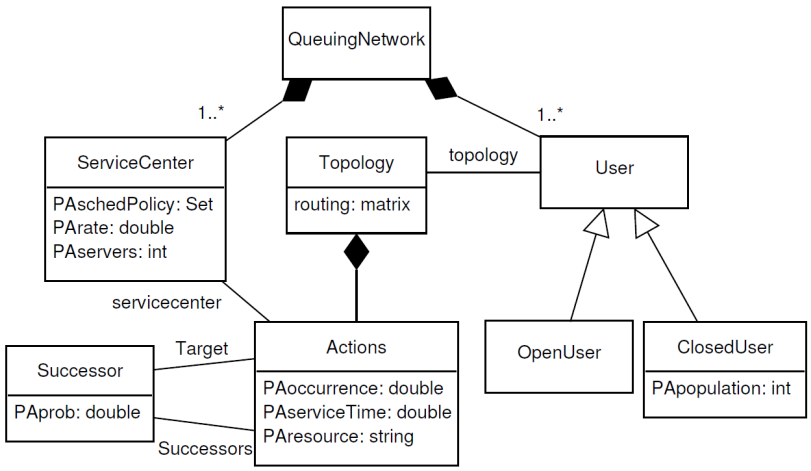

Figure 3.5: UML-QNE QN model [12]

UML diagrams with SPT annotations can also be transformed into CSM [97], an intermediate performance model allowing for flexibility in the transformation process, as presented in section 2.1.1. Query/View/Transformation (QVT) of Meta Object Facility (MOF)

[86] is recommended to define the model transformation, but, due to lack of tools to support this language, XSLT (Extensible Stylesheet Language Transformations) is used. XSLT operates on models represented in XML format, as illustrated in [55] when transforming UML models to LQN.

Each class in the meta-model has attributes (shown in figure 3.6), which correspond to tagged values in the UML SPT Profile.

CSM Class	Attributes
Component	ID; name; host ProcessingResource ID ref; 'is active' flag; description (opt); multiplicity (opt); containing component ID (opt)
ActiveResource	ID; name; time per operation; scheduling policy; description (opt)
Scenario	ID; name, collection of Steps
Step	ID; name; Component ref, host ProcessingResource demand, optional collection of pairs of ExternalService ID refs and demands; probability (opt); repetition count (opt); subscenario ref to nested Scenario (opt); description (opt); selection policy (opt)
Start	*Step attributes* + Workload ID ref
End	*Step attributes*
ResourceAcquire	*Step attributes* + Resource ID ref; resource units (opt); priority (opt)
ResourceRelease	*Step attributes* + Resource ID ref; resource units (opt)
Workload	ID; arrival stream type (open or closed); arrival process; distribution type; closed system population size (opt); mean inter-arrival delay (opt); lower bound on the inter-arrival delay (opt); upper bound on the inter-arrival delay (opt); inter-arrival process description (opt)
PathConnection	ID; Message ID ref (opt); condition (opt); label (opt)
Sequence	*Path Connection attributes* + source Step ref; target Step ref
Branch	*Path Connection attributes* + source Step ref; target Step refs
Merge	*Path Connection attributes* + source Step refs; target Step ref
Fork	*Path Connection attributes* + source Step ref; target Step refs
Join	*Path Connection attributes* + source Step refs (2 or more); target Step ref
Message	type (none, asynchronous, synchronous, reply); size; multiplicity (opt)

Figure 3.6: Attributes of CSM meta-classes [97]

The first phase, U2C, consists of converting deployment and activity diagrams into CSM: input consists of XMI files representing the UML model; the CSM internal representation is a DOM (Domain Object Model) tree that can be exported in XML format. Some of the resources and components in UML SPT can be directly mapped to CSM objects, as shown in figure 3.7. Attributes are obtained from tagged values (figure 3.6).

From each activity diagram, a CSM scenario is obtained by connecting steps with PathConnector objects; pseudostates are converted into corresponding PathConnector objects. If two pseudostates are consecutive, a dummy Step is inserted between the two connectors in the CSM model. Each partition (swimlane) of the activity diagram corresponds to a component in CSM, which should be found in the deployment diagram (describing resources or components of the system). A transition from one swimlane to another implies releasing one CSM component and acquiring the other. If the specification is incomplete, default values are assigned to parameters and reports are provided to the user after the conversion; ambiguities are to be solved explicitly by the user.

Another intermediate model mentioned in section 2.1.1 is IM, represented as an XML tree [56], used when transforming annotated UML models into LQN models. Transformation rules between models are expressed using XSLT, in order to make use of low-level operations on XML trees, such as XMLgebra and XACT [64, 65]; the implementation makes use of DTD and XPath to operate directly at XML level the rules defined at a higher abstraction

Type of Object in UML	Stereotype in STP Profile	Type of Object in CSM
Deployment Diagram		
Node	PAresource	Passive Resource
Node	PAhost	Processing Resource
Component	PAresource	Passive Resource
Component	none	Component
Activity Diagram		
SimpleState	PAstep	Step
CompositeState	PAstep	Step with nested Scenario
PseudoState: Initial	none	Start Step
PseudoState: Fork	none	Fork PathConnector
PseudoState: Join	none	Join PathConnector
PseudoState: Branch	none	Branch PathConnector
PseudoState: Merge	none	Merge PathConnector
FinalState	none	End Step

Figure 3.7: Mapping of UML SPT stereotypes to CSM types [97]

level by graph transformations. The input, expressed as UML (in XMI format), is converted into IM based on an algorithm that maps UML nodes and states (from activity diagrams) to IM concepts (some of these are shown in figure 3.8). The second conversion, from IM to LQN needs mapping between the two domains of IM and LQN respectively. A challenging transformation rule is deciding whether to assign groups of steps (from IM) to LQN phases or activities; this was implemented by checking if there is a fork that does not involve a reply. If there is such an operation, then the corresponding entry is mapped to an activity, otherwise it is considered a phase.

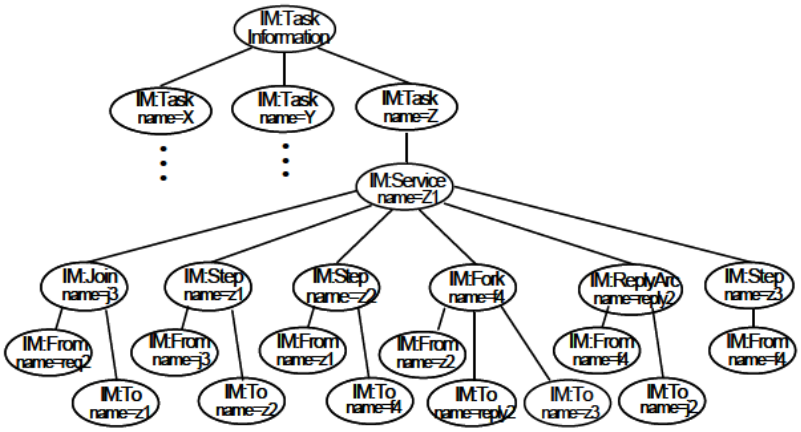

Figure 3.8: Intermediate Model example [56]

3.3. Frameworks

The next step towards flexibility in deriving performance models consists in building frameworks. They allow integration of various UML tools and notations, meaning different kinds of diagrams that can be used to describe the system, with the possibility to derive several alternative performance models. This is a kind of N-by-M problem to translate N design notation types into M performance model types.

Performance by Unified Model Analysis (PUMA) [126] is a framework, based on CSM, which allows integration of various tools to facilitate CSM extraction from the design model (XMI representation of UML diagrams or UCMs) and also to translate CSM into performance models (LQN, Petri Nets, QN).

The UML-to-CSM translation, also described in [97], is focused on performance related annotations. In this case, the challenges of the translation are pointed out:

- filtering out relevant information (UML models contain multiple system views);
- handling incomplete or inconsistent UML models.

Because of the wide range of UML diagrams, a UML model can contain redundant information. On the other hand, not all aspects of the system are important for performance analysis. This is the reason of emphasizing the filtering process.

Specifications in UML format are semi-formal; usually the notation can be extended to accommodate the intended purpose and domain of the system that is being modeled. Thus, a UML model cannot be checked for correctness. However, it can be checked for consistency and completeness as far as performance aspects are concerned. Performance modeling requires a limited set of conditions and notations that need to be checked; for instance, model construction requires that scenarios are continuously connected, and that all resources acquired should be released. Another issue derives from concurrent execution of scenarios. They should be analyzed separately as individual CSMs that interact; the interaction should be modeled separately too and should be performed through resources.

The PUMA approach in scenario-based performance engineering is shown in figure 3.9. For each target performance modeling tool, a conversion from CSM is defined, as presented in section 2.1.1.

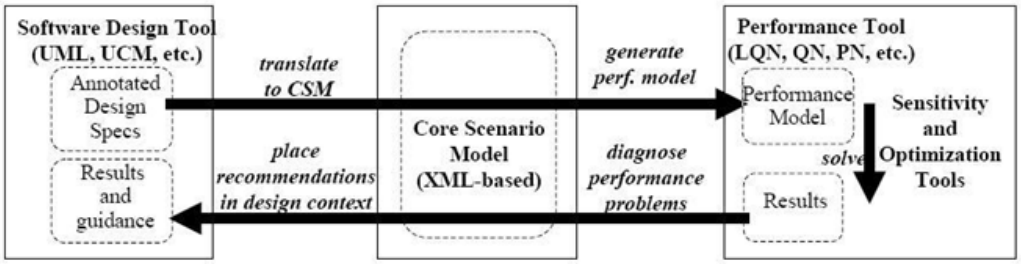

Figure 3.9: PUMA approach [99]

Another framework proposal is Unified Performance Engineering (UPE) [121], a model-driven SPE framework. The performance model evolves together with the system

design model during the design stage. Unlike the previous example of PUMA [126], where the performance annotations were included in the design model, in this case, additional information is kept in views, so the system design model can serve as a basis for other specific models (e.g. security model).

Models and transformations between them are based on MOF and QVT [86], using a high-level transformation language. The framework architecture is shown in figure 3.10. Intermediate Performance Model (IPM) is similar to CSM [97], it includes both design and performance information, after combining the performance view with the UML model. IPM will be transformed into a specific performance model to be analyzed by appropriate performance analysis tools. Feedback mechanisms (results inserted into the performance view) imply inverse transformations.

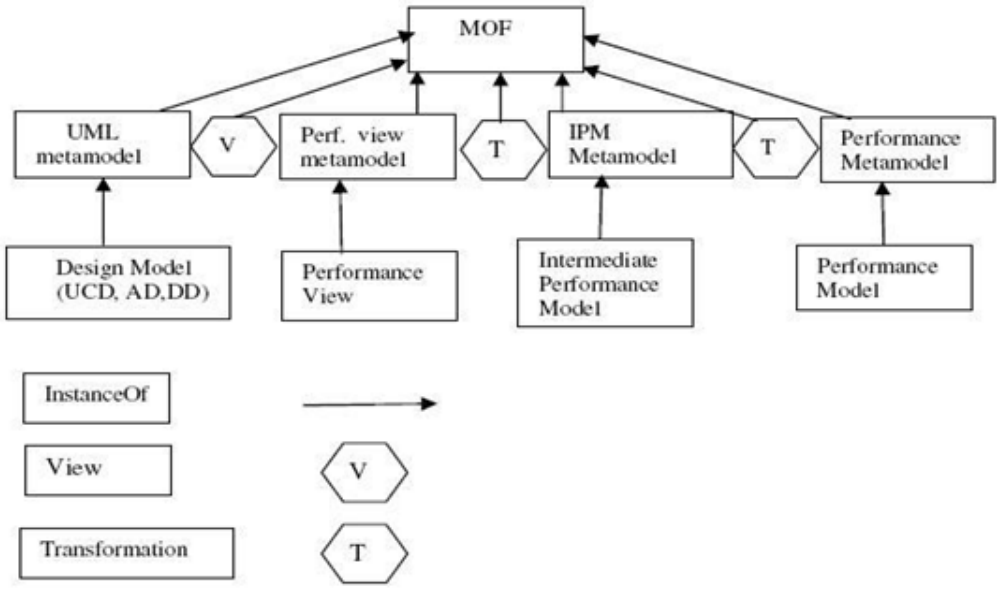

Figure 3.10: Unified Performance Engineering (UPE) framework architecture [121]

In order to make use of the diversity of small, specific, available tools based on Model Driven Architecture (MDA) approach [88] and XML format, a tool interoperability framework is suggested in [41]. MDA supports software systems development by transformation of platform-independent models into platform specific models, executable components and applications. The focus is moved from coding to modeling. Both the UML system design model and the performance model (LQN has been chosen, because of the substantial work that has been carried out in this field) are described as MOF [87] meta-models, so that the transformation can also be expressed at the meta-model level. It is eventually translated (automatically) into a specific implementation technology, so that the UML model instance can be converted into an LQN model instance (for the particular software system being analyzed).

A three layer approach is described for the model transformation framework: a technology independent MetaModeling Layer (MML), a technology specific Model Implementation Layer (MIL) and a Tool Layer (TL). MML has three sub-layers, extensions of the corresponding

MOF metadata architecture levels; at the M1 sub-layer, the SPT Profile is used to annotate UML model elements with performance-oriented data, obtained by estimations of experts or by measurements on similar systems.

As it was the case for CSM [97], because tool support is not available for QVT, XML based technologies such as XSLT are used for transformations. In the prototype implementation of the model transformation framework, the MTL (Meta Transformation Language) transformations have been translated to XSLT rules that are given as input to a standard XSLT processor, which converts the XMI representation of the UML model into the LQN model, also in XMI format. Framework flexibility can be improved by considering CSM instead of the LQN model; thus, a variety of PMs would be addressed with the cost of two transformations instead of one.

It would seem that direct XSLT conversions from UML models to PMs are more efficient, but this approach shows limitations in terms of reusability and maintainability, due to verbosity and poor readability of XMI and XSLT. By separation of transformation specification from its implementation, maintainability is facilitated and reuse of patterns in transformation is possible. The productivity can also increase, by automation; this aspect will be even more obvious when QVT tools will be available to convert transformations directly into executable code (without needing the intermediate XSLT transformation). Conceptually, it is necessary to move from the already established platform interoperability at tool layer to the more effective meta-model-driven model interoperability. The ultimate goal is to have a single CASE tool that would be transparently and efficiently used to create models, execute model transformation and also perform model evaluation.

In the last few years, developments have been made to provide complete environments, starting at system modeling, having the possibility of applying performance evaluation repeatedly before system implementation, in order to gradually improve system design.

Such an environment is presented in [109]: TANGRAM-II covers the whole modeling cycle, from model building to model solution and experimentation. An advantage is that it offers both an analytical approach and a simulation one, providing flexibility for performance analysis: the user can choose the desired method.

A simulation-based performance analysis tool (that also provides an analytical model for flexibility), called PerfCenter, has also been recently developed and presented in [120, 43]. Its purpose is to help the data center architects by providing means to automatically analyze different design alternatives. The results are provided as response times, throughputs, but the main drawback in using this tool is that it requires the input to be specified in a particular textual format, specification that needs to be provided manually: there is no tool that can generate such output starting from system diagrams.

Another approach to provide flexibility is the use of frameworks where the model solver component may either be analytical or a simulator (usually, simulators are preferred for accurate results).

Such a framework is described in [39] and its stated intent is integration of software models with platform models, in order to devise meaningful performance models. The paper focuses on defining platform models and mapping software components onto items from the platform model; the UML-RT (UML-Real-Time) notation is used for model prototypes.

Another, more general framework (based on standard UML notation), is presented in [46]. Its advantage is the use of a library of optimized node models (built by performance engineers). These node models have been defined by expert performance engineers and are used

by system architects to yield system models as combinations of such nodes. The system models are translated to performance models using the PMIF notation, by using a translator tool (e.g. XPRIT) and then they are simulated (or solved) using another tool (e.g. TANGRAM-II).

3.4. UML 2 or MARTE-Compliant Methodologies and Tools

A MARTE compliant performance evaluation tool, developed as a plug-in for Rational Software Architect (RSA) v7, is described in [118]. The tool implements an algorithm that converts an input UML MARTE model into a Performance Evaluation Process Algebra (PEPA) model, which can further be evaluated by solving the underlying Continuous Time Markov Chain (CTMC).

A simulation tool that uses UML 2 diagrams is DESMO-J (Discrete-Event Simulation and Modeling in Java). N. Knaak and B. Page explain the benefits of using UML 2 in [66] and express their intention to adopt these diagrams as input for DESMO-J. After a review of diagram types and their utility (structural, behavior, and interaction diagrams), activity diagrams are presented in more detail. In the context of DES, these diagrams provide means to express both event-based simulation (modeling event routines) and process-based simulation: modeling the lifecycle of simulation processes, by using features such as concurrency, object flow and message passing.

V. Cortellessa et al. have implemented MOSES (MOdeling Software and platform architEcture in UML 2 for Simulation-based performance analysis) [40], a methodology that accomplishes integration of software models with platform models, in order to devise meaningful performance models. This approach allows estimating the performance of the same software architecture on multiple platform architectures without underlying (possibly incorrect) model transformations. Instead of such transformations, this methodology integrates in one notation software and platform models plus annotations, thus building a performance model in a unified notation.

The general methodology involves the following steps:

- Separately build software architectural model and platform architectural model.
- Merge software and platform model to obtain an integrated architectural model.
- Annotate the integrated model with data related to performance.
- Simulate the annotated model to obtain the indices of interest.

Tool support was needed for visual modeling and simulation: Telelogic Tau G2 was chosen because it also provides a language, SDL (Standard and Description Language), to describe statecharts and to model actions. Hence, specification of platform details (needed for performance analysis) can be delayed: SDL blocks can be inserted at a later time, not necessarily at software design time.

A transformation methodology from UML diagrams with MARTE annotations to a LQN model is described by Petriu in [100]. The conversion steps are presented below:

1. Generate LQN model structure
   (a) map high-level component instances to LQN tasks according to patterns;

(b) map deployment diagram nodes to LQN hardware devices;

2. Generate LQN entries, phases, activities from scenarios

   (a) for each scenario

      i. generate a LQN reference task and its dummy processor corresponding to the scenario workload;
      ii. match messages with inter-component communication style from patterns;
      iii. map external message calls to entries;
      iv. for each entry
         A. group corresponding execution occurrences according to patterns;
         B. map groups to phases or activities;
         C. for each phase and activity compute service time and number of calls.

3.5. Summary

The chapter starts with a brief review of the input formalisms and transformation methodologies used until year 2000: SPAs, CLISSPE, CHAM, MSC, LTS, or UML diagrams were converted to EG or QN on which performance analysis was performed.

After year 2000, the process was automated by tool development along with new formalisms and emerging transformation techniques. High level descriptions can be provided in the form of UCMs. An even better idea is to combine information from several representations (UML, MSC, ADLs) of the same system. Before having dedicated annotations for performance analysis, methods were elaborated to convert UML diagrams into Petri Nets or Markov Processes. After the adoption of UML SPT, several approaches described how performance annotations could be used in different types of diagrams. Conversion algorithms were developed to obtain intermediate models (CSM, IM) from UML SPT diagrams.

A step forward towards flexibility is taken by using frameworks that allow conversion of N input formalisms to M performance models: PUMA, UPE. In the last few years, complete environments were developed, such as TANGRAM-II, or PerfCenter.

Since UML 2 and MARTE were recently adopted and are still subject to improvements, there are few analysis tools relying on them (DESMO-J, RSA plug-in). This is the reason for which an approach needs to be defined concerning the way MARTE notations can be mapped to a flexible, generic performance meta-model which is the subject of the following chapter.

4. HYBRID ANALYTICAL/SIMULATION MODEL AND SOLVER

4.1. Novel Performance Model

Performance models have been defined separately so far either for analytical solving or for simulation. This section describes a hybrid model that allows combining of a simulation submodel with an analytical approach applied to a QN model. In order to apply the two types of methods, the performance model is partly transformed into a simulation submodel, and also into a QN model for the analytical solving stage.

In addition to performance related information from the UML input model, entity interactions are stored in the hybrid model. In order to improve analysis efficiency, only the necessary information is kept in the model, with no redundancies.

The outcome of this research activity is building a performance meta-model that combines flexibility and conciseness. Its actual purpose is to provide a propagation environment for intermediate analysis results shared by the simulation submodel and the QN model. Also, the performance model allows model hierarchical or sequential decomposition into layers, depending on the input system specification.

4.1.1 Hybrid Meta-Model Overview

Reference models from both analytical and simulation approaches were chosen: the simulation model from $UML-\Psi$ [74] and an intermediate meta-model called Core Scenario Model (CSM) [97].

The two selected meta-models are similar, having a scenario oriented structure, so the new performance meta-model is derived in a straightforward manner in [28], a research paper written by the author of this thesis. The meta-model in [28] was improved and extended [31, 32], as shown in figure 4.1.

The main three categories of entities are typical in scenario oriented models: workloads, resources, and steps; the steps are grouped into scenarios. Workloads can be open or closed (the adopted notation in the MARTE Profile is still undergoing substantial changes, so the notation in SPT is used to specify distributions). Resources may be either active (processing units or modules) or passive (buffers and other kinds of logical resources, or even LANs, since they are shared within networks). PassiveResources can be acquired or released by ResourceActions and also inherit attributes such as "Utilization" and "Throughput", since these parameters are very likely to reveal the presence of bottlenecks in the system. ActiveResources are used by ProcessingSteps: the corresponding action is executed on a processing unit, also called a host. Several steps may rely on the same host and if the host is not multithreaded, the particular scheduling policy will be applied and the requests will be queued up until they can be serviced. ForkAction and JoinAction allow definition of parallel sequences of steps (they may not be executed in parallel if they rely on shared resources available in limited amounts). CallAction is useful when calling a scenario within another scenario.

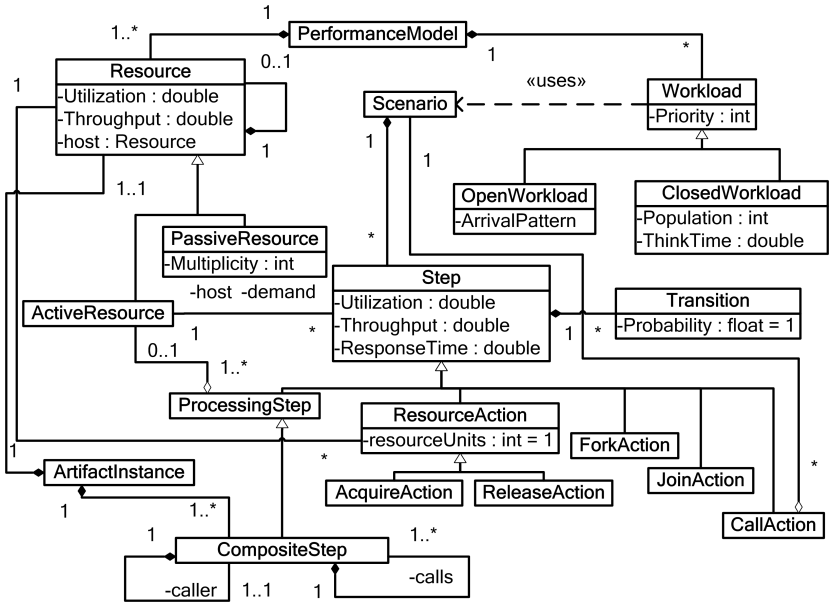

Figure 4.1: Hybrid meta-model [31]

Performance parameters are available for each entity by inheritance from PerformanceObject, the base class for all entity types presented in the meta-model depicted in figure 4.2. The ArtifactInstance entity was added to denote a lifeline from a sequence diagram, connecting a host resource to the composite steps on a lifeline. A CompositeStep is a processing step that calls other (possibly composite) steps and also has a caller step (deduced from the source of the incoming message in the sequence diagram).

The following subsections present the model decomposition strategy and the transformations of the performance model into analytical and simulation (sub)models.

4.1.2 Model Decomposition

Level numbers are assigned to entities of the performance model as the model is built. The input system model can be provided in different formats, depending on the information available about the system and on the nature of the system.

In case sequence diagrams are available, the model is hierarchically decomposed, based on call nesting, as shown in figure 4.3.

For a system specified using activity diagrams, the model is sequentially decomposed, following request chains along transitions. An example is shown in figure 4.4.

The explanation for using a sequential decomposition in case of activity diagrams is based on UML representation practices. Usually, UML steps rely on one resource (active or passive) at most and not on other steps. Resource dependencies can be defined in deployment diagrams, which usually have a maximum of two layers denoting the software applications (services) and the platforms they are running on. Hence, performance submodels obtained from UML MARTE activity diagrams will lead to QN models with two layers at most. In this

PerformanceObject
+ID : int +Name : string #Level : int +CallProbabilities +Clients +ResponseTimes +ResponseTime : double +Utilization : double +QueueLength : double +Throughput : double +PopulationCount : int +ThinkTime : double
-SetLevel() +Convergent(in PerformanceResults, in delta : double) : bool +UpdateResultsFromSimulation() +PropagateRequests() +ComputeDemand(in Resource, in maxLevel : int) : double +ComputePassiveTimeDemand(in PassiveResource, in maxLevel : int, in requestedUnits : double) : double

Figure 4.2: PerformanceObject base class

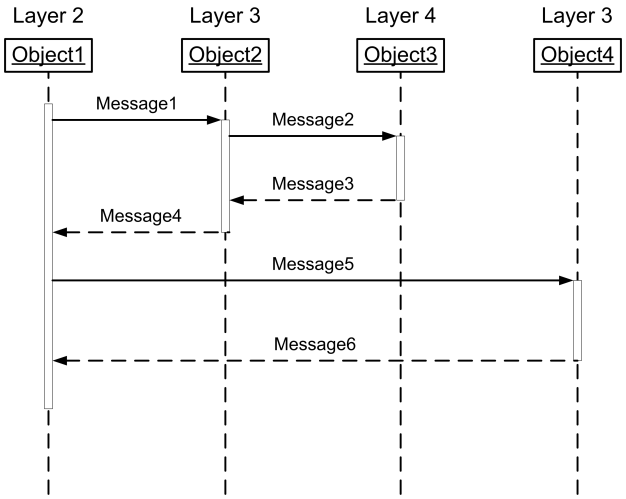

Figure 4.3: Performance model layers based on nested calls from sequence diagrams

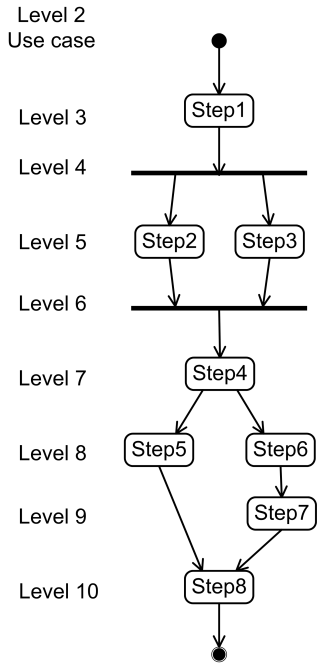

Figure 4.4: Performance model levels following control flow in activity diagrams

case, a hierarchical solving method is not applicable. The only possibility to decompose such a model is sequentially, following transitions between steps.

Submodels are obtained by decomposition using the batched layering strategy [48]: submodel *l* includes entities on level *l* as servers and all their clients, regardless of the level of the clients (of course, higher than *l*, but not necessarily equal to *l-1*). Such submodels are only used during the analytical solving for input sequence diagrams, where the hierarchical approach is applied; for activity diagrams, levels are only useful when separating the simulation submodel, the analytical approach being applied to the entire model.

In the remainder of the thesis, the following convention (based on figure 4.4) is made when referring to level values:

- Level *l* is considered "above" or "higher than" level *k*, when $l < k$.
- Level *l* is considered "below" or "lower than" level *k*, when $l > k$.

4.1.3 Simulation Submodel

The pure simulator is based on the simulation model defined by Marzolla in $UML - \Psi$ [74]; the original implementation based on single-threaded coroutines was improved by adopting a multi-threaded approach, using thread pools.

The simulation submodel, derived from $UML - \Psi$, is built from all entities with level values greater than or equal to a certain level *k*, which will be referred to as the *simulation level*. This means that entities on the *simulation level* generate scenarios and all their clients

will be transformed into requests for the simulation submodel. Entities are either Steps from Scenarios, or Resources: Steps are considered clients of the Resource they are using, and of the Steps towards which they have transitions. Steps that are clients of Resources are excluded from the submodel workload, since resources cannot be considered use cases. Each Step on the *simulation level* generates a use case, whose composite action starts with that Step and includes all steps following it, on all remaining levels.

When simulating steps from activity diagrams, the collection of outgoing transitions presents a set of options, each having a probability of being called next. In case of steps from sequence diagrams, called CompositeSteps in the meta-model, there is an additional aspect to consider in the list of outgoing transitions: their nesting is important.

The author of this thesis extended the simulation model from $UML - \Psi$ to support nested calls, in addition to sequential calls that were originally supported. The base class for all action entities, which initially included only a list of transitions, has been extended to keep a list of calls and a method that handles the calls during the simulation. The extensions are highlighted in figure 4.5.

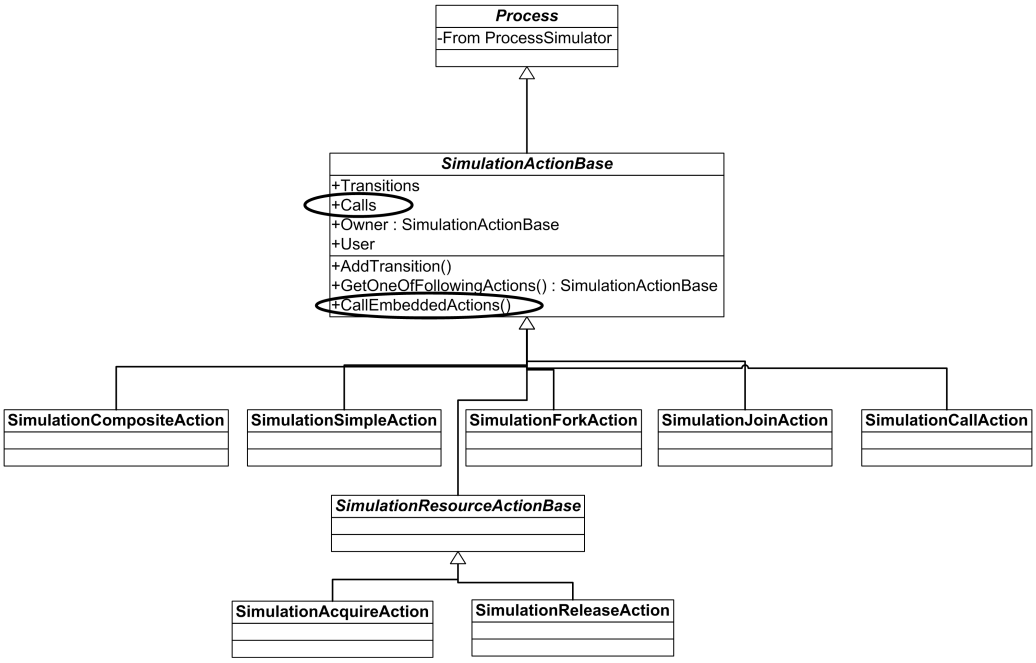

Figure 4.5: Extensions of the simulation action model to support nested calls

In case there are several steps on the *simulation level* called by the same client, because of the use case oriented structure of the simulation model, all such calls will be part of a single use case, in which call order is emphasized: the first call is to the first action in the list, and, after it finishes execution, control is passed to the second action and so on. In order to capture this kind of call sequencing, a transformation was defined from the hierarchical model to a composite activity with explicit transitions between sequential calls, as shown in figure 4.6.

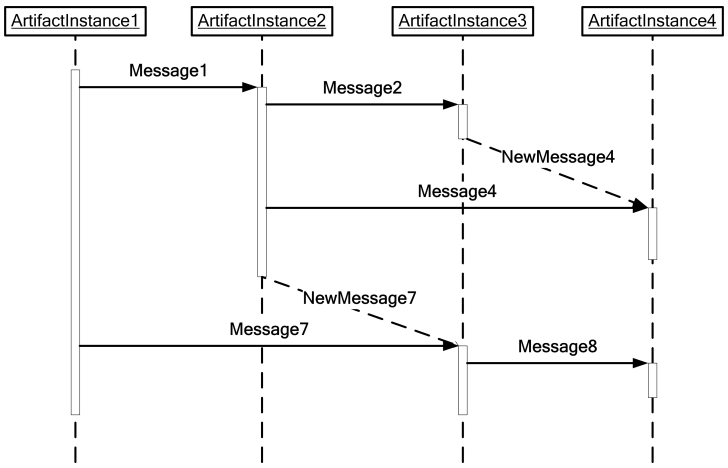

Figure 4.6: Transformation of composite steps: nested calls become sequential messages

Only call messages have been depicted, return messages are not relevant for the sequential order; the transformation affects message 4 (in case the *simulation level* $k = 3$) or message 7 (for $k = 2$), by changing their source to be the previous step on the same level from the execution flow. In case the *simulation level* $k = 2$, only message 7 is transformed, message 4 is not altered since nested calls are supported for simulation actions inside the use case, only the use case has the single entry point restriction.

4.1.4 Queueing Network Model/Submodels

The QN submodel(s) on levels above the simulation submodel have been defined at different granularities, depending on the way the input model is specified. The input model may provide information either on hierarchical (sequence diagrams) or on sequential (activity diagrams) interactions between steps.

A fine granularity is used for input models described by sequence diagrams: each submodel on a certain layer is solved using results from analysis of lower-level submodels. Input service times for a particular QN submodel are cumulated from response times on lower levels obtained both analytically (during LQN solving) and by simulation (for levels lower than the *simulation level*).

In case of input models defined using activity diagrams, due to lack of layering, a single QN model is defined, which uses results from simulation of lower-levels. In order to clearly separate the simulated submodel, entity levels are established sequentially, instead of hierarchically: level number increases along the activity flow, following the transitions between steps. This sequential approach aggregates all service times on levels higher than the *simulation level* and response times on lower levels (simulation submodel) at use case level.

The submodels subject to analytical solving are built as follows:

- service centers: system resources, both active and passive
  - resource demands are computed from service times on levels above the *simulation level* and response times obtained from simulation;

- tasks:
  - steps on the current submodel (QN) layer, in case of hierarchical decomposition,
  - use cases (scenarios), when a single QN is built for the entire system.

In case of passive resources, simulation effects are present in values of input demands for service centers of the (L)QN model.

In case of hierarchically decomposed models, these demands are computed as sums of service times for all composite and non-composite steps enclosed between an AcquireAction and a ReleaseAction for a passive resource. Both resource management actions should be performed by the same entity (they belong to the same lifeline) and are positioned on the same level. The execution flow between these two actions includes steps on the same level, which in turn are composed by steps on lower levels, whose response times have been obtained by simulation (for levels below the *simulation level*).

For sequentially layered models, demands are computed considering all levels in the QN model: service demands from the input UML model are considered until the *simulation level* is reached, then response times are used instead, since they are more accurate, being obtained by simulation.

Each of the above mentioned time intervals (demand) is weighted by the quantity of held resources; steps situated between an AcquireAction and a ReleaseAction for a passive resource, R, hold a quantity given by the request in the AcquireAction.

4.2. Hybrid Solver

4.2.1 Iterative Process

The hybrid analysis method is an iterative process which was refined in time.

Iterations are repeated until values of parameters converge, a predefined *iterationCount* is reached, or the user explicitly requests cancellation of the solving process. The convergence test is applied to values for utilization, throughput and response time obtained in consecutive iterations: the relative change should be lower than a predefined constant. In case of the simulated submodel, which also has a convergence test for iterations within the simulation step, it has a different meaning: intervals of values are computed continuously for each parameter and model node, so it is checked whether the intervals for each parameter are narrow enough, in order to be able to consider the mean value relevant.

The value of the *simulation level* is configurable before starting hybrid solving: from 3 (the system workload is on level 1 and the use cases are on level 2) or 2 (for sequence diagrams, no use cases are needed) to the maximum level in the performance model. Choosing this maximum value for the *simulation level* leads to pure analytical solving.

After the original approach and the reasons for improvements are presented, the iteration structure for the current approach is described for two types of system models: hierarchical (modeled by LQN) and sequential (modeled by QN).

Approach Refinement

The initial hybrid solving process is based on model layering: each submodel on a certain layer is solved using results from analysis of submodels on other layers (higher layers or lower

layers, depending on the result propagation direction). This idea was successfully applied in LQNS [48] and is appropriate for LQN input models, having multiple layers. In case of Closed Queueing Networks (CQN), Franks proves there is a two-way dependency between levels, so performance parameter values need to be propagated both downwards (think time) and upwards (response time).

Based on the previously presented model decomposition strategy (section 4.1.2), a simulation step is inserted from the *simulation level* downwards between two analytical steps (top-down and bottom-up). The pseudo-code of these three-step iterations, as presented by the author in [30], is depicted in figure 4.7.

$Cummulate(S_1)$ // add service time values for all servers
DO iteration
  $Generate(Z_1)$ // generate think time values for requests
  FOR submodel $l = 1..k$
    $MVA(l, Z_l, S_l) \Rightarrow \rho_l, \lambda_l$ // utilization, throughput
    $Z_{l+1} = f(\rho_l, \lambda_l)$ // propagate think time values
  $Simulate(M_{k+1}, Z_{k+1}) \Rightarrow \mathrm{P}_{k+1}, \Lambda_{k+1}, \mathrm{T}_{k+1}$ // utilization,
                                  // throughput, response time
  FOR submodel $l = k..1$
    $S_l = f(\mathrm{T}_{l+1})$ // propagate service time values
    $MVA(l, Z_l, S_l) \Rightarrow \rho_l', \lambda_l'$
WHILE (NOT $Convergent(\mathrm{P}_1, \Lambda_1, \mathrm{T}_1)$ AND iteration count not exceeded)

Figure 4.7: Initial hybrid iterative approach: three step iterations applied on fine-grained analytical submodels

The LQN oriented approach was subject to further experimentations and proved to have problems when applied to UML MARTE input models based on activity diagrams to describe the system behavior, because of the mostly sequential nature of such diagrams, as already explained in subsection 4.1.2. This is the reason for keeping the fine granularity only for input models described by sequence diagrams.

Another addressed problem of the initial approach is the difference between the way input data is specified for the analytical solver, compared to the simulator. While the analytical solver based on Mean Value Analysis (MVA) needs mean input values for request distributions, the simulator includes pseudo-random number generators in order to cover larger fluctuations. The initial top-down solving step propagates mean values to be used as input for the simulation, whose accuracy thus becomes useless.

The solution is to keep only the bottom-up analytical solving step, applied after the simulation. The input parameter values are generated according to request distributions and propagated down to the *simulation level* considering branches and probabilities.

Hierarchical Approach

Each iteration starts with initializing resource performance parameters, where results of simulation and analytical computations will be stored. Workloads are usually defined as distributions,

since exact values cannot be known for arrival rates (throughput) and external delays (think time). Distributions are specified as tagged values in the annotated UML input model. Each iteration will have different values for these input parameters, as provided by random number generators in *Generate* (Z_1, R_1).

Requests are then propagated to the *simulation level*, k, by following the transitions in each scenario, so that the simulated submodel will have clearly defined actors for the remaining scenarios.

The simulator runs model M_k, which consists of all submodels starting from level k down to the lowest level, and receives as input the set of propagated think time values and population count for closed requests, and the arrival rates for open requests.

Results from the simulation are saved in the performance model entities, such as resources and steps.

Prior to the analytical solving, mean values are computed for think time and arrival rate distributions, in order to obtain deterministic results.

Results are propagated both ways between the analytical solver and the simulator. The requests for the simulated submodel have parameters propagated from the model input along the scenario control flow: throughput (for open requests) and population count (for closed requests) are multiplied by transition probabilities, and think time is increased for each encountered processing step (analytical results for host resource response time are added to the think time value). A task that acts as a server in the submodel on a certain level l may become a client in the submodel on level $l+1$, if it relies on tasks on lower levels.

$$R_{i,l+1} = \Sigma_j \left(R_{j,l} \cdot P_{i,j}\right) \tag{4.1}$$

$$C_{i,l+1} = \Sigma_j \left(C_{j,l} \cdot P_{i,j}\right) \tag{4.2}$$

$$Z_{i,l+1} = \Sigma_{PS} RT_{PS.Host} + Z_{i,l} \tag{4.3}$$

where j iterates over the collection of clients on level l belonging to the server i on level $l+1$, $R_{i,l}$ is the arrival rate (throughput) of an open request chain i on level l, $C_{i,l}$, and $Z_{i,l}$ are the population count and external delay of a closed request chain i on level l. PS denotes a processing step on level l, and $RT_{PS.Host}$ refers to the response time of the active resource required by the processing step.

Service times are propagated upwards: they are computed from waiting times of servers from lower level submodels, as shown in section 2.1.2 equation 2.1: values are aggregated from the submodel immediately below the current submodel or the simulation submodel (see figure 4.8).

The pseudo-code is presented in figure 4.8, in order to provide a clear high-level description of the solving technique. The following notations are used:

$Z_l = \{Z_{i,l} | 1 \leq i \leq c\}$ denotes the external delay values, where c is the number of requests at level l, level 1 means pure clients (request generators).

$C_l = \{C_{i,l} | 1 \leq i \leq c\}$ denotes the population count values, where c is the number of requests at level l, level 1 means pure clients (request generators).

$R_l = \{R_{i,l} | 1 \leq i \leq c\}$ denotes the arrival rate values, where c is the number of requests at level l, level 1 means pure clients (request generators).

$\rho_l = \{\rho_{il} | 1 \leq i \leq r\}$ denotes the utilization values, where r is the number of resources (service centers) of the submodel on level l.

$\lambda_l = \{\lambda_{il} | 1 \leq i \leq r\}$ denotes the throughput values, where r is the number of resources (service centers) of the submodel on level l.

$w_l = \{w_{mj} | m \in S_l\}$ denotes the response time values, where S_l is the set of server tasks from the submodel on level l.

$S_l = \{s_{i,l} | 1 \leq i \leq t\}$ denotes service time values, where t is the number of server tasks from level l.

$P_k = \bigcup_{l \in L, l \geq k} \rho_l$ includes all utilization values for tasks on level k and lower levels.

$\Lambda_k = \bigcup_{l \in L, l \geq k} \lambda_l$ includes all throughput values for tasks on level k and lower levels.

$T_k = \bigcup_{l \in L, l \geq k} \{RT_{mj} | j \in S_l, m \in S_i, \forall i < k, i \geq 1\}$ includes all response time values for tasks on level k and lower levels, which provide service to tasks on higher levels.

DO iteration

$Generate(Z_2, R_2)$

FOR submodel $l = 2..(k-1)$

$R_{l+1} = f(R_l)$ - equation (4.1)

$C_{l+1} = f(C_l)$ - equation (4.2)

$Z_{l+1} = f(Z_l)$ - equation (4.3)

$Simulate(M_k, Z_k, C_k, R_k) \Rightarrow P_k, \Lambda_k, T_k$

$Compute(Z_2, R_2)$

FOR submodel $l = 2..(k-2)$

$R_{l+1} = f(R_l)$ - equation (4.1)

$C_{l+1} = f(C_l)$ - equation (4.2)

$Z_{l+1} = f(Z_l)$ - equation (4.3)

FOR submodel $l = (k-1)..2$

$S_l = f(T_{l+1})$ - equation (2.1)

$MVA(l, Z_l, C_l, R_l, S_l) \Rightarrow \rho_l, \lambda_l, w_l$

WHILE (NOT $Convergent(P_2, \Lambda_2, T_2)$ AND *iterationCount* not exceeded)

Figure 4.8: Hybrid iteration hierarchical pseudo-code [31]

Sequential Approach

The hybrid iteration structure is different for sequential models deduced from activity diagrams. Since such models have maximum two layers, the hierarchical approach is replaced by a sequential one. Levels are assigned following the control flow and their main purpose is to separate the simulation submodel.

Workload parameters are propagated the same way for simulation, the only difference is that they need not be propagated for the analytical solver, since the solver will analyze the entire system (not consecutive submodels in a bottom-up manner, as presented earlier). The difference is highlighted in figure 4.9.

Service times are aggregated at use case level from all service times on lower levels and response times from simulation, in order for the analytical solver to address the entire system model, as shown in figure 4.9.

DO iteration

$Generate(Z_2, R_2)$

FOR submodel $l = 2..(k-1)$

$R_{l+1} = f(R_l)$ - equation (4.1)

$C_{l+1} = f(C_l)$ - equation (4.2)

$Z_{l+1} = f(Z_l)$ - equation (4.3)

$Simulate(M_k, Z_k, C_k, R_k) \Rightarrow P_k, \Lambda_k, T_k$

$Compute(Z_2, R_2)$

$MVA(Z_2, C_2, R_2, S_2) \Rightarrow \rho_2, \lambda_2, w_2$

WHILE (NOT $Convergent(P_2, \Lambda_2, T_2)$ AND *iterationCount* not exceeded)

Figure 4.9: Hybrid iteration sequential pseudo-code

4.2.2 Analytical Algorithm Extensions

The algorithm is based on MVA extensions for multi-class mixed system models, accepting both open and closed requests, described in [68, 77] and is modified by the author of the thesis, after comparing computed values to simulation results. Both the open model solver and the closed model solving algorithm (Chandy-Neuse) are also adapted by the author to consider multiplicity of resources, by using a correction factor [117] for closed models and adjustments in input parameters for open models. The pure analytical algorithms (formulae) are presented in appendix A.

The Open Queueing Network (OQN) is solved first and the results are used to elongate the service demands of service centers that accept both closed and open requests, and then the resulting closed model is solved. A factor is computed using formula (4.4), and then this factor

is multiplied with the service time in the formula for response time from the Bard-Schweitzer approximation.

$$F_j = \frac{1}{1 - U_j} \ , \tag{4.4}$$

where U_j is the utilization for resource j, corresponding to all open requests.

This factor is used in the following expression for response time of resource j to closed requests of type i [68]:

$$RT_{i,j} = F_j \cdot S_{i,j} \cdot (1 + Q_{i,j}) \ , \tag{4.5}$$

where $S_{i,j}$ is the service time for resource j and requests of type i, and $Q_{i,j}$ is the queue length for one less request in each type of closed request as computed in the Chandy-Neuse estimation.

The same factor is used in [68] to compute the response time for open requests, after the Chandy-Neuse algorithm has been applied to closed requests:

$$RT_{i,j} = F_j \cdot V_{i,j} \cdot S_{i,j} \cdot \left(1 + Q_{closed_j}\right) \ , \tag{4.6}$$

where $V_{i,j}$ is the visit rate of requests of type i to resource j, $S_{i,j}$ is the service time for resource j and requests of type i, and Q_{closed_j} is the queue length of resource j for all closed requests as computed in the Chandy-Neuse algorithm (already adjusted by F_j).

Tests were performed for open models and the values obtained by applying formula (4.6), where $Q_{closed_j} = 0$, are far from values obtained by simulation. The reason is simple, F_j has the same value regardless of the considered request type i, while the formula for response time should be elongated by the wait time caused by other requests being serviced by the same resource. The wait time depends on the resource, but also on the type of request. This observation led to a change in the formula for response time of open requests.

$$RT_{i,j} = W_{i,j} + V_{i,j} \cdot S_{i,j} \cdot \left(1 + Q_{closed_j}\right) \ , W_{i,j} = \Sigma_{k \neq i} \lambda_k \cdot V_{k,j} \cdot S_{k,j}^2 \ , \tag{4.7}$$

where $W_{i,j}$ is the wait time for requests of type i at resource j, and λ_k is the arrival rate for each open request type.

The formula for wait time was deduced by the thesis author as a result of observations on simulated behavior and by applying probability theory. The proof starts from the interval between two requests of the same type, T_k, and assumes that the service time for a certain request type has to be lower than this interval, otherwise the resource would be saturated by a single request type and no analysis would further be needed. The probability that resource j is busy servicing a request of type k is given by the following formulae:

$$P_k = V_{k,j} \cdot \frac{S_{k,j}}{T_k} \ , T_k = \frac{1}{\lambda_k} \ . \tag{4.8}$$

The wait time value is the weighted sum of service times for requests that may be occupying resource j when a request of type i arrives, the weights being the previously mentioned probabilities. Only requests of other types are considered in the formula, since requests of the same type should have already been serviced by the time this request arrived.

$$W_k = \Sigma_{k \neq i} P_k \cdot S_{k,j} \ . \tag{4.9}$$

Thus, the wait time expression in formula (4.7) is derived from formulae (4.8) and (4.9).

The formula for response time in closed systems was also altered, by considering the effects of resource multiplicity. Closed Queueing Networks (CQNs) with multiple-server stations were studied in [117] and a correction factor, denoted by $Y_{i,j}$, was defined:

$$RT_{i,j} = F_j \cdot S_{i,j} \cdot (1 + Y_{i,j} \cdot Q_{i,j}) \ , Y_{i,j} = \frac{1}{C_j} \cdot U_{i,j}^{4.464 \cdot \left(C_j^{0.676} - 1\right)} \ , \tag{4.10}$$

where C_j is the multiplicity of resource j, and $U_{i,j}$ is the utilization of resource j by requests of type i.

4.3. Summary

This chapter introduces a novel hybrid analytical/simulation model, from which both a simulation submodel, and a Queueing Network model are derived in a straightforward manner. According to the classification in [108], the performance model belongs to class III: the analytical model of the entire system is solved using results from the simulation of a submodel.

The simulated submodel is delimited from the performance model by using a decomposition strategy adapted from the LQNS approach, in order to fit both hierarchical and sequential system models.

A new hybrid solving approach is defined, inserting simulation results into analytical formulae. The analytical solver is applied to a QN model of a sequential flow model or to layered hierarchical submodels. The hierarchical approach is based on the technique used in LQNS [48], but brings significant changes: the top-down pass is skipped, only the upwards pass is performed, after the simulation. The analytical solving step uses formulae from MVA extensions [77]. Two approximation techniques are considered: Bard-Schweitzer (Approximate-MVA) and Chandy-Neuse [25], the latter being an improved estimation. The formulae for solving open systems have been adapted by the thesis author to obtain better accuracy.

Enhanced flexibility is achieved for the solving method, by several extensions, compared to existing approaches. The input system model can include mixed requests (open and closed); LQNS and $UML - \Psi$ only support closed models. Both active and passive resources are modeled, and a resource multiplicity correction factor [117] is used to obtain more accurate results.

5. TRANSFORMATION OF UML MARTE MODELS TO THE HYBRID MODEL

5.1. UML 2.0 Diagrams and the MARTE Profile

From the multitude of ways to define a system by using UML 2.0 [92] diagrams, there are specific combinations that cover the definition of a wide range of distributed systems. Such system representations should also be meaningful for performance analysis.

The resources, both active and passive, are specified by deployment diagrams, and in order to add behavior to these resources, there are two possibilities:

- use case diagram detailed with activity diagrams,
- sequence diagram.

Activity and sequence diagrams are chosen, as behavioral diagrams, because they are already widely adopted by the modeling community and they are suitable for distributed systems; in case of sequence diagrams, use cases are no longer needed, since the workload is already included in the diagram. Other behavior related diagrams, such as state machine or timing diagrams, are useful only to provide details of particular event-based systems. Communication diagrams (formerly called collaboration diagrams) do not show message sequencing clearly, focusing on objects and interactions between adjacent objects.

In the following sections, for each of these sets of diagrams, the process of extracting the performance model will be described, as presented by the thesis author in [28, 29].

Most of the performance annotations are compliant to the MARTE Profile, briefly reviewed in appendix B. Since there are annotations that need to be further refined, the profile is still subject to changes; the beta version specification is available at [89]. Workload distributions are poorly supported: several popular types are missing, as already reported by the community. For this reason, the workload distributions needed as inputs to the proposed hybrid analysis method will rely on SPT annotations [91].

5.2. Transformation of Deployment Diagrams

The resources that are important for performance analysis should be stereotyped as GaExecHost (GQAM), SchedulableResource (GRM), or just Resource (GRM) / PaLogicalResource (PAM) for passive resources. An example of a deployment diagram is presented in figure 5.1.

The performance model will include two objects of type ActiveResource: LDAPService and dirAuthService. These entities representing software applications (services) run on specific platforms (GaExecHost), which may be considered resources too, but they will only be used by the previously mentioned active resources (services). In this two-layer deployment, the two future active resources are stereotyped as both SchedulableResource and Artifact. The

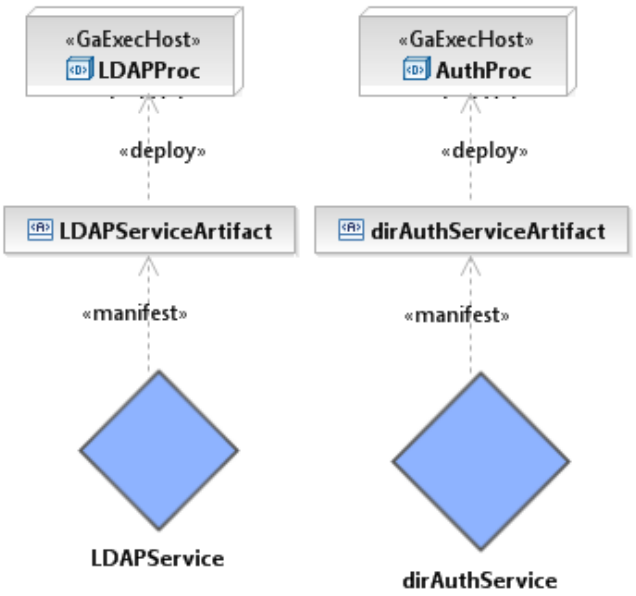

Figure 5.1: Deployment diagram with two layers

SchedulableResource stereotype is especially important when using sequence diagrams for behavior specification, since references from lifelines are towards objects having this stereotype.

In order to maintain compatibility with models defined in SPT, simplified deployment diagrams, such as the one in figure 5.2, are also considered for transformation.

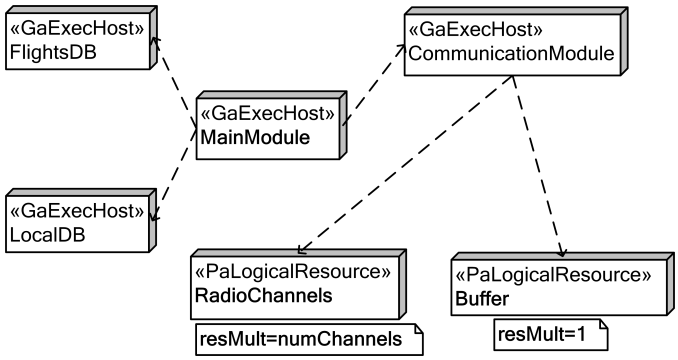

Figure 5.2: Deployment diagram with one layer

In this case, each object stereotyped as GaExecHost will become an ActiveResource, and each object stereotyped as PaLogicalResource will become a PassiveResource. The multiplicity can be specified using the "resMult" tagged value, as shown in the figure.

5.3. Transformation of Use Case and Activity Diagrams

A use case diagram connects actors to use cases (behavior). An actor stereotyped GaWorkloadEvent will become a Workload (open or closed) in the performance model, while a use case will become a Scenario (a composite step).

The scenario can be defined either by an activity diagram, or a state machine diagram. Each Activity/State stereotyped with PaStep will become a Step, while each Control Flow/Transition will become a Transition in the performance model. Fork and Join nodes are treated as pseudo-actions. Branch and Merge nodes need not be considered as distinct steps, because multiple outgoing or incoming transitions may be defined for each step, and the probability, in case of Branch, is included in the Transition object.

The ActiveResource for a step is given by tagged values of the PaStep stereotype. In case of single-layered deployment, the "host" property indicates the GaExecHost resource, while for double-layered deployment the "concurRes" property refers to a SchedulableResource.

An example of annotated use case and activity diagrams is presented in figure 5.3.

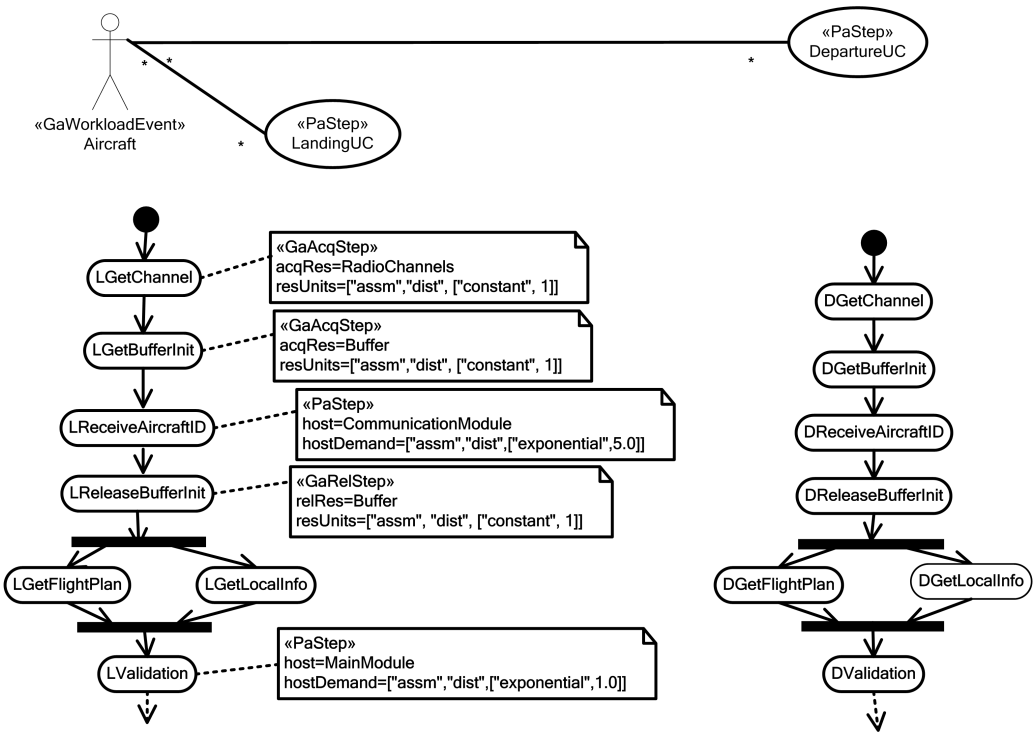

Figure 5.3: Annotated use case and activity diagrams

Each use case, stereotyped as PaStep, is detailed by a corresponding activity diagram. The component steps of a scenario are stereotyped according to the performed action. The stereotypes GaAcqStep / GaRelStep are used for passive resource acquire / release operations, allowing specification of the required resource ("acqRes") and resource count ("resUnits"). PaStep denotes steps executed on active resources ("host"), from which a certain duration

is required ("hostDemand"). The example refers to a system specified by a single-layered deployment diagram, hence the usage of the "host" tagged value.

5.4. Transformation of Sequence Diagrams

A sequence diagram is a scenario itself, because it provides successive calls (steps) among instances of resources.

UML 2.0 changed sequence diagrams significantly, the expressiveness of the language was highly increased; the semantics are explained in [80].

A lifeline belongs to an item stereotyped as PaRunTInstance, having a tagged value called "instance" that points to a SchedulableResource (in case of double-layered deployment), and also a "host" property that specifies a GaExecHost resource (for single-layered deployment). Each message is attached to an operation stereotyped as PaStep executed on the resource of the target lifeline. The workload is usually defined by the first message in the sequence, stereotyped as GaWorkloadEvent.

An example of a sequence diagram designed in RSA is presented in figure 5.4.

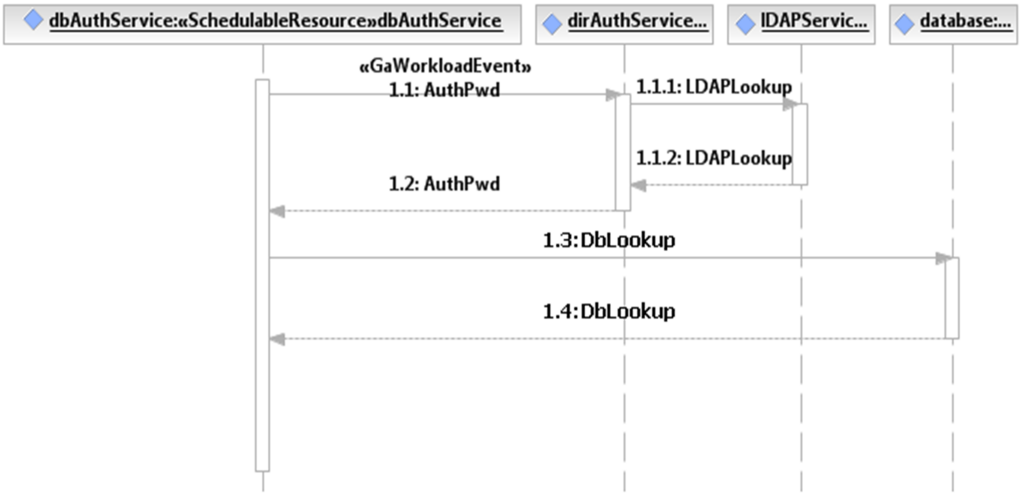

Figure 5.4: Annotated sequence diagram

A message is connected to an operation through the "SendEvent" property, as shown in figure 5.5. For each resource the list of operations should be defined and then they should be assigned to messages. When editing the "Operation" sub-property of a message (using the Browse button), the model tree is displayed, so the appropriate operation is selected.

In order to illustrate the way steps are annotated in RSA, several windows are displayed in figure 5.6. The PaStep stereotype is applied to an operation of a SchedulableResource object, operation that can be accessed from the Project Explorer. After selecting an existing operation of a resource, in the Properties view at least the "concurRes" (or "host" in case of single-layered deployment) and "hostDemand" values should be set. For "concurRes" the

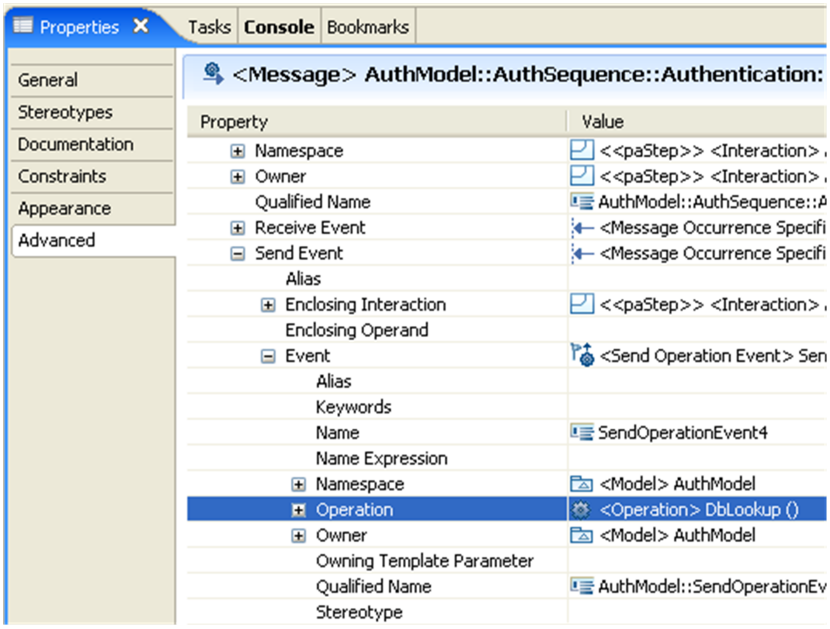

Figure 5.5: Connection between messages and operations in sequence diagrams: SentEvent attribute

value is chosen from the list of available SchedulableResource items. The "hostDemand" attribute is edited using the windows shown in the top right corner of figure 5.6.

In the example, variables are used instead of constant values, since the tool that implements the proposed hybrid approach allows definition of variables in a separate configuration file, specified as input additionally to the system model. More details on the tool are presented in the next chapter: chapter 6.

5.5. Summary

Applying the hybrid proposed approach (chapter 4) does not require learning a new specific modeling language, as other approaches do. It relies on the widely adopted UML notation, with extensions for performance analysis: the MARTE Profile (or SPT Profile).

The complete performance model is obtained by applying the mapping rules presented in this chapter to diagrams. The guidelines are summarized in Table 5.1, as previously shown in [28, 29]; stereotypes are mapped, without details on tagged values, for brevity purposes.

In conclusion, the proposed approach is flexible in two ways:

- Several sets of input diagrams are supported: behavior can be described either by sequence or by activity diagrams;
- Performance annotations from both the most recent MARTE profile and the previous SPT profile are considered.

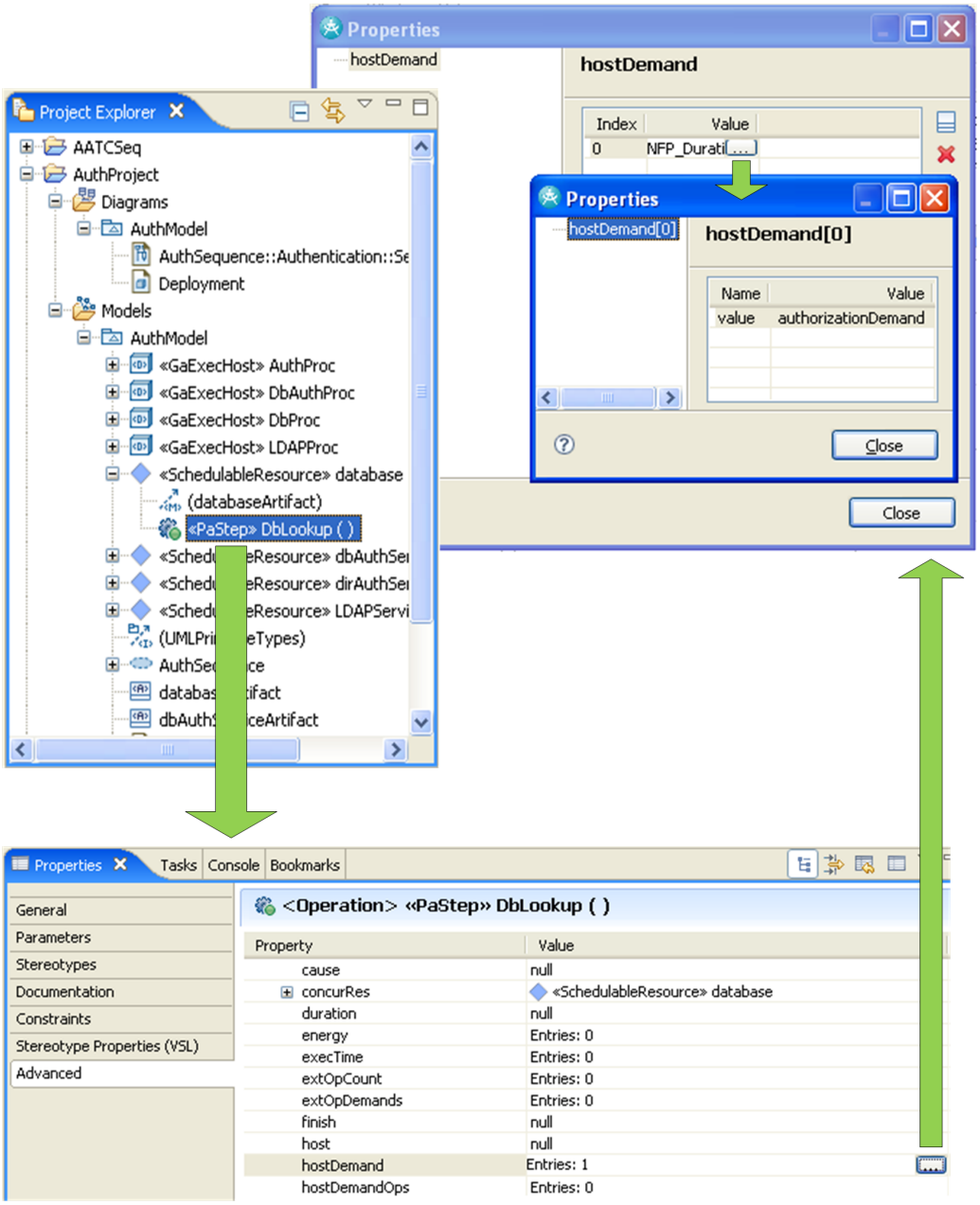

Figure 5.6: Annotations for steps in sequence diagrams

Table 5.1: Transformation rules from UML diagrams to the performance model

Diagram Type	UML Element Stereotype	Performance Model Element
Deployment Diagram	GaExecHost	ActiveResource
	SchedulableResource	ActiveResource
	Resource	PassiveResource
	PaLogicalResource	PassiveResource
Use Case Diagram	WorkloadEvent	Workload(Open or Closed)
	GaScenario	Scenario
Activity / State Machine Diagram	PaStep	Step
	ControlFlow or Transition	Transition
Sequence Diagram	WorkloadEvent	Workload(Open or Closed)
	PaStep	Step

6. PERFORMANCE HYBRID MODEL SOLVER AND SIMULATOR

6.1. Tool Overview

PHYMSS (Performance Hybrid Model Solver and Simulator) [30] is a tool that has been developed by the thesis author and intends to encompass as much as possible from the performance analysis process and provide flexible analysis options. It implements two performance analysis techniques: a simulation approach and the proposed hybrid method.

The tool accepts XMI files with the UML representation of the system model, annotated using the MARTE Profile. Both a simulator and a hybrid solver are available for performance analysis. The hybrid approach is based on a simulation model that can be treated as a LQN model during the high-level analytical solving process. Performance results are inserted into the UML model and can be exported as an XMI file. The tool is developed in C#, using Microsoft .NET Framework 3.5. The block diagram of the system is presented in figure 6.1.

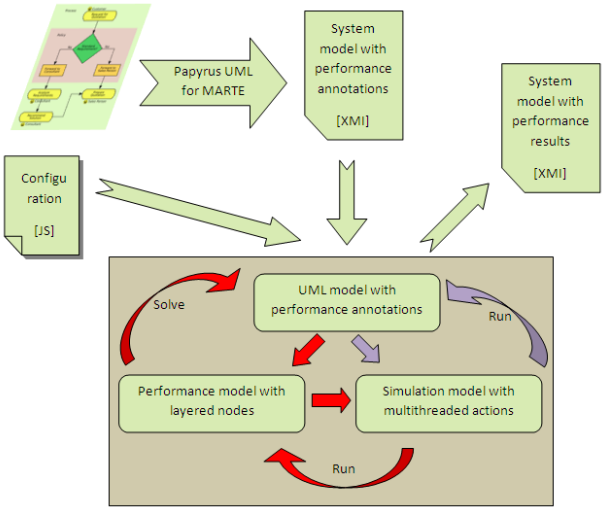

Figure 6.1: PHYMSS block diagram [30]

Tool input is represented in XMI format and can be obtained as output from visual design editors for UML diagrams. Papyrus UML [5] is such an editor; it is open-source and supports extensions for the MARTE profile. However, Papyrus UML doesn't fully support UML 2.0 sequence diagrams, so Rational Software Architect (RSA) is an alternative for defining such diagrams; MARTE annotations for RSA are supported by means of a plugin [3].

PHYMSS supports two types of input model specification:

- Deployment diagrams for resources, and sequence diagrams for behavior (including both clients and scenarios).
- Deployment diagrams for resources, and use case diagrams for scenarios and client behavior with scenario details presented in activity diagrams.

Simulation and analytical solving parameters, such as duration, confidence interval relative width for the convergence test or iteration count are specified in a JavaScript configuration file; this language has been chosen in order to be easily adopted by users and interpreted by the application. This configuration file is also useful in order to parameterize the system model description: variables, such as arrival pattern, can be left unassigned inside the XMI file, and their values specified in the configuration file. Hence, the effects of different values for system parameters on system performance can be evaluated without changing the UML model, only the configuration file needs to be modified.

Inside the system, which is illustrated as a brown box, the UML model is stored with all performance annotations. This is where performance results are stored too, after applying simulation or the hybrid approach, as shown by the two highlighted alternative paths. The hybrid approach requires building the performance model, based on the hybrid meta-model presented in section 4.1.1: the hybrid model instance creates a simulation submodel that is run. The simulation results are then propagated upwards by solving the submodels in the higher levels of the performance model. The pure simulator builds the simulation model from the UML model and executes it, inserting the results into the UML model as statistics while the simulation runs.

After having applied one of the two approaches, performance analysis results can be exported into an XMI file, with the same structure as the input file: each UML element will have values specified for parameters such as response time, throughput or utilization.

6.2. Performance Prediction Process with PHYMSS

In order to highlight the ease of analysis, the steps of the prediction process when using PHYMSS are shown in figure 6.2.

Most of the process is automated; the user will provide annotated system models and will decide whether improvements are needed, after viewing the performance analysis results.

The tool can be used in two main types of situations:

- An existing distributed application (legacy system) needs to be optimized and several re-design alternatives are analyzed in order to select the best of them for implementation.
- A new system is designed, and it has important performance requirements that need to be ensured, so performance analysis is used to study the feasibility of the system before spending time on implementation.

In the first case, UML diagrams are extracted by reverse engineering from the existing system and are edited to obtain re-design alternatives, in the second case they are created directly in a UML visual diagram editor.

The diagrams are annotated with arrival rate information and service time distributions: they are estimations in case of a new system and measurements in case of a re-design

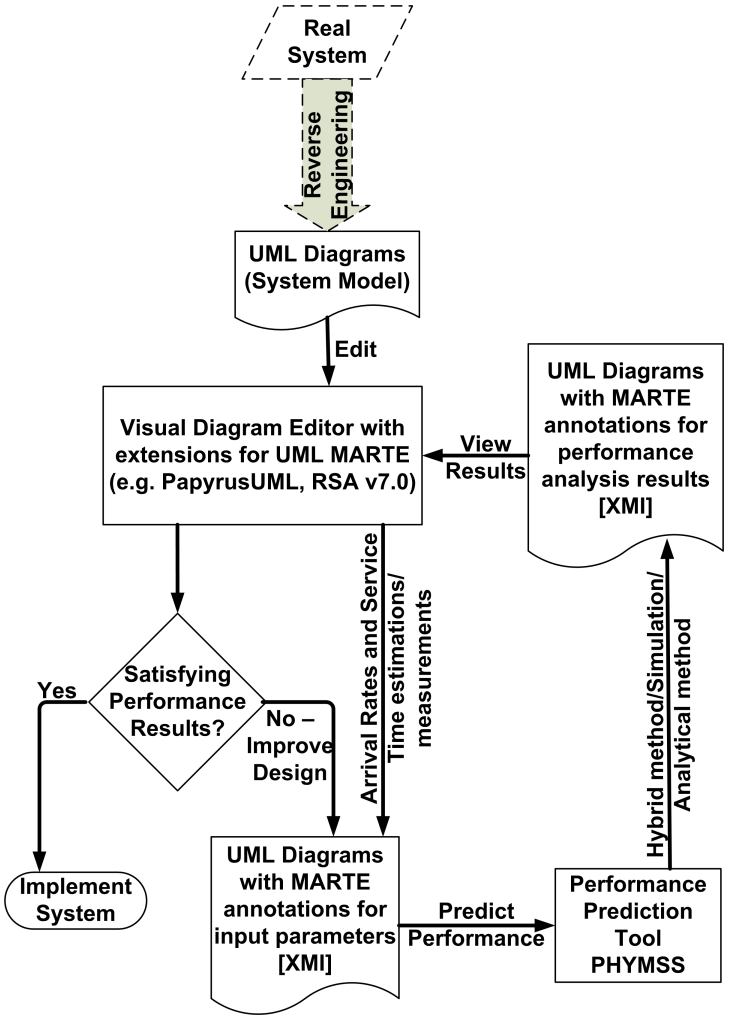

Figure 6.2: Role of PHYMSS in the prediction process [31]

operation. By arrival rate information, several parameters are referred to, depending on the type of system workload:

- open workload: arrival rate (in fact, distribution of inter-arrival time is used, to avoid fluctuations, as explained in the following section);
- closed workload: population count and think time distribution.

The annotated diagrams exported in XMI format by the visual editor are inputs for PHYMSS which applies one of the three available analysis methods (Pure simulation, hybrid analysis, analytical solving) and provides feedback into the diagrams: mean values for performance results inserted as MARTE annotations in the original diagrams (they can be saved as a different XMI file).

The output diagrams can be visualized using UML diagram editors and if the performance results are not satisfactory, the diagrams are improved and the analysis process is repeated.

6.3. Implemented Performance Analysis Methods

PHYMSS implements the proposed hybrid approach, and a multi-threaded simulator in order to validate the hybrid method results by comparison. Pure analytical solving is possible when selecting the maximum value for the level where the simulation would begin (the simulation will be bypassed).

The component packages, each implementing a specific methodology and their dependencies are illustrated in figure 6.3.

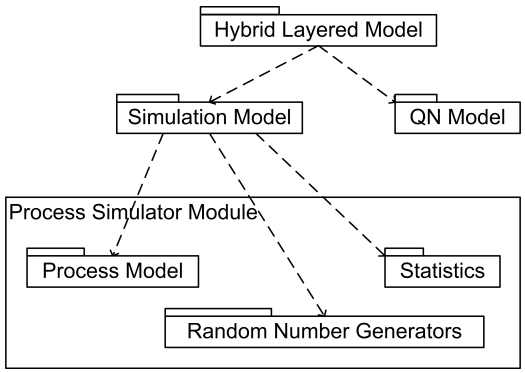

Figure 6.3: PHYMSS components

The pure simulator is based on the simulation model defined by Marzolla in $UML - \Psi$ [74]. The implementation in PHYMSS by the author does not rely on single-threaded coroutines, as in the original approach, but is improved by using thread pools, and thus allowing for multiple threads to be run simultaneously.

The parser for the input model could not be reused since the $UML - \Psi$ relied on UML SPT diagrams and a configuration file written in Perl. It has been implemented by

the thesis author considering UML diagrams with MARTE performance annotations (or SPT annotations where needed) and JavaScript syntax for the configuration file.

The simulation model of $UML - \Psi$ was subject to several extensions, in order to obtain more flexibility.

Firstly, the input model range includes open models, additionally to closed models supported in the original implementation. In order to achieve this, the open workload was implemented and used in an efficient way from the simulation point of view by specifying its inter-arrival time instead of the usual arrival rate. The motivation is simple: when specifying arrival rates (frequency), they have to be inverted for the simulator to work with durations and this leads to large fluctuations when distributions are given for arrival rates. Instead, it is better to use the distribution of inter-arrival time directly, to avoid unwanted fluctuations due to intermediate computations.

Secondly, the simulation model was improved to cover nested calls inside scenarios, since the original model only supported transitions. The simulation engine was adapted so that each step is able to perform both nested calls and transitions to other steps during the simulation.

The analytical approach presented in [30] uses a QN model and a MVA (layered) solver. It was improved by changing the granularity level, using the Chandy-Neuse estimation and resource multiplicity correction factor, extending the input model range to mixed requests and modeling of passive resources, as shown in [31].

The Process Simulator Module, developed by Claudiu Rad as Bachelor's Degree Project [103] under the supervision of the thesis author, is based on the *libcppsim* library [72] and provides three types of functionality to the simulation model, as shown in figure 6.3:

- Process Model: the simulator is a process oriented approach, and relies on coroutines; the current implementation replaces single-threaded coroutines with a multi-threaded approach, based on the C# *ThreadPool* [103].
- Random Number Generators: used to simulate request arrivals according to workload inter-arrival time distributions.
- Statistics: compute mean values and variance for different types of data (performance parameters) collected during simulation.

An additional property was implemented by the thesis author for the random number generators: mean value. This is needed in order to ensure reproducible results for the analytical solver. For exponential and constant distributions, the mean value was already available as an input parameter. Uniform distributions (both discrete and continuous) have an intuitive formula for the mean value: $\frac{min+max}{2}$. In case of Gamma and Erlang distributions, the mean value is computed from input parameters:

$$mean_{Gamma} = k \cdot \theta, mean_{Erlang} = \frac{k}{\lambda} \tag{6.1}$$

where k is the shape (Gamma) or integer (Erlang), θ is the scale, and $\lambda = \frac{1}{\theta}$ is the rate.

For Weibull distributions, defined by the shape k and the scale λ, the mean value is

computed using the Gamma function, Γ, as shown in (6.2).

$$mean_{Weibull} = \lambda \cdot \Gamma\left(1 + \frac{1}{k}\right), \tag{6.2}$$

A numeric approximation for the Gamma function was found in [82], for $z \in \mathbb{R}, z \geq 1$:

$$\Gamma(z) \approx \sqrt{\frac{2\pi}{z}} \cdot \left(\frac{z}{e}\right)^z \cdot \left(1 + \frac{1}{15z^2}\right)^{\frac{5}{4}z}, \tag{6.3}$$

For the normal distribution, the mean value is given in the distribution definition. However, for truncated normal distributions, the mean value needs to be deduced by using integral calculus and variable substitution. For brevity, only the results of the calculus are presented in (6.4, 6.5, 6.6).

$$mean_{LRT} = S^2 \cdot \frac{f(K_1) - f(K_2)}{\Phi(K_2) - \Phi(K_1)} + M, \tag{6.4}$$

$$mean_{LT} = S^2 \cdot \frac{f(K_1)}{1 - \Phi(K_1)} + M, \tag{6.5}$$

$$mean_{RT} = -S^2 \cdot \frac{f(K_2)}{\Phi(K_2)} + M, \tag{6.6}$$

where LRT means truncated at both ends, LT means left truncated, and RT means right truncated, K_1 is the left bound, K_2 is the right bound, M and S are the input mean and variance respectively.

Characteristics of the LT Normal Distribution are presented in [61]. RT and LRT formulae were deduced accordingly.

Functions f and Φ are the probability distribution function and cumulative distribution function and have the following formulae:

$$f(x) = \frac{1}{\sqrt{2\pi S^2}} \cdot e^{-\frac{(x-M)^2}{2S^2}}, \tag{6.7}$$

$$\Phi(k) = \frac{1}{2}\left[1 + \text{erf}\left(\frac{k}{\sqrt{2}}\right)\right], \tag{6.8}$$

where $erf(x)$ is the error function defined by 6.9 and approximated by 6.10, as shown in [124].

$$\text{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-x^2}\, dx \tag{6.9}$$

$$\text{erf}(x) \approx \left[1 - \exp\left(-x^2 \frac{\frac{4}{\pi} + ax^2}{1 + ax^2}\right)\right]^{\frac{1}{2}}, \tag{6.10}$$

where the constant a has the following formula:

$$a = \frac{8}{3\pi} \cdot \frac{\pi - 3}{4 - \pi}. \tag{6.11}$$

6.4. User's Guide

The Graphical User Interface (GUI) is designed to be user-friendly: Microsoft's Windows Presentation Foundation (WPF) [79] was used to provide a better user experience.

PHYMSS provides two interaction patterns:

- menu with submenu options, as shown in figure 6.4;
- toolbar with buttons (intuitive symbols) for each available option; when hovering over a button, an explicit tooltip is also displayed (see figure 6.5.

The options (toolbar items) are enabled only when they can actually be used. Hence, the order in which the steps should be performed is obvious by restricting the set of allowed options.

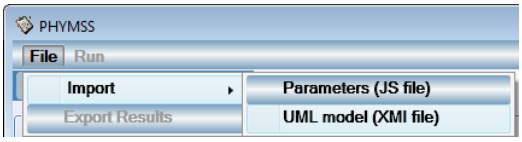

Figure 6.4: PHYMSS menu

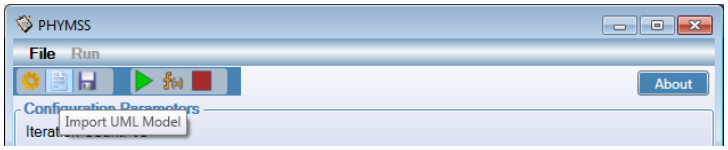

Figure 6.5: PHYMSS toolbar

The options are grouped in two main categories:

- File
  - Import parameters (JS file)
  - Import UML model (XMI file)
  - Export results (XMI file)
- Run
  - Run Simulation
  - Solve (Hybrid or pure analytical method)
  - Cancel

The initial operation flow is presented in figure 6.6, where each operation is preceded by its symbol on the application toolbar. At application startup, the only available option is Import Parameters; after importing the JS configuration file, the Import UML Model button is enabled. After importing the model, the Run Simulation and Solve buttons are active.

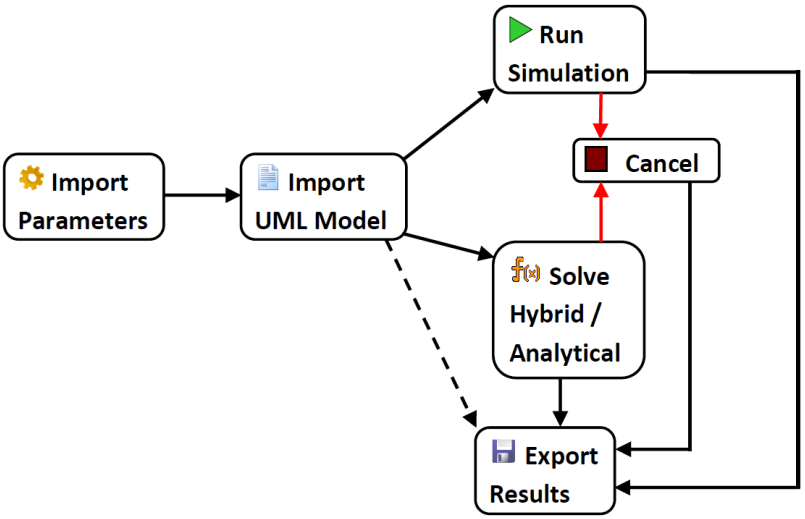

Figure 6.6: PHYMSS operation flow

During simulation or hybrid analysis, the only active button is Cancel. When clicking the Cancel button, the process stops after the current iteration finishes, in order for the algorithm to reach a stable state.

After normal termination of simulation or hybrid solving or after cancelling one of these operations, all operations are active except Cancel (which is active only during simulation or solving). The Export button can be used even before the analysis, but there are no performance results, so it would only create a copy of the input XMI model; that's why the arrow from Import UML Model to Export Results is dashed: this is not a useful transition.

The main window is divided in two panels:

- Configuration Parameters
  - Imported parameters are displayed (read-only)
  - Simulation or hybrid analysis settings can be set/reset
- Simulation Status
  - Simulated entities to be monitored can be selected
  - For each monitored entity, an embedded panel is displayed with statistical information on performance parameters (min, max, average and confidence interval).

The only simulation related setting available in the Configuration Parameters panel is the Show Simulation Progress check box: unchecking it leads to an empty Simulation Status panel.

The solver parameters are editable after clicking the Solve button (or Hybrid Method option from the Run menu):

- Show Simulation Progress check box refers to the submodel that is simulated

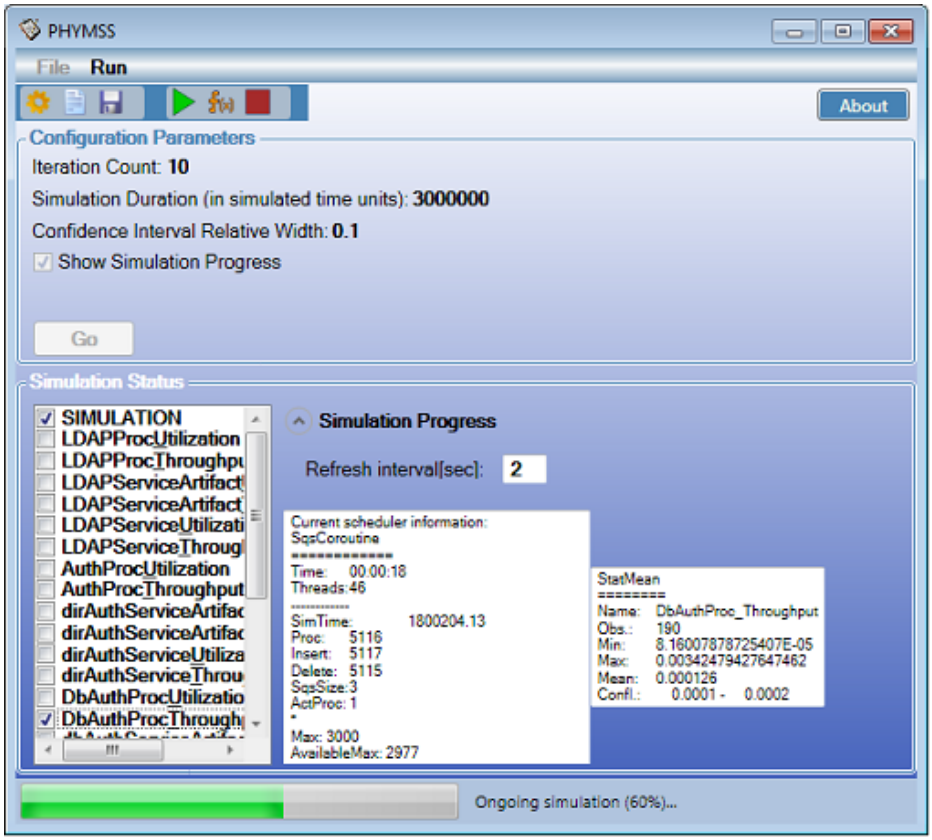

Figure 6.7: PHYMSS Simulation Status Panel

- Simulation Start Level allows choosing the *simulation level* as defined in chapter 4; the range starts from 2 (in case of input sequence diagrams) or 3 (for use case and activity diagrams, since the use cases insert an additional unused layer)
- Enhanced Accuracy check box selects the analytical approximation: A-MVA if unchecked [104] and Chandy-Neuse if checked [25].

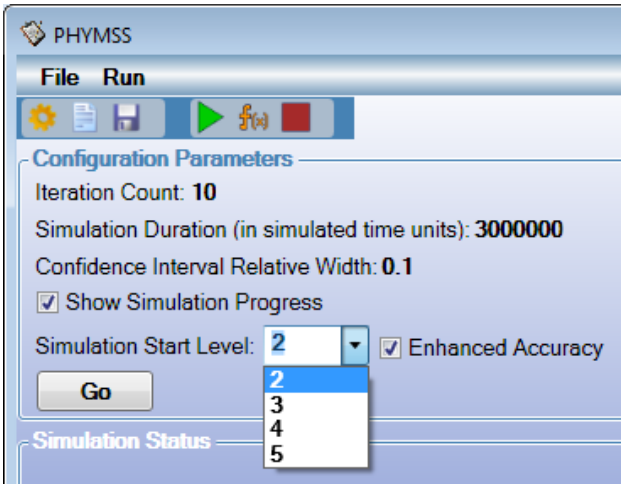

Figure 6.8: PHYMSS Configuration Parameters Panel

When all parameters are set, the Go button starts the simulation / solver and the Configuration Parameters becomes read-only during the analysis; the only active button or option from the menu is Cancel. After the simulation / solving ends or is cancelled by the user, the Go button transforms into Clear All: the data resulting from the analysis (performance parameters) must be reset before starting a new analysis. Of course, the results should be first exported (if needed) and afterwards reset.

6.5. Summary

PHYMSS and the methodology it implements have been considered relevant and with potential by experts in the field of SPE: the paper describing PHYMSS [30] was published at the 1st Joint WOSP/SIPEW International Conference on Performance Engineering.

The application provides two performance analysis approaches:

- Simulator: multi-threaded version of *UML* – Ψ, extended to support both open and closed workloads, and also nested calls, in addition to transitions inside scenarios.
- Hybrid Solver: novel approach combining layered (where possible) analytical solving with simulation of a submodel from a certain level downwards, level determined hierarchically, or sequentially when there are too few layers. The pure analytical approach can be obtained by skipping the simulation step.

Input XMI files are supported, a specific parser has been implemented to interpret both MARTE and SPT annotations, where MARTE annotations are not defined yet. Two sets of diagrams can be used to describe the input model:

- Deployment diagrams for resources, and sequence diagrams for behavior (including both clients and scenarios)
- Deployment diagrams for resources, and use case diagrams for scenarios and client behavior with scenario details presented in activity diagrams

The GUI is user-friendly, the imported configuration is displayed and several options are available at run-time:

- Hybrid solver:
  - Level k where simulation starts is customizable
  - The analytical solver allows two approximations: A-MVA [104] and a better estimation technique, Chandy-Neuse [25].
- Simulation: intermediate performance results can be viewed during the simulation in a dedicated panel.

7. CASE STUDIES

7.1. Input Models

The proposed methodology is flexible, so that the implementation in PHYMSS accepts open, closed or mixed system models as input models. Observing analysis results on a single input model does not suffice for a thorough validation. Two kinds of systems have been considered for modeling as application inputs:

- Open systems:
  - Help Desk, described in [77] as an application of analytical methods.
  - Authentication System, based on a real online system, the model being built by the author of this thesis and described in [31].
- Closed system:
  - Airport Air-Traffic Control (AATC), whose model was built by the author of this thesis in [27, 26], in order to study the benefits of simulation approaches in performance analysis.

Each of the above mentioned models will be described in the following sections.

Results are obtained by using PHYMSS 1.7, installed and run on Windows 7, the computer system having an Intel Core 2 DUO CPU, at 2.93 GHz and with 4 GB RAM.

7.2. Validation of Implementation and Improvements for Pure Analysis Methods

7.2.1 Help Desk System

The open system is presented in detail in [77]: a help desk application that would provide assistance to employees of a large company in solving problems related to their computing environments. Three main functions are defined:

- access to a database of Frequently Asked Questions (FAQ);
- creation of a help ticket;
- viewing of open help tickets.

Each scenario is described in [77] using database language (SQL-like) pseudo-code and then database management specific formulae and empirical studies are presented in order to derive the global demand of each use case for each resource. A programming-oriented pseudo-code of activity flow within scenarios is shown in figure 7.1.

```
FAQQuery
FOR i = 1..KeywordsPerQuery
  SELECT FROM KeywordT WHERE Keyword; //get keyword ID for given keyword
  SELECT FROM KeywordQuestionT WHERE KeywordID; //get question IDs for
keyword ID
  FOR j = 1..QuestionsPerKeyword
    SELECT FROM QuestionT WHERE QuestionID; //get questions with given IDs

NewTicket
SELECT FROM EmployeeT WHERE EmployeeID; //check if employee with given ID is
valid
IF ProbValidEmployee THEN
  UPDATE TicketT NUM_ROWS = 1; //add 1 row to the table with tickets
  UPDATE TicketEmployeeT NUM_ROWS = 1; //create association to employee
  UPDATE TicketKeywordT NUM_ROWS = KeywordsPerTicket; //link ticket to
keywords

ViewStatus
SELECT FROM TicketEmployeeT WHERE EmployeeID; //get ticket IDs for employee
FOR i=1..TicketsPerEmployee
  SELECT FROM TicketT WHERE TicketID, Status; // get open tickets
```

Figure 7.1: Help Desk System model: pseudo-code for scenarios

The pseudo-code is made of SELECT and UPDATE operations on the database tables, tables referred to using identifiers with the âĂIJTâĂİ suffix. The database model is described in detail in [77]: tables, indexes and relationships between tables. The tables mapped to entities are: QuestionT, KeywordT, EmployeeT, and TicketT; the relationships between entities are modeled as tables with pairs of IDs: KeywordQuestionT, TicketEmployeeT, and TicketKeywordT. The identifiers in italics are parameters that were deduced from system specifications: *KeywordsPerQuery* $= 2$, *QuestionsPerKeyword* $= 20$, *ProbValidEmployee* $= 0.9$, *KeywordsPerTicket* $= 5$, *TicketsPerEmployee* $= 80.3$. Arrival rates and computed service demands for each scenario and service center (resource) are presented in Table 7.1. The case study assumes a bottleneck has been solved by multiplying a disk into 4 identical instances, designated by D_j.

Table 7.1: Input performance parameters for Help Desk System

	FAQ	New Ticket	Status View
Arrival Rate [tps]	1.92	0.41	0.27
CPU Demand [sec]	0.237	0.046	0.607
D_j Demand [sec]	0.316	0.0617	0.809

7.2.2 Simulation Model Validation

The simulator from the tool is an extension of *UML* $-$ Ψ [74], but instead of sequential coroutines, it relies on multithreaded servers. It will be validated first, in order to be able to consider it as reference for the validation of the hybrid approach.

Several techniques, used in model verification and validation, are described by Sargent in [106], methods that apply either to the conceptual model, or to the computerized one:

- Animation: operational behavior through time
- Comparison to other valid models: analytical models may be used for simple cases
- Degenerate tests: degeneracy of behavior for certain values of input and internal parameters
- Face validity: opinion of individuals
- Historical data validation: data collected on a real system
- Internal validity: variability in several runs of a stochastic model
- Operational graphics: values of parameters shown graphically as the model runs
- Parameter variability-sensitivity analysis: effect of variations in values of input and internal parameters on model behavior or output
- Predictive validation: compare predictions based on the model with real system behavior
- Traces: behavior of entities is traced to check the model logic and accuracy.

A conceptual model is turned into a computerized model in order to be run, by implementing it in a specific simulation dedicated language or a high-level programming language. The former would provide error free behavior and pseudo-random generators, while the latter needs more development and testing efforts. Verification of a computerized model versus the conceptual model is usually performed by structured walkthroughs (static testing) and traces (dynamic testing) [106].

In case of the simulation model in PHYMSS, traces were used in early validation to observe model behavior over simulated time. Variability was also monitored by computing variance for value ranges of each output parameter.

In order to present quantitative aspects of validation, the previously described input model was subject to simulation. The step by step behavior of the open model is presented in appendix C.

The resulting mean values are compared to simulation results (expressed as simulated time units [stu]) in Table 7.2. The comparison between these quantities is possible since the values of input parameters expressed as seconds ([sec]) and transactions per second ([tps]) were used as input values for the simulator. Therefore, in the following tables, the simulation results will be considered as follows: response time in [sec] and throughput in [tps].

For each simulation response time value, variance is specified between parentheses. Variance is computed differently for response time (7.1) and for other parameters (utilization, throughput) (7.2).

$$Variance\,(RT) = \frac{Interval.width}{MeanValue}, \tag{7.1}$$

where $Interval.width$ is the width of the interval including all measured values.

Table 7.2: Help Desk simulation results compared to computed mean values

Req. Type	CPU Queue Length	Disk Queue Length	CPU Wait Time [sec]	Disk Wait Time [sec]	CPU Resp. Time [sec]	Disk Resp. Time [sec]	CPU Resp. Time (Variance) [stu]	Disk Resp. Time (Variance) [stu]
T1	0.27	0.6	0.05	0.17	0.29	0.49	0.32 (0.030)	0.52 (0.027)
T2	0.67	1	0.09	0.26	0.14	0.33	0.19 (0.090)	0.38 (0.092)
T3	1.5	1.5	0.12	0.2	0.72	1.01	0.66 (0.099)	0.91 (0.097)

$$Variance\,(U, Thr) = \frac{\Sigma_i x_i^2 - \frac{(\Sigma_i x_i)^2}{n}}{n-1}, \tag{7.2}$$

where x_i are measured values and n is the total number of measurements.

The offset values between simulation results and step-by-step calculus are shown in Table 7.3. As it can be observed from both Tables 7.2 and 7.3, variance and offset values are lower than or equal to 0.1.

Table 7.3: Offset between simulation results and step-by-step calculus

CPU Resp. Time Offset [sec]	Disk Resp. Time Offset [sec]
0.03	0.03
0.05	0.05
0.06	0.1

7.2.3 Analytical Enhancements Validation

The improved analytical solver was validated by comparing results to simulation results used as reference values. In order to maintain the semantics of the model, use cases are not decomposed into steps, they are seen as atomic operations. When held by a use case, a resource should be blocked for the entire duration of that use case, not only a step duration. Otherwise, interleaved execution of steps from distinct use cases leads to a significantly altered waiting time distribution among steps, compared to the wait time at use case level. The previously described open system model has the following behavior: multiple resources accessed simultaneously by the steps in the three scenarios. When applying analytical formulae to the matrix of resource demands per type of input request, results are obtained in a straightforward manner.

Modeling of simultaneous multiple resource possession is hard to achieve, since the MARTE UML annotations allow defining only one host resource per step. Dividing one step into parts corresponding to each held resource would be an alternative, but then the interconnection of these sub-steps has to be decided. Connecting them in sequence, would insert dependencies that are not present in the analytical formulae, which treat the acquisition of each resource as an independent activity. The correct approach is to connect the sub-steps

in parallel, using a Fork-Join structure to replace the initial multiple-resource step. The configuration file with analysis settings and the model input parameters is shown in figure 7.2 and the model is presented in figure 7.3.

```
var iterationCount=1;
var simDuration=40000;
var confRelWidth=0.1;

var FAQRate=["unbounded", ["constant", 0.52]];
var NewTicketRate=["unbounded", ["constant", 2.44]];
var StatusViewRate=["unbounded", ["constant", 3.704]];

var KeywordsPerQuery=2;
var QuestionsPerKeyword=20;
var ProbValidEmployee=0.9;
var KeywordsPerTicket=5;
var TicketsPerEmployee=80.3;
```

Figure 7.2: Help Desk configuration file for analytical method validation

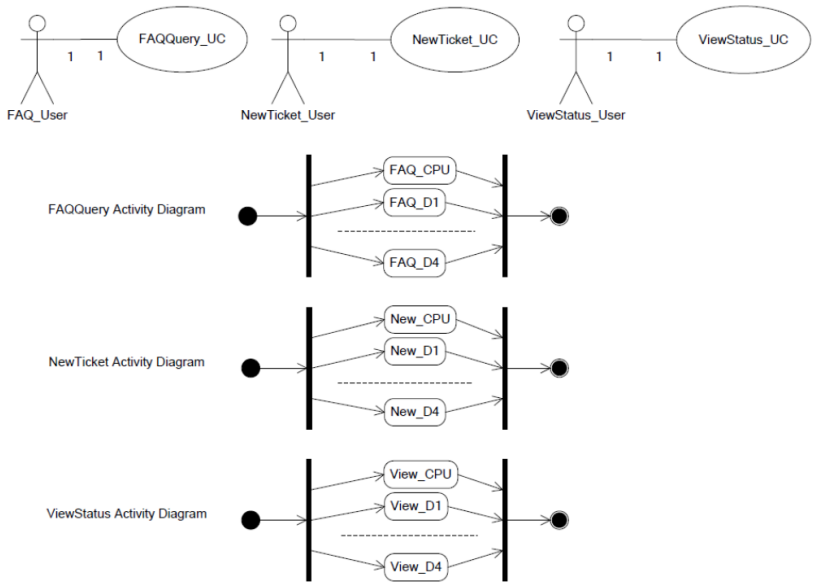

Figure 7.3: Help Desk one-step scenarios for analytical method validation

Results of analytical solving are compared to values obtained by simulation (reference values). The relative error [%] is computed as $|R_{analytical} - R_{simulation}|/R_{simulation} * 100$.

The relative error for most parameters is below 5%, and for a few it is around 10% which deems the analysis correct, as accepted in the field papers, such as [117]. The total response times for use cases could not be obtained directly by simulation because of the Fork-Join structure that returns the maximum time of the branches; the use case response times were computed by adding response times obtained by simulation for sub-steps corresponding to resources. For example, $R_{FAQ} = R_{FAQ\_CPU} + R_{FAQ\_D1} + R_{FAQ\_D2} + R_{FAQ\_D3} + R_{FAQ\_D4}$.

In order to compare response times at resource level, unit tests were implemented to have access to intermediate results, before exporting them to the output file; results are

Table 7.4: Analytical solver results validation

	Simulation	Analytical Solver	Offset	Relative Error [%]
CPU Utilization	**0.637**	0.64	0.003	0.47
D_j Utilization	**0.849**	0.85	0.001	0.12
$R_{FAQ}[sec]$	**2.4**	2.31	0.09	3.75
$R_{New}[sec]$	**1.78**	1.97	0.19	10.67
$R_{View}[sec]$	**4.3**	4.72	0.42	9.77
Mean Throughput [tps]	**2.59**	2.6	0.01	0.38

investigated and decomposed, in order to check the offset from the simulation results. The comparison is presented in Table 7.5.

Table 7.5: Analytical solver validation: detailed response time comparison [sec]

	Simulation	Analytical Solver	Offset
$R_{FAQ}[CPU]$	**0.31**	0.34	0.03
$R_{FAQ}[Dj]$	**0.52**	0.49	0.03
$R_{New}[CPU]$	**0.20**	0.25	0.05
$R_{New}[Dj]$	**0.39**	0.43	0.04
$R_{View}[CPU]$	**0.66**	0.71	0.05
$R_{View}[Dj]$	**0.91**	1	0.09

7.3. Real Distributed System for PHYMSS Validation

Hybrid solver validation will be performed by considering the model of a real system, in order to be able to perform measurements: a typical authentication system that allows for role impersonation in accessing a database.

7.3.1 System Model

UML deployment and sequence diagrams for this scenario have been defined, by reverse engineering. The sequence diagram depicting the authentication scenario is presented in figure 7.4. The diagrams were designed using RSA 7.0 extended with a plugin for MARTE [3].

The entry point is the database authentication service (dbAuthService) which performs authentication and authorization (AandA) of a given user account. The first step consists in password authentication (AuthPwd) delegated to the directory authentication service (dirAuthService) which relies on the central LDAPService to validate the user account in the domain. After the account is validated, the credentials are sent to the database server for authorization: checking access rights of the account (DbLookup). A user having a valid account might be granted access only to specific databases, based on predefined database roles. In case the corresponding role is found, the database will be accessed by impersonation,

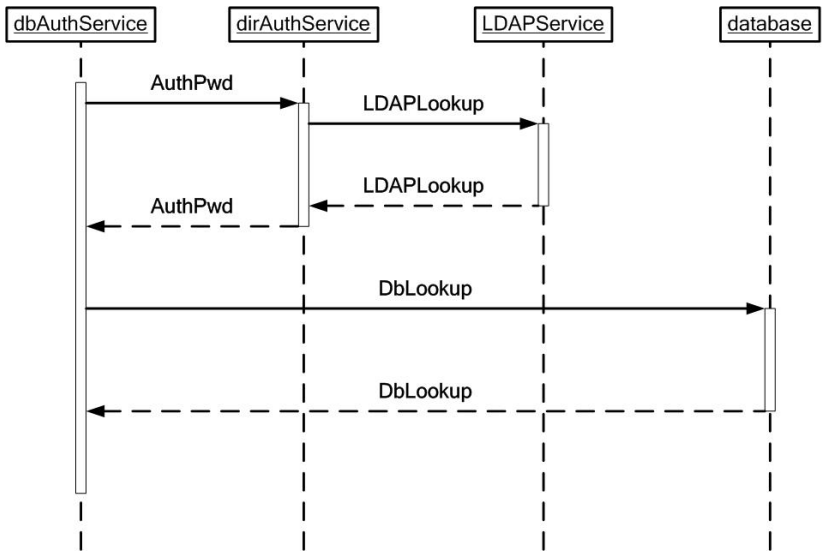

Figure 7.4: Authentication System sequence diagram [31]

using the role credentials. The above mentioned services are usually deployed on different machines.

Input model MARTE annotations are parameterized and the values are set in the configuration file. Arrival rates and service time distributions, have been derived based on measurements on the real online system, as shown in Table 7.6.

Table 7.6: Measured input parameters for the Authentication System

Input Parameter	Distribution Type	Distribution Parameters [msec]
Arrival rate	Exponential	$\lambda = 10000$
Authentication Demand	Normal	$\mu = 1290$ $\sigma = 4.31$
Authorization Demand	Constant	15.6

Samples have been collected during the busy hour of each day within one week (five working days) by using code instrumentation and log files, both on servers and client machines. Histograms have been built from measurements in order to deduce the distribution types (exponential, normal), and then specific parameters were extracted for each distribution (mean, variance). As an example, in order to find the median for service demands of the authentication step, several histograms have been built; each of these was subject to a filtering process (keeping only the relevant, centered, data) followed by building a finer-grained histogram for the remaining data. The final histogram is shown in figure 7.5. The configuration file used for the authentication case study in presented in figure 7.6.

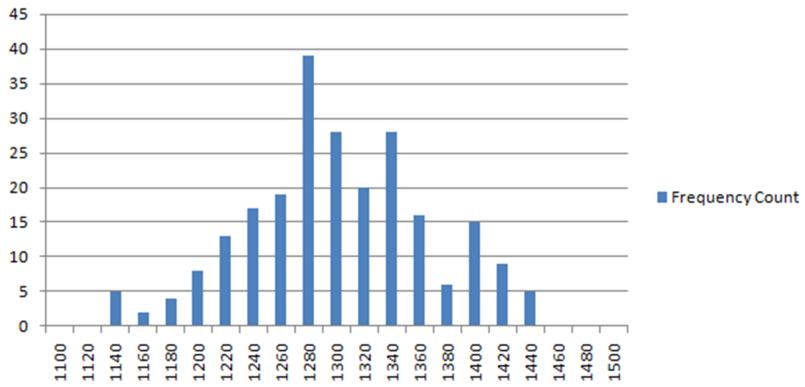

Figure 7.5: Authentication step demand (msec) distribution histogram

```
var reqArrivalPattern=["unbounded",
["exponential",10000.0]];
var authorizationDemand=["assm", "dist",
["constant",15.60]];
var ldapLookupDemand=["assm", "dist",
["normal", 1290, 4.31]];
var authPwdDemand=["assm", "dist",
["constant",0.0]];
var AandADemand=["assm", "dist",
["constant",0.0]];
var iterationCount=10;
var simDuration=3000000;
var confRelWidth=0.1;
```

Figure 7.6: Authentication System configuration file

7.3.2 Performance Results

The models were analyzed using three methods, for comparison: pure analytical, hybrid approach, and pure simulation. The results are shown in Table 7.7.

Table 7.7: Response time values comparison

Step	Analytical [msec]	H[4]	H[3]	Simulation	Measured [msec]
Authorization	15.6	15.6	15.6	15.6	1.36 (4.42)
LDAPLookup	1287.67	1391.54	1378.79	1363.36	
DirAuth	1287.67	1391.54	1378.79	1363.36	
AandA	1303.27	1407.14	1394.39	1378.96	1403.71 (344)
Duration	<1"	0'31"	1'55"	1'10"	-

Analysis duration is also presented in the table (in minutes and seconds) because there is no direct correspondence between simulated time and real time: the simulated time progresses several units at a time, between consecutive processes as defined by the scheduler.

For the hybrid approach, the level is specified between square brackets. Analytical solving is obtained by choosing the maximum available value for the *simulation level* (meaning no simulation is performed). For the measured values, the mean value is followed by the standard deviation between parentheses.

Response time mean values for steps are compared, and the analysis time is also presented. The discrepancy in case of the measured Authorization time comes from not modeling the caching at server side. The response times are either 15 or 0 msec, as it can be seen from the high standard deviation (4.42 msec) compared to the mean value (1.36 msec).

For $k = 3$, the simulation submodel includes a large part of the performance model, which leads to more accurate results, but also to a long analysis duration (because of the iterative process, which repeats the simulation and analytical solving at least twice).

For $k = 4$, the simulation submodel is smaller, leads to an improvement in speed (compared to pure simulation) and to precise results. The precision is not the same as for $k = 3$, but still more accurate compared to the pure analytical approach.

7.4. Improving System Design: Early Problem Detection by Performance Analysis

A case study has been built and several design alternatives are evaluated, in order to test the ability of PHYMSS to detect design flaws. The Airport Air Traffic Control (AATC) system manages incoming and outgoing flights for a certain airport. For simplicity, the aircraft is viewed as an external autonomous entity, interacting with the airport software system. The AATC system will be modeled with closed requests and activity diagrams, leading to a sequential layering, instead of the hierarchical approach applied previously.

Several improvements of both the analytical and the hybrid solver can be observed in the context of this case study: extending the input model range (by using both closed and open models as inputs), modeling of passive resources and resource multiplicity, and the improvements of Chandy-Neuse compared to A-MVA estimations for closed models.

7.4.1 Initial Design

The software system includes a MainModule which performs computations, validations and coordinates the activities. The database is split in two sections, in order to facilitate access: LocalDB, with airport local information, such as weather conditions and runways' status; FlightsDB, with flight plans (FP) of the aircrafts that are about to arrive; the flight plan should be sent by the preceding control center to the airport before the aircraft enters the airport airspace (usually when the aircraft passes by the control center).

The CommunicationModule is used for radio transmissions to and from the aircrafts flying in the airport airspace, or even on the ground, before departure. It uses a finite set of RadioChannels (each flight must be allocated a different channel, so that collisions do not occur, since consequences could be devastating). Another resource of the CommunicationModule is a Buffer, used to store incoming or outgoing messages, until they can be processed or sent respectively.

Each scenario in the use case diagram is detailed by an activity diagram, involving queries on the databases, computations and communication with aircrafts by using radio channels. Use case, deployment and activity diagrams for landing and departure are presented in figure 7.7. The UML model was defined using Papyrus UML [5] with its extension for MARTE, in order to have access to performance annotations and specific stereotypes (such as PaStep and GaScenario). In the diagrams, notes were attached to a few relevant activities, in order to illustrate the tagged values used for performance annotations; also, levels assigned to each entity when the model is built are displayed between square brackets, after each entity name.

Landing is guided from the control tower at the airport; for each aircraft, a radio channel is allocated on demand and released after the necessary information has been communicated. The Buffer associated to the CommunicationModule is acquired before sending or receiving a message and released afterwards.

The aircraft first sends its ID to the AATC system; it is assumed that the flight plan (FP) of the aircraft had been previously received from the last control center reached by it and the airport monitoring system stored the FP in FlightsDB. Thus, the system can retrieve information about the aircraft, based on its ID. AATC also needs to check the weather conditions and if there are any free runways, by consulting LocalDB, in order to decide whether to allow the aircraft to land or not. The two databases, LocalDB and FlightsDB, are queried in parallel, to save time. The decision on landing permission is performed by the LValidation step in the diagram and is followed by one of the three alternative branches; PHYMSS allocates default equal probabilities to them when no values are provided as annotations in the input model (0.33 for each branch, in this case).

In case of landing approval, a runway is selected and landing parameters are computed (direction, descent rate) for the aircraft to follow the correct trajectory to the runway. These parameters are sent to the aircraft using the allocated radio channel, which is released after this operation.

Another equiprobable option is for the aircraft to wait, in a queue, flying over the airport; it will repeatedly ask for permission to land, until it is approved.

The third option is that the landing is denied, meaning no runway is free and also the waiting queue is full, so the aircraft will be redirected to another airport; the FP is updated accordingly and sent back to the aircraft and to the next control center, on the route to the new destination.

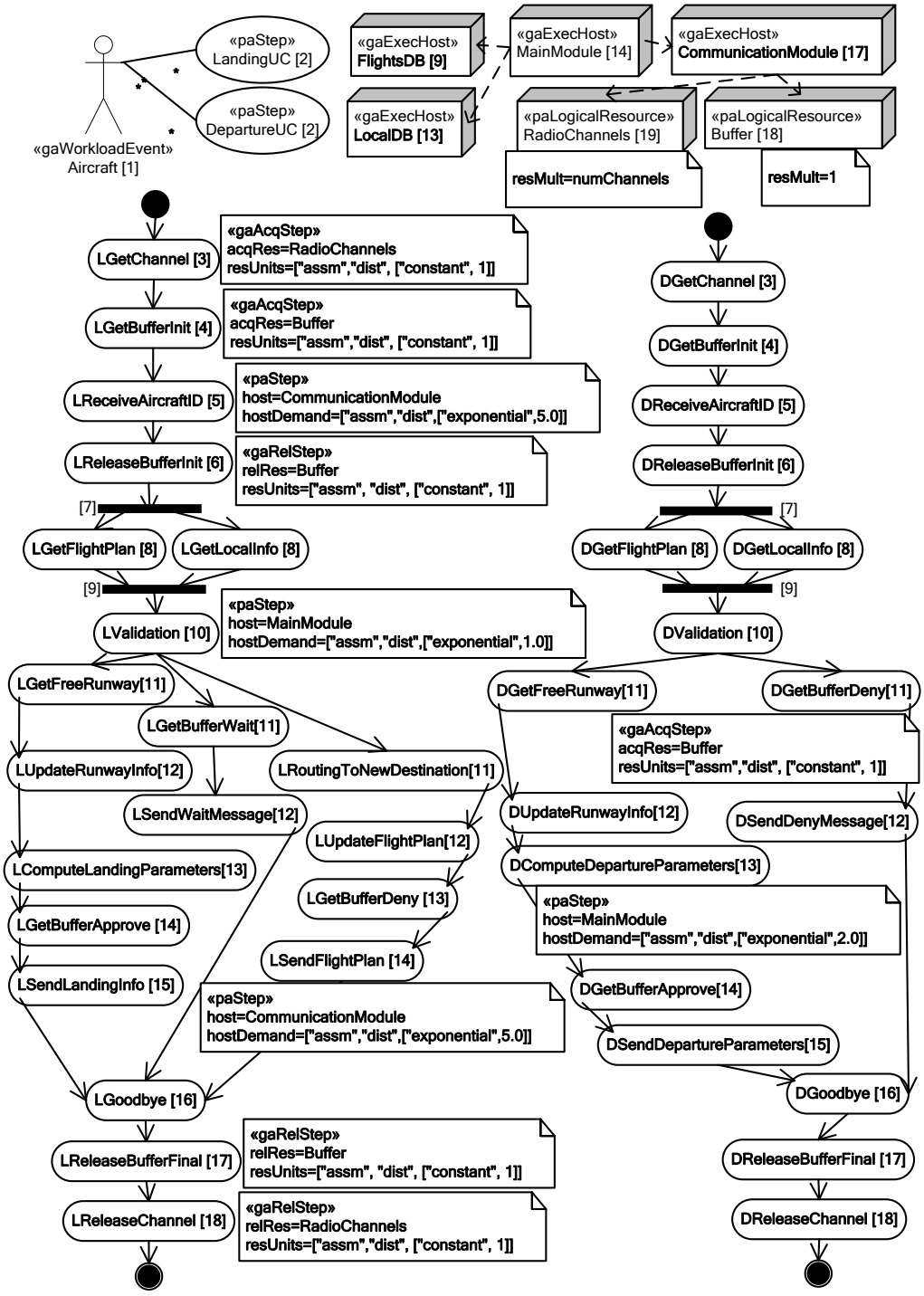

Figure 7.7: AATC initial use case, deployment and activity diagrams

The Departure scenario also involves communication between the aircraft and the AATC system by using a radio channel. Similarly to the Landing scenario, the system retrieves information from FlightsDB, based on aircraft ID.

Depending on the weather and runway availability, the aircraft is allowed to take off or not. In case of approval, the system sends the appropriate parameters to the aircraft. Otherwise, the aircraft will repeat the same scenario later, waiting for approval.

The system model also has configuration parameters, such as aircraft arrival pattern, probabilities of scenarios, number of available radio channels; they are defined in the configuration file, presented in figure 7.8.

```
var prob_landing=0.6;
var prob_departure=0.4;
var planeArrivalPattern=["closed", 8, ["exponential",70.0]];
var numChannels=100;
var iterationCount=10;
var simDuration=3000;
var confRelWidth=0.1;
```

Figure 7.8: AATC configuration file for the closed model

The value for *simDuration* was chosen as the minimum number of simulation time units (multiple of 1000) for which all output parameters have enough collected measurements in order for the statistical engine of the simulator to compute significant mean values. This condition is important both for the simulation of the entire model, and for the hybrid method, since this method inserts values obtained by simulation into the analytical solving procedure.

The first experiment was performed for $numChannels = 1$, and the results were predictable: a bottleneck was detected, by obtaining values of RadioChannels utilization higher than 0.9 by simulation and higher than 1 with hybrid and analytical approaches. However, two more entities had utilizations higher than 1 (or close to 0.9 by simulation): CommunicationModule and Buffer.

The number of radio channels was increased to 100 and the experiment repeated with the results of simulation (Sim[$simDuration/1000$]), hybrid analysis for different values of the *simulation level* (H[k]), and pure analytical solving (Ana[$k = 19$]) presented in Table 7.8.

While the analytical solver converges in 2 iterations (because the computed mean results are always the same), the simulation and the hybrid solver yield different results each time they are run: if the run is convergent, the results are slightly different according to the previously defined confidence interval, and if it is not convergent, then larger differences may occur. However, the simulator compensates for this problem by dividing the predefined run duration into 10 intervals and running the simulation 10 times, so that mean values and confidence intervals of output parameters are computed for values obtained over 10 simulations, instead of only one. Because simulation does not converge, higher values for *simDuration* were considered more relevant when simulating the entire model: a longer simulation will provide more accurate results.

As anticipated, the RadioChannels utilization problem was solved, but analysis results show the other two entities (CommunicationModule and Buffer) still have poor performance, having analytical and hybrid utilization values higher than 1. While simulation measurements for utilization values are always lower than 1 (but close to it, in case of a bottleneck), analytical

Table 7.8: Performance analysis results for initial design.

	S[10]	S[5]	S[3]	H[3]	H[4]	H[5]	H[11]	H[13]	H[14]	A[19]
U_{MM}	**0.16**	0.17	0.15	0.18	0.18	0.2	0.18	0.18	0.19	**0.17**
U_{FDB}	**0.15**	0.18	0.14	0.2	0.2	0.22	0.19	0.19	0.21	**0.19**
U_{LDB}	**0.37**	0.37	0.31	0.42	0.42	0.48	0.42	0.42	0.44	**0.42**
U_{CM}	**0.89**	0.86	0.88	1.01	1.01	**1.12**	1.01	1	**1.04**	**1.01**
U_{RC}	**0.04**	0.03	0.04	0.02	0.02	**0.04**	0.01	0.01	0.02	**0.01**
U_{Buf}	**0.89**	0.85	0.88	1.01	1.03	**2.74**	**1**	**0.96**	**1.21**	**1.01**
Resp. Time	**69.2**	67.1	72.3	**64.0**	**63.3**	49.5	**60.1**	**59.7**	50.6	**52.9**
Avg λ	**0.11**	0.11	0.1	0.068	0.072	0.099	0.062	0.06	0.069	**0.065**
Time	**1'1"**	0'31"	0'18"	1'30"	1'35"	1'24"	1'6"	0'24"	0'8"	**0.1"**
Iter. Count	-	-	-	6	5	5	7	3	2	2

methods, and consequently the hybrid approach, can lead to values higher than 1, the result showing the degree of resource overloading. Such a result may be used to fix design issues, it indicates resource multiplication: how many resources of a certain type should be in the system, in order for the requests to be properly handled and the bottleneck removed.

Not all possible values for the *simulation level*, k, between 3 and the maximum level yield valid results – this depends on the logic of the particular system being modeled. In case of AATC, levels ranging from 7 to 10 are not valid, since in that section of the diagram there are fork and join nodes, which create problems when they are not covered by the same model (simulation or analytical). Also, in the final section of the diagram, simulation does not make sense when starting between an acquire step and a release step, so 14 is the maximum considered value for k in the experiments, because no resources are acquired on lower levels.

Highlighted values for $k = 3, k = 4, k = 11, k = 13$ are closer to simulation results than to analytically computed values.

Parameter values that differ significantly from simulation results (reference values) are explained below.

In case $k = 5$, the simulation submodel does not include the first acquisition of Buffer, so the bottleneck is no longer situated at buffer level, but at CommunicationModule level, since the submodel includes all usages of this active resource by processing steps. This leads to high values for response time of such steps using CommunicationModule, which in turn leads to high values for resource demand and consequently resource utilization for passive resources (RadioChannels, Buffer), because the analytical solver computes resource demand as resource requested units multiplied by step response time (obtained by simulation).

In case $k = 14$, the simulation submodel starts below acquire steps LGetBufferDeny, LGetBufferWait, and DGetBufferDeny, so the bottleneck at Buffer is not observable in the submodel, and leads to high values of response time for steps involving CommunicationModule (which is intensively used below level 14); these values are then used by the analytical solver in computing resource demand for passive resources (Buffer) held during these steps (LSendLandingInfo, LSendFlightPlan, DSendDepartureParameters), so that utilization has a high value both for the active (CommunicationModule) and the passive resource (Buffer).

7.4.2 Design Improvements

In order to remove the bottleneck, the communication buffer should be split in two buffers: one for sending messages and one for receiving messages, so that the load is shared. The improved design is illustrated in figure 7.9, the use case diagram remains the same, while changes in activity diagrams are highlighted as notes with a dark-shaded background, since the referred entities are only visible as tagged values. In this case, the bottleneck is only moved from the buffer to calls to the module. This can be observed in Table 7.9.

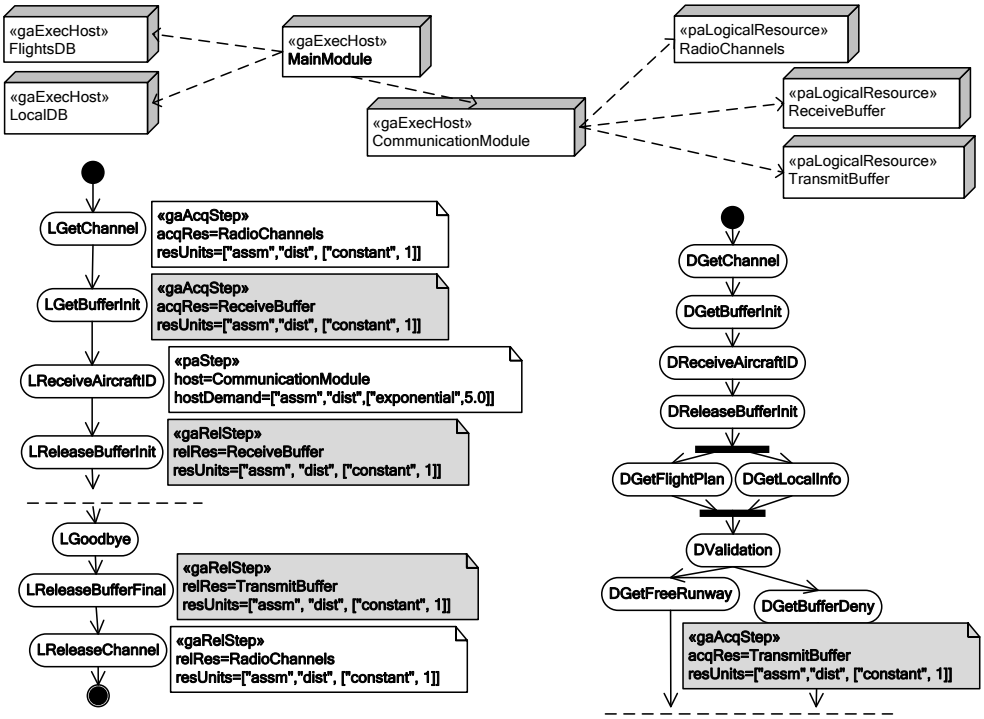

Figure 7.9: AATC improved deployment and activity diagrams

In case $k = 3$ and $k = 4$, values of utilization for ReceiveBuffer are more accurate (closer to simulation results) because resource demand is computed using response time values for steps LReceiveAircraftID and DReceiveAircraftID, which are part of the simulated submodel. In fact, all highlighted values for cases $k = 3$, $k = 4$ and $k = 5$ are closer to simulation results. Utilization values for TransmitBuffer should theoretically be lower than 1.00, since now the load should be divided between the two buffers; this explains the analytical and hybrid results for $k = 11$ and $k = 13$. However, simulation results show higher utilization values, both for TransmitBuffer and for ReceiveBuffer, because these resources are used in correlation with CommunicationModule, which still acts as a bottleneck.

In case $k = 14$, the previous explanation holds, with TransmitBuffer as the passive resource, instead of Buffer.

Table 7.9: Performance analysis results for improved design.

	S[10]	S[5]	S[3]	H[3]	H[4]	H[5]	H[11]	H[13]	H[14]	A[19]
U_{MM}	**0.16**	0.16	0.23	**0.18**	**0.18**	**0.18**	0.19	0.19	0.2	**0.19**
U_{FDB}	**0.18**	0.21	0.26	**0.2**	**0.2**	**0.2**	0.21	0.21	0.22	**0.21**
U_{LDB}	**0.42**	0.37	0.47	**0.43**	**0.43**	**0.42**	0.45	0.46	0.48	**0.46**
U_{CM}	**0.92**	0.91	0.85	**1.04**	**1.04**	**1.02**	1.07	1.08	**1.11**	**1.08**
U_{RC}	**0.04**	0.04	0.03	0.02	0.02	0.02	0.02	0.02	0.03	**0.02**
U_{RB}	**0.56**	0.55	0.47	**0.59**	**0.6**	0.33	0.35	0.36	0.37	**0.36**
U_{TB}	**0.89**	0.82	0.76	**1**	**0.96**	**0.96**	0.7	0.74	**2.04**	**0.71**
RT	**66.3**	57.1	53.9	**61.4**	**61.3**	**62.4**	49.7	49.0	42.6	**42.3**
Avg λ	**0.1**	0.1	0.12	0.063	0.062	0.068	0.066	0.061	0.086	**0.071**
Time	**1'4"**	0'32"	0'21"	1'43"	1'48"	2'2"	0'40"	0'20"	0'26"	**0.06"**
Iter. Count	-	-	-	5	5	6	5	2	4	2

The real solution to remove the bottleneck is to also split CommunicationModule, so that, in activity diagrams, incoming messages will be hosted by ReceiveModule and use ReceiveBuffer, while outgoing messages will be hosted by TransmitModule and be stored in TransmitBuffer. The changes in deployment and the Landing activity diagram can be seen in figure 7.10, the Departure diagram has similar changes. The resource contention being solved, enough values of output parameters are collected in fewer simulation time units: $simDuration = 1000$. Analysis results are presented in Table 7.10.

Table 7.10: Performance analysis results for best design.

	S[10]	S[5]	S[1]	H[3]	H[4]	H[5]	H[11]	H[13]	H[14]	A[19]
U_{MM}	**0.21**	0.18	0.15	0.19	0.19	0.19	0.19	0.19	0.19	**0.19**
U_{FDB}	**0.19**	0.23	0.21	0.22	0.22	0.22	0.22	0.22	0.21	**0.22**
U_{LDB}	**0.45**	0.39	0.48	0.46	0.46	0.46	0.46	0.46	0.45	**0.46**
U_{RM}	**0.36**	0.34	0.39	0.36	0.36	0.36	0.36	0.36	0.35	**0.36**
U_{TM}	**0.7**	0.65	0.74	0.72	0.72	0.72	0.72	0.72	0.71	**0.72**
U_{RC}	**0.03**	0.03	0.03	0.02	0.02	0.02	0.02	0.02	0.02	**0.02**
U_{RB}	**0.36**	0.33	0.37	0.34	0.36	0.36	0.36	0.36	0.35	**0.36**
U_{TB}	**0.7**	0.65	0.73	0.73	0.7	0.72	0.74	0.73	**1.47**	**0.72**
RT	**42.1**	40.9	50.7	56.9	57.0	57.3	48.0	47.5	47.9	**40.5**
Avg λ	**0.1**	0.1	0.1	0.077	0.075	0.086	0.073	0.065	0.074	**0.072**
Time	**1'14"**	0'38"	0'7"	0'47"	1'10"	0'43"	0'32"	0'9"	0'7"	**0.06"**
Iter. Count	-	-	-	5	8	5	9	4	5	2

In case $k = 14$, the previous explanation holds with TransmitModule as the active resource, instead of CommunicationModule; since splitting the CommunicationModule solved the global resource bottleneck, TransmitModule is no longer affected, only TransmitBuffer is locally intensively utilized in the simulation submodel.

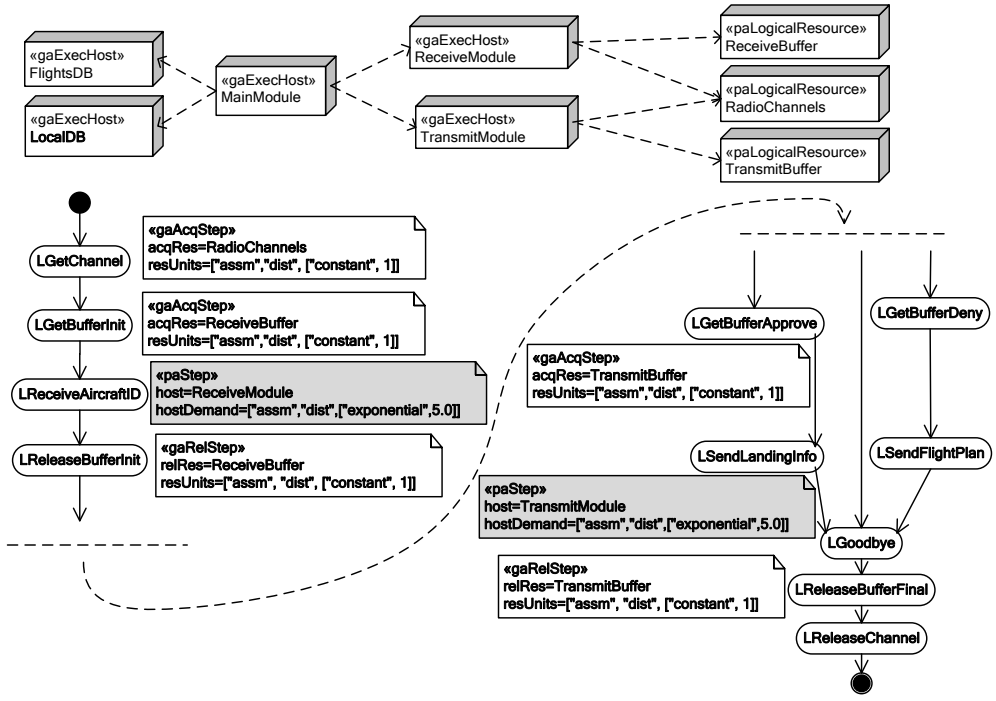

Figure 7.10: AATC best diagrams: deployment and Landing activity diagram

7.4.3 Validation of Improvements to Analytical Estimations

In order to compare the accuracy of Bard-Schweitzer (A-MVA) and Chandy-Neuse estimations, results of the hybrid solver for the best system design and several values of the *simulation level*, k, are presented in Table 7.11. Simulation results for 10000 time units are used as reference values.

Table 7.11: Performance analysis results for best design, using A-MVA

	S[10]	H[3]	H[4]	H[5]	H[11]	H[13]	H[14]	A[19]
U_{MM}	**0.21**	0.18	0.18	0.18	0.18	0.18	0.18	**0.18**
U_{FDB}	**0.19**	0.2	0.2	0.2	0.2	0.2	0.2	**0.2**
U_{LDB}	**0.45**	0.43	0.43	0.43	0.43	0.43	0.43	**0.43**
U_{RM}	**0.36**	0.34	0.34	0.34	0.34	0.34	0.34	**0.34**
U_{TM}	**0.7**	0.67	0.67	0.67	0.67	0.67	0.67	**0.67**
U_{RC}	**0.03**	0.02	0.02	0.02	0.02	0.02	0.03	**0.02**
U_{RB}	**0.36**	0.33	0.33	0.34	0.34	0.34	0.34	**0.34**
U_{TB}	**0.7**	0.67	0.64	0.64	0.67	0.71	2.01	**0.67**
RT	**42.11**	65.38	65.23	64.9	56.04	55.83	53.95	**48.62**
Avg λ	**0.1**	0.07	0.07	0.083	0.065	0.065	0.076	**0.067**
Time	**1'14"**	1'12"	0'55"	0'56"	0'23"	0'7"	0'20"	**0.08"**
Iter. Count	-	8	6	8	6	3	8	2

In conclusion, analytical results, and consequently hybrid solving results are less accurate than results obtained by using the Chandy-Neuse estimation (see Table 7.10).

7.4.4 Heuristic Rules for Choosing the Simulation Level

After studying the experimental results and finding explanations based on the hybrid algorithm, the following heuristic rules for choosing the *simulation level* can be extracted:

- The simulation submodel should include at least one resource.
- The simulation submodel should not start between fork and join nodes.
- The simulation submodel should not start between acquire and release nodes (as proven by cases $k = 5$ for the initial design and $k = 14$ for all design alternatives).

Solving for $k = 3$ and $k = 4$ is a bit slow because most of the model is simulated, so the optimal value for k in this application is 13.

7.4.5 Performance Results

The hybrid method results depend on the *simulation level* k. For low values, the solver is accurate but time-consuming (since a large part of the system model is simulated); simulation at high levels is very expensive: the number of possibilities to be experimented is greater than at lower levels. For large values, resources are outside the simulation submodel.

Intermediate values (such as 11 and 13, in the considered system model) are accurate and the solution time is low (most values below 1 minute). For $k = 13$, solving time is comparable to simulation for the same number of iterations, because MVA is fast, even with the simulated submodel being evaluated several times. The hybrid solving time is lower than more accurate simulations (run for more iterations).

In all cases, the hybrid method was convergent after only a few iterations, which led to low solving times for high values of k, because only a small submodel was simulated.

7.5. Summary

The order of validating the implemented approaches is very important:

- First, the simulator is validated, in order to consider it as a reference when validating the hybrid approach.
- The improved analytical approach is then validated, compared to classical analytical calculus and also to simulation results. This step is necessary, so the comparison of the hybrid analysis to pure analytical results can be trusted.
- Finally, hybrid analysis results are compared to both simulation and analytical solving, and also to measurements on a real online system.

The simulator validation results show that it follows the behavior of the system, according to performance annotations of the input models.

The analytical solver is obviously improved for open systems, by using the formula modified by the author of this thesis. Other improvements to the analytical solver can be observed only in the context of a more complex system, such as the case study in section 7.4.

In case of the Authentication System, the analysis results show that for higher levels (low values for the *simulation level* k) the results accuracy increases, they are closer to simulation results. Hence, the hybrid solver is more accurate than analytical methods. The thesis author defined and implemented the measurement strategy, based on code instrumentation, and also performed post-processing of data samples.

The proposed hybrid method proved to be faster than simulation and yielded results that are more accurate than the analytical approach (in case $k = 4$ for the considered case study). Measurements on the real online system were very close to results of the hybrid approach.

The AATC system model is an original case study built by the thesis author after reviewing behavioral aspects of existing systems. Additionally to obtaining complex activity diagrams, the purpose of this case study was mainly to illustrate how the system design can evolve based on the performance analysis feedback loop.

The accuracy of performance results for AATC system models depends on the simulation level, so that several heuristic rules on how to choose this level were deduced. Intermediate values of k proved to be the best options, yielding accurate results with low analysis durations ($k = 13$).

8. CONTRIBUTIONS AND FUTURE WORK

8.1. Contributions

This thesis addresses performance analysis based on models of distributed systems, an important research direction of a timely domain, SPE. An adjacent field also covered in the thesis is analysis automation and standardization of modeling techniques for distributed systems. In order to propose innovative approaches, thorough knowledge is required concerning both the foundations and the latest developments in the field.

The main contributions of the thesis author are:

- Comprehensive study and systematization of publications in the SPE field and related areas.
  - A wide range of papers are organized according to the contributions they bring to a certain research direction; some of them have several contributions, both to performance analysis and to model transformation and standardization.
- New hybrid performance model defined by the thesis author.
  - Scenario oriented meta-model: straightforward mapping to UML items (with performance annotations).
  - Class III hybrid model: an analytical model of the system uses results from simulation of a submodel.
  - First time a class III hybrid model is applied to distributed software systems.
  - Generality: model break down strategy (to separate the submodel) implemented both hierarchically and sequentially (in case there are too few hierarchical layers, or the hierarchy is not specified at all).
- Hybrid solver approach created, implemented and validated by the thesis author.
  - New approach in combining simulation results with analytical calculus.
  - Flexibility: mixed open and closed requests are supported.
  - Enhanced formulae for analytical computations in case of open models.
  - Good understanding, implementation and integration of existing techniques in the analytical solver: A-MVA, Chandy-Neuse, resource multiplicity correction factor.
- Standardization of input model specification
  - Wide range of analyzable models: either activity, or sequence diagrams are supported to specify input model behavior.
  - Flexibility: SPT annotations are interpreted where the MARTE profile lacks clarity.

- PHYMSS tool built by the author to implement the proposed hybrid approach and an improved simulator.
  - XMI model parsing: UML MARTE annotations.
  - Simulation model enhancements: multithreaded implementation and nested calls.
  - Pure analytical approach made possible, by skipping the simulation step.
  - Enhanced user experience: not only a proof-of-concept tool, but a user-friendly (GUI toolbar, simulation monitoring panel) and robust application (exception handling techniques).
- Hybrid method validation
  - Validation sequence: simulator first, used as a reference in analytical method validation and later for comparison with the hybrid approach.
  - Measurements: code instrumentation in client and server of the authentication system; deduction of distributions, by building histograms of the samples.
- Original case study design: AATC
  - Realistic: based on studying the behavior of real air-traffic control systems.
  - Complexity: various control flow structures are included in the activity diagrams.
  - Model evolution: step-by-step improvements guided by performance analysis results.

8.2. Publications

The PhD research activity is described in two PhD Reports:

- PhD Report #1: "A Hybrid Approach to Performance Evaluation of Distributed Systems", presented in 2009.
- PhD Report #2: "Performance Hybrid Model Solver and Simulator. Novel Approach to Hybrid Method Composition.", presented in 2010.

Several papers were published in proceedings of international symposiums and conferences; they are presented below in chronological order:

- "Simulation-based performance evaluation of distributed software systems. A case study". In 8th International Conference on Technical Informatics, CONTI'2008, June 5-6, Vol. 2 Computer and Software Engineering, CONTI '08, pages 85–90, 2008.
  - The paper presents the benefits and methodology of performance prediction using an existing simulator, called $UML - \Psi$; the design of an air-traffic control system is evaluated and improved based on simulation results.
- "Towards early performance assessment based on UML MARTE models for distributed systems". In 5th International Symposium on Applied Computational Intelligence and Informatics, SACI 2009, May 28-29, SACI '09, pages 521–526. IEEE, 2009. (ISI Proceedings)

  - An initial performance meta-model is defined starting from input UML MARTE deployment, use case and activity diagrams; the meta-model allows straightforward transformation into either a simulation or an analytical model.

- "Phymss: performance hybrid model solver and simulator based on UML MARTE diagrams". In Proceedings of the first joint WOSP/SIPEW international conference on Performance engineering, WOSP/SIPEW '10, pages 243–244, New York, NY, USA, 2010. ACM. (International Database, ACM Sponsored)

  - A first approach to the hybrid solving algorithm and a tool that implements it, PHYMSS, are presented.

- "Hybrid analytical-simulation model used to evaluate and improve system performance". In Proceedings of the 10th International Symposium on Parallel and Distributed Computing (ISPDC 2011), ISPDC 2011, 2011. (International Database, IEEE Sponsored)

  - The hybrid meta-model and solving algorithm are extended to allow nested calls, as given in input UML sequence diagrams with MARTE annotations. A real online system is modeled to validate to accuracy of performance prediction results obtained with PHYMSS.

- "Performance prediction for UML MARTE models with a hybrid model solver". In Proceedings of the 6th International Conference "Zilele Academiei de Ştiinţe Tehnice din Romania" (Zilele ASTR 2011), 2011.

  - Summary of the entire research activity, implementation in PHYMSS and validation results.

The author also published papers in the Scientific Bulletin of "Politehnica" University of Timişoara, Romania, Transactions on Automatic Control and Computer Science:

- "Improving Distributed Systems Design by Simulation-Based Performance Analysis. A Case Study.", BS-UPT TACCS Volume 53(67) No. 3 / September 2008.

  - The air-traffic control system case study is presented in detail, showing how simulation results can be used for performance analysis and design improvements.

- "Automation of Performance Assessment Applied to UML MARTE Models for Distributed Systems", BS-UPT TACCS Volume 55(69) No. 2 / June 2010.

  - The transformation rules from UML MARTE diagrams to the proposed hybrid meta-model are stated and PHYMSS, the tool that implements the hybrid solver based on the meta-model is described.

For all previously mentioned papers, Cosmina Chişe is the first author and the scientific advisor, Prof. Dr. Eng. Ioan Jurca is the second author.

8.3. Future Work Directions

- Extended range of input diagrams.
  - Communication diagrams (formerly called collaboration diagrams) that illustrate object interaction from a different perspective than sequence diagrams; however, the messages are numbered, so the sequencing can be extracted, even if it is not obvious.
  - Interaction Overview diagrams that combine activity and sequence diagrams by using interaction fragments (referencing various other diagrams) as part of the control flow.
- PHYMSS GUI: configuration editor.
  - The configuration information is currently read from the JS file and displayed by the tool.
  - The usage would be improved by having the possibility to edit the settings directly from the tool and saving the changes in the JS file.
- Heuristically detect the appropriate value for the simulation level.
  - A set of heuristic rules have already been extracted for the case studies used during validation.
  - In order to generalize the rules, more experiments should be performed.
  - These rules can be implemented in the tool, in order to assist the user in choosing an appropriate simulation level: a restricted set of values could be indicated by the tool.
- Extensions to the hybrid meta-model and solver.
  - The meta-model could be extended to include domain-specific performance parameters, such as Quality of Service attributes, and a solver could be developed to check conformance to SLAs (Service-Level Agreements).
  - An alternative to MVA as part of the hybrid solver is assessment of worst-case scenarios, which is widely used for analysis of real-time systems: the analytical solver and the propagation rules may be adapted in order to obtain worst-case values instead of mean values for performance parameters.
- Dissemination of future research results.
  - The author will publish papers to describe methodology evolution and corresponding validation results, and also attend conferences in the field of SPE or other related domains.

A. PURE ANALYTICAL SOLVING TECHNIQUES

A.1. Bard-Schweitzer Algorithm for Closed Queueing Networks (A-MVA)

The main parameters involved in the calculus and their formulae are the following:

- N_i is the population count for requests of type i, N_i^k is the population count for requests of type i when there is one less request of type k:
  $N_i^k = \begin{cases} N_i, & i \neq k \\ N_i - 1, & i = k \end{cases}$;
- $Q_{i,j}$ is the queue length for resource j, $j = 0..M-1$, and request type i, $i = 0..K-1$, $Q_{i,j}^k$ is the queue length when there is one less request of type k:
  $Q_{i,j}^k = \begin{cases} Q_{i,j}, & i \neq k \\ \frac{N_i - 1}{N_i} \cdot Q_{i,j}, & i = k \end{cases}$;
- $F_{i,j}$ is the fraction of requests of type i which require service from resource j, Fi, j^k is the fraction of requests at a resource, when there is one less request of type k: $F_{i,j} = \frac{Q_{i,j}}{N_i}$;
- $D_{i,j,k}$ is the difference in the fraction of requests of type i resulting from the removal of one request of type k: $D_{i,j,k} = F_{i,j}^k - F_{i,j}$.

In conclusion, the formula that relates all above mentioned parameters is:
$Q_{i,j}^k = N_i^k \cdot (F_{i,j} + D_{i,j,k})$.
Algorithm Inputs:

- K = number of request types,
- M = number of resources (service centers),
- N_i,
- $Q_{i,j}$,
- Di, j, k,
- $S_{i,j}$ = demand by requests of type i from resource j,
- $V_{i,j}$ = visit rate for requests of type i at resource j.

Algorithm Outputs:

- $R_{i,j}$ = response time for requests of type i at resource j,
- $Thr_{i,j}$ = throughput for server j and requests of type i,

- $U_{i,j}$ = utilization of resource j by requests of type i,
- $Q_{i,j}$ is propagated between iterations and used in the convergence test.

The algorithm is iterative and a typical iteration is described below.

Initially $Q_{i,j} = \frac{N_i}{M}$, meaning that requests are equally distributed among resource queues.

$F_{i,j} = \frac{Q_{i,j}}{N_i}$, $Q^k_{i,j} = N^k_i \cdot (F_{i,j} + D_{i,j,k})$, $D_{i,j,k} = 0$ (A-MVA) $\Rightarrow Q^k_{i,j} = \frac{N^k_i}{N_i} \cdot Q_{i,j}$

$R_{i,j} = \begin{cases} S_{i,j} \cdot \left(1 + Q^i_j\right), & \text{for a single-server resource} \\ S_{i,j}, & \text{for delay centers} \end{cases}$;

$Q^k_j = \Sigma^{M-1}_{i=0} Q^k_{i,j}$;

$Q_{i,j} = N_i \cdot \frac{V_{i,j} \cdot R_{i,j}}{\Sigma^{M-1}_{m=0} V_{i,m} \cdot R_{i,m}}$;

$Thr_{i,j} = \frac{Q_{i,j}}{R_{i,j}} = \frac{N_i \cdot V_{i,j}}{\Sigma^{M-1}_{m=0} V_{i,m} \cdot R_{i,m}}$;

$U_{i,j} = Thr_{i,j} \cdot S_{i,j}$.

Convergence condition: $\frac{|Q^{new}_{i,j} - Q^{previous}_{i,j}|}{Q^{previous}_{i,j}} < \delta$, where δ is a predefined constant that enforces solution accuracy.

A.2. Chandy-Neuse Algorithm for Closed Queueing Networks (Linearizer)

The fraction $F_{i,j}$ is assumed to be a linear function of population, so that $D_{i,j,k} = D^p_{i,j,k}$.

$D^p_{i,j,k}$ is the difference of fractions caused by removal of a request of type k, for an input population that has one less request of type p.

$D_{i,j,k}$ is the difference of fractions caused by removal of a request of type k, when the population of type p is complete.

Algorithm Inputs: K, M, N_i, $Q_{i,j}$, $D_{i,j,k}$, $S_{i,j}$, $V_{i,j}$.

Algorithm Outputs: $R_{i,j}$, $Thr_{i,j}$, $U_{i,j}$, $Q_{i,j}$.

The algorithm is iterative and a typical iteration is described below.

Initially $D_{i,j,k} = 0$ and $Q_{i,j} = \frac{N_i}{M}$, meaning that requests are equally distributed among resource queues.

Apply A-MVA algorithm with $D_{i,j,k}$ and $F_{i,j}$ propagated from previous iterations.

For each request type p, apply A-MVA for the system with one less request of that type: A-MVA is applied K times using as inputs $D^p_{i,j,k}$ and $Q^p_{i,j}$ computed in previous iterations ($D^p_{i,j,k} = D_{i,j,k}$).

Compute $F_{i,j}$, $F^k_{i,j}$, $D_{i,j,k}$.

Convergence condition: $\frac{|Q^{new}_{i,j} - Q^{previous}_{i,j}|}{Q^{previous}_{i,j}} < \delta$, where δ is a predefined constant that enforces solution accuracy.

A.3. MVA Algorithm for Mixed Queueing Networks

Open requests are handled first and the Open Queueing Network (OQN) is partially solved, then an inflate factor, F_j , is computed and passed to the Closed Queueing Network (CQN) solver; results (queue length values) from the CQN solver are then used to adjust the output parameter values.

Algorithm Inputs: K_{closed} (number of closed requests), K_{open} (number of open requests), M, N_i, $Q_{i,j}$, $D_{i,j,k}$, $S_{i,j}$, $V_{i,j}$, λ_i = arrival rate (throughput) of open requests of type i.

Algorithm Outputs: $R_{i,j}$, $Thr_{i,j}$, $U_{i,j}$, $Q_{i,j}$.

The algorithm starts with the calculus of utilization and throughput for the OQN:

$U_j = \Sigma_{i=0}^{K_{open}-1}(\lambda_i \cdot V_{i,j} \cdot S_{i,j})$, $Thr_{i,j} = \lambda_i \cdot V_{i,j}$.

$F_j = \frac{1}{1-U_j}$ is the inflate factor; it is passed to the CQN (Chandy-Neuse or Bard-Schweitzer) solver, to be used when computing the response time:

$R_{i,j} = F_j \cdot V_{i,j} \cdot S_{i,j} \cdot (1 + Q_{i,j})$.

After solving the CQN, the parameter values are aggregated, so they can be used in the final response time and queue calculus:

$Q_{closed,j} = \Sigma_{i=0}^{K_{closed}-1} Q_{i,j}$;

$R_{i,j} = F_j \cdot V_{i,j} \cdot S_{i,j} \cdot (1 + Q_{closed,j})$;

$Q_{i,j} = \lambda_i \cdot R_{i,j}$.

B. THE UML MARTE PROFILE

MARTE is a replacement for the SPT profile. The latter provides a grammar for powerful concepts (symbolic variables and time expressions), but does not support user-defined NFPs and specialized domains. The MARTE NFP (NonFunctional Properties) modeling framework reuses structural concepts in QoS&FT (Quality of Service & Fault Tolerance), but reduces the usage complexity; also, it introduces VSL (Value Specification Language) which extends and formalizes concepts from TVL (Tag Value Language, defined in SPT).

Besides the NFP, Time and Core Elements, there is a package which provides notions to define the execution platform: Generic Resource Modeling (GRM).

The basis for all analysis packages is the Generic Quantitative Analysis Modeling (GQAM) package that defines basic modeling concepts and NFPs. Many stereotypes defined in this package are useful for performance analysis, the most important are listed below:

- GaWorkloadEvent used to describe the system workload, with the "pattern" tagged value.
- GaExecHost used to refine the Resource definition (from the GRM package).
- GaStep provides most tagged values inherited by PaStep (in the Performance Analysis Modeling sub-profile): rep (repetitions), prob (probability), hostDemand, respT (response time).
- GaAcqStep, GaRelStep, used to define resource acquire/release steps.

The Performance Analysis Modeling (PAM) sub-profile comes with a few specific stereotypes and tagged values. This package provides means to express the input and output of performance analysis methodologies. As input, UML diagrams are used, with well-defined extensions for performance related annotations (parameters, such as request arrival rate, mean execution time for external operations). These are called parameterized NFPs and are used to define variations that may occur in the studied system, defined in a certain AnalysisContext. The AnalysisContext consists of system behavior (scenarios), resources and workload. Also, platform specific (environment) information should be provided as input in order to allow for certain analysis methods (like simulation) to be carried on accurately. There are predefined libraries with models for middleware or operating systems and they can be integrated into the system model. Output can also be expressed as NFPs of the UML model elements: response time of scenarios, utilization of resources, throughput (especially for communication resources, expressed in operations/second). The main extensions in the package are illustrated in figure B.1, along with the basic types defined in GQAM from which they are derived.

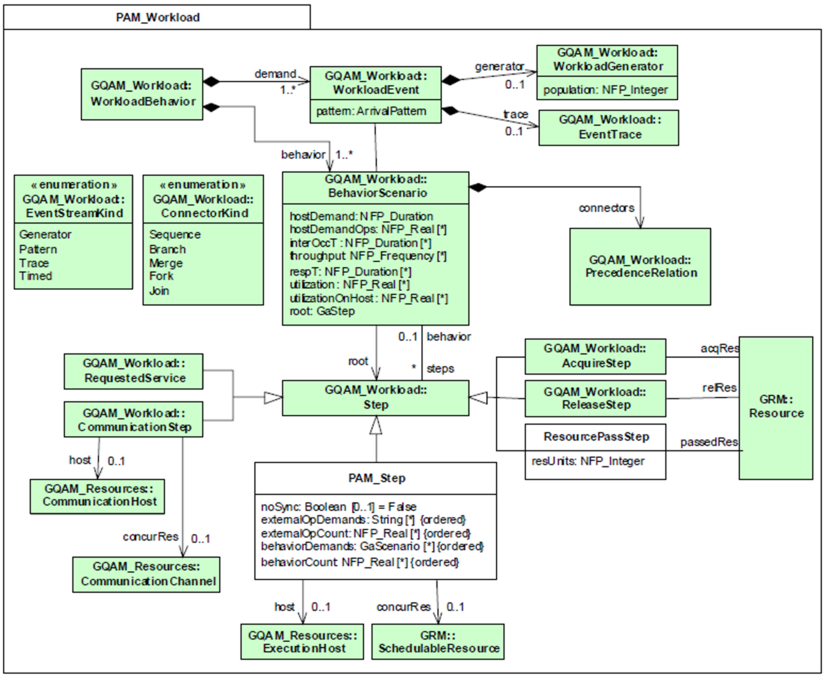

Figure B.1: MARTE performance extensions for workload, behavior and time observations [89]

C. OPEN MODEL BEHAVIOR DESCRIBED STEP BY STEP

The computation of performance parameters for the Help Desk open system described in chapter 7.1 is illustrated by the table with intermediate values in figure C.1. Microsoft Excel was used to derive each row from the previous one, based on automatically applied formulae.

Req. Type	Sim. Time	CPU Q. len.	Disk Q. len.	CPU Dmd.	Disk Dmd.	CPU Wait Time	Disk Wait Time	CPU Response Time	Disk Response Time	CPU Processing End Time	Disk Processing End Time
T1	0.52	0	0	0.237	0.316	0.000000	0.000000	0.237000	0.316000	0.757833	0.836833
T1	1.04	0	0	0.237	0.316	0.000000	0.000000	0.237000	0.316000	1.278667	1.357667
T1	1.56	0	0	0.237	0.316	0.000000	0.000000	0.237000	0.316000	1.799500	1.878500
T1	2.08	0	0	0.237	0.316	0.000000	0.000000	0.237000	0.316000	2.320333	2.399333
T2	2.44	0	0	0.046	0.062	0.000000	0.000000	0.046000	0.062000	2.485024	2.501024
T1	2.60	0	0	0.237	0.316	0.000000	0.000000	0.237000	0.316000	2.841167	2.920167
T1	3.13	0	0	0.237	0.316	0.000000	0.000000	0.237000	0.316000	3.362000	3.441000
T1	3.65	0	0	0.237	0.316	0.000000	0.000000	0.237000	0.316000	3.882833	3.961833
T3	3.70	1	1	0.607	0.809	0.179130	0.258130	0.786130	1.067130	4.489833	4.770833
T1	4.17	1	1	0.237	0.316	0.323167	0.604167	0.560167	0.920167	4.726833	5.086833
T1	4.69	1	2	0.237	0.316	0.039333	0.399333	0.276333	0.715333	4.963833	5.402833
T2	4.88	1	2	0.046	0.062	0.085785	0.524785	0.131785	0.586785	5.009833	5.464833
T1	5.21	0	1	0.237	0.316	0.000000	0.256500	0.237000	0.572500	5.445333	5.780833
T1	5.73	0	1	0.237	0.316	0.000000	0.051667	0.237000	0.367667	5.966167	6.096833
T1	6.25	0	0	0.237	0.316	0.000000	0.000000	0.237000	0.316000	6.487000	6.566000
T1	6.77	0	0	0.237	0.316	0.000000	0.000000	0.237000	0.316000	7.007833	7.086833
T1	7.29	0	0	0.237	0.316	0.000000	0.000000	0.237000	0.316000	7.528667	7.607667
T2	7.32	1	1	0.046	0.062	0.211593	0.290593	0.257593	0.352593	7.574667	7.669667
T3	7.41	2	2	0.607	0.809	0.167259	0.262259	0.774259	1.071259	8.181667	8.478667
T1	7.81	1	1	0.237	0.316	0.369167	0.666167	0.606167	0.982167	8.418667	8.794667
T1	8.33	1	2	0.237	0.316	0.085333	0.461333	0.322333	0.777333	8.655667	9.110667
T1	8.85	0	1	0.237	0.316	0.000000	0.256500	0.237000	0.572500	9.091167	9.426667
T1	9.38	0	1	0.237	0.316	0.000000	0.051667	0.237000	0.367667	9.612000	9.742667
T2	9.76	0	0	0.046	0.062	0.000000	0.000000	0.046000	0.062000	9.802098	9.818098
T1	9.90	0	0	0.237	0.316	0.000000	0.000000	0.237000	0.316000	10.132833	10.211833
T1	10.42	0	0	0.237	0.316	0.000000	0.000000	0.237000	0.316000	10.653667	10.732667
T1	10.94	0	0	0.237	0.316	0.000000	0.000000	0.237000	0.316000	11.174500	11.253500
T3	11.11	1	1	0.607	0.809	0.063389	0.142389	0.670389	0.951389	11.781500	12.062500
T1	11.46	1	1	0.237	0.316	0.323167	0.604167	0.560167	0.920167	12.018500	12.378500
T1	11.98	1	2	0.237	0.316	0.039333	0.399333	0.276333	0.715333	12.255500	12.694500
T2	12.20	1	2	0.046	0.062	0.060378	0.499378	0.106378	0.561378	12.301500	12.756500
T1	12.50	0	1	0.237	0.316	0.000000	0.256500	0.237000	0.572500	12.737000	13.072500
T1	13.02	0	1	0.237	0.316	0.000000	0.051667	0.237000	0.367667	13.257833	13.388500
T1	13.54	0	0	0.237	0.316	0.000000	0.000000	0.237000	0.316000	13.778667	13.857667
T1	14.06	0	0	0.237	0.316	0.000000	0.000000	0.237000	0.316000	14.299500	14.378500
T1	14.58	0	0	0.237	0.316	0.000000	0.000000	0.237000	0.316000	14.820333	14.899333
T2	14.63	1	1	0.046	0.062	0.186187	0.265187	0.232187	0.327187	14.866333	14.961333
T3	14.81	2	2	0.607	0.809	0.051519	0.146519	0.658519	0.955519	15.473333	15.770333
T1	15.10	1	1	0.237	0.316	0.369167	0.666167	0.606167	0.982167	15.710333	16.086333
T1	15.63	1	2	0.237	0.316	0.085333	0.461333	0.322333	0.777333	15.947333	16.402333

Figure C.1: Computation of performance parameters for the Help Desk system

Bibliography

[1] ArgoUML object-oriented design tool with cognitive support, URL http://argouml.tigris.org/.

[2] GreatSPN GRaphical Editor and Analyzer for Timed and Stochastic Petri Nets, URL http://www.di.unito.it/~greatspn/index.html.

[3] MARTE Profile for Rational Software Architect (RSA) 7.0, URL http://www.omgmarte.org/node/31.

[4] NS-2 network simulation tool, URL http://www.isi.edu/nsnam/ns/.

[5] Papyrus UML diagram editor, URL http://www.papyrusuml.org/.

[6] F. Andolfi, F. Aquilani, S. Balsamo, and P. Inverardi, Deriving performance models of software architectures from message sequence charts, in *Proceedings of the 2nd international workshop on Software and performance*, WOSP '00, pp. 47–57, ACM, New York, NY, USA, 2000, ISBN 1-58113-195-X, URL http://doi.acm.org/10.1145/350391.350404.

[7] F. Aquilani, S. Balsamo, and P. Inverardi, Performance analysis at the software architectural design level, *Perform. Eval.*, vol. 45: pp. 147–178, July 2001, ISSN 0166-5316, URL http://portal.acm.org/citation.cfm?id=383224.383229.

[8] L. B. Arief, *A Framework for Supporting Automatic Simulation Generation from Design*, Ph.D. thesis, Dept. of Computer Science, University of Newcastle Upon Tyne, UK, 2001.

[9] L. B. Arief and N. A. Speirs, A UML tool for an automatic generation of simulation programs, in *Proceedings of the 2nd international workshop on Software and performance*, WOSP '00, pp. 71–76, ACM, New York, NY, USA, 2000, ISBN 1-58113-195-X, URL http://doi.acm.org/10.1145/350391.350408.

[10] S. Balsamo, A. Di Marco, P. Inverardi, and M. Simeoni, Model-Based Performance Prediction in Software Development: A Survey, *IEEE Trans. Softw. Eng.*, vol. 30: pp. 295–310, May 2004, ISSN 0098-5589, URL http://0-portal.acm.org.millennium.lib.cyut.edu.tw/citation.cfm?id=987527.987640.

[11] S. Balsamo, P. Inverardi, and C. Mangano, An approach to performance evaluation of software architectures, in *Proceedings of the 1st international workshop on Software and performance*, WOSP '98, pp. 178–190, ACM, New York, NY, USA, 1998, ISBN 1-58113-060-0, URL http://doi.acm.org/10.1145/287318.287354.

[12] S. Balsamo, R. Mamprin, and M. Marzolla, Performance Evaluation of Software Architectures with Queuing Network Models, in *Proceedings of the European Simulation and Modelling Conference 2004*, ESMc '04.

[13] S. Balsamo and M. Marzolla, A simulation-based approach to software performance modeling, in *Proceedings of the 9th European software engineering conference held jointly with 11th ACM SIGSOFT international symposium on Foundations of software engineering*, ESEC/FSE-11, pp. 363–366, ACM, New York, NY, USA, 2003, ISBN 1-58113-743-5, URL http://doi.acm.org/10.1145/940071.940122.

[14] S. Balsamo and M. Simeoni, Deriving Performance Models from Software Architecture Specifications, in *Proceedings of the European Simulation Muticonference 2001*, ESM 2001.

[15] J. Banks, J. S. Carson, B. L. Nelson, and D. M. Nicol, *Discrete-Event System Simulation*, Prentice Hall, 5th ed., 2010, ISBN 0136062121, URL http://www.pearsonhighered.com/educator/product/Discrete-Event-System-Simulation-5E/9780136062127.page.

[16] S. Becker, H. Koziolek, and R. Reussner, Model-Based performance prediction with the palladio component model, in *Proceedings of the 6th international workshop on Software and performance*, WOSP '07, pp. 54–65, ACM, New York, NY, USA, 2007, ISBN 1-59593-297-6, URL http://doi.acm.org/10.1145/1216993.1217006.

[17] S. Bernardi, S. Donatelli, and J. Merseguer, From UML sequence diagrams and statecharts to analysable petri net models, in *Proceedings of the 3rd international workshop on Software and performance*, WOSP '02, pp. 35–45, ACM, New York, NY, USA, 2002, ISBN 1-58113-563-7, URL http://doi.acm.org/10.1145/584369.584376.

[18] M. Bernardo, *Theory and Application of Extended Markovian Process Algebra*, Ph.D. thesis, University of Bologna, Italy, 1999.

[19] M. Bernardo, P. Ciancarini, and L. Donatiello, ÆMPA: a process algebraic description language for the performance analysis of software architectures, in *Proceedings of the 2nd international workshop on Software and performance*, WOSP '00, pp. 1–11, ACM, New York, NY, USA, 2000, ISBN 1-58113-195-X, URL http://doi.acm.org/10.1145/350391.350394.

[20] A. Bertolino and R. Mirandola, Towards Component-based Software Performance Engineering, in *Proceedings of the6th ICSE Workshop on Component-Based Software Engineering*, CBSE '03, Carnegie Mellon University, USA, and Monash University, Australia, 2003, URL http://www.csse.monash.edu.au/~hws/cgi-bin/CBSE6/Proceedings/papersfinal/p8.pdf.

[21] A. Bertolino and R. Mirandola, CB-SPE tool: putting component-based performance engineering into practice, in *Proceedings of the 7th International Symposium on Component-Based Software Engineering*, CBSE '04, pp. 233–248, Springer, 2004, URL http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.103.9563&rep=rep1&type=pdf.

[22] A. Bertolino and R. Mirandola, Software performance engineering of component-based systems, in *Proceedings of the 4th international workshop on Software and performance*, WOSP '04, pp. 238–242, ACM, New York, NY, USA, 2004, ISBN 1-58113-673-0, URL http://doi.acm.org/10.1145/974044.974081.

[23] J. P. Buzen, Computational algorithms for closed queueing networks with exponential servers, *Commun. ACM*, vol. 16: pp. 527–531, September 1973, ISSN 0001-0782, URL http://doi.acm.org/10.1145/362342.362345.

[24] S. Chakraborty, S. Künzli, L. Thiele, A. Herkersdorf, and P. Sagmeister, Performance evaluation of network processor architectures: combining simulation with analytical estimation, *Comput. Netw.*, vol. 41: pp. 641–665, April 2003, ISSN 1389-1286, URL http://portal.acm.org/citation.cfm?id=765784.765791.

[25] K. M. Chandy and D. Neuse, Linearizer: a heuristic algorithm for queueing network models of computing systems, *Commun. ACM*, vol. 25: pp. 126–134, February 1982, ISSN 0001-0782, URL http://doi.acm.org/10.1145/358396.358403.

[26] C. Chiṣe and I. Jurca, Improving distributed systems design by simulation-based performance analysis. A case study., *BS-UPT TACCS*, vol. 53(67): pp. 157–164, September 2008.

[27] C. Chișe and I. Jurca, Simulation-based performance evaluation of distributed software systems. A case study., in *8th International Conference on Technical Informatics, CONTI'2008, June 5-6, Vol. 2 Computer and Software Engineering*, CONTI '08, pp. 85–90, 2008.

[28] C. Chișe and I. Jurca, Towards early performance assessment based on UML MARTE models for distributed systems, in *5th International Symposium on Applied Computational Intelligence and Informatics, SACI 2009, May 28-29*, SACI '09, pp. 521–526, IEEE, 2009, URL http://ieeexplore.ieee.org/Xplore/login.jsp?url=http%3A%2F%2Fieeexplore.ieee.org%2Fiel5%2F5089334%2F5136205%2F05136304.pdf%3Farnumber%3D5136304&authDecision=-203.

[29] C. Chișe and I. Jurca, Automation of performance assessment applied to UML MARTE models for distributed systems, *BS-UPT TACCS*, vol. 55(69): pp. 59–66, June 2010.

[30] C. Chișe and I. Jurca, Phymss: performance hybrid model solver and simulator based on UML MARTE diagrams, in *Proceedings of the first joint WOSP/SIPEW international conference on Performance engineering*, WOSP/SIPEW '10, pp. 243–244, ACM, New York, NY, USA, 2010, ISBN 978-1-60558-563-5, URL http://doi.acm.org/10.1145/1712605.1712643.

[31] C. Chișe and I. Jurca, Hybrid analytical-simulation model used to evaluate and improve system performance, in *Proceedings of the 10th International Symposium on Parallel and Distributed Computing (ISPDC 2011)*, ISPDC 2011, 2011.

[32] C. Chișe and I. Jurca, Performance prediction for UML MARTE models with a hybrid model solver, in *Proceedings of the 6th International Conference Days of the Academy of Technical Science from Romania (Zilele ASTR 2011), vol. 1*, Zilele ASTR 2011, pp. 116–121, Editura Politehnica, Bd. Republicii, Nr. 9, 300159 Timișoara, 2011, ISSN 2066–6586.

[33] V. Cortellessa, A. Di Marco, P. Inverardi, F. Mancinelli, and P. Pelliccione, A Framework for the Integration of Functional and Non-functional Analysis of Software Architectures, *Electron. Notes Theor. Comput. Sci.*, vol. 116: pp. 31–44, January 2005, ISSN 1571-0661, URL http://dx.doi.org/10.1016/j.entcs.2004.02.088.

[34] V. Cortellessa and L. Frittella, A framework for automated generation of architectural feedback from software performance analysis, in *Proceedings of the 4th European performance engineering conference on Formal methods and stochastic models for performance evaluation*, EPEW'07, pp. 171–185, Springer-Verlag, Berlin, Heidelberg, 2007, ISBN 3-540-75210-2, 978-3-540-75210-3, URL http://portal.acm.org/citation.cfm?id=1779905.1779923.

[35] V. Cortellessa and L. Frittella, A framework for automated generation of architectural feedback from software performance analysis, Tech. rep., Dipartimento di Informatica, Universitá degli Studi dell'Aquila, L'Aquila, Italy, 2007.

[36] V. Cortellessa, M. Gentile, and M. Pizzuti, XPRIT: An XML-Based Tool to Translate UML Diagrams into Execution Graphs and Queueing Networks, in *Proceedings of the The Quantitative Evaluation of Systems, First International Conference*, pp. 342–343, IEEE Computer Society, Washington, DC, USA, 2004, ISBN 0-7695-2185-1, URL http://portal.acm.org/citation.cfm?id=1025129.1026113.

[37] V. Cortellessa and R. Mirandola, Deriving a queueing network based performance model from UML diagrams, in *Proceedings of the 2nd international workshop on Software and performance*, WOSP '00, pp. 58–70, ACM, New York, NY, USA, 2000, ISBN 1-58113-195-X, URL http://doi.acm.org/10.1145/350391.350406.

[38] V. Cortellessa and R. Mirandola, PRIMA-UML: a performance validation incremental methodology on early UML diagrams, *Sci. Comput. Program.*, vol. 44: pp. 101–129, July 2002, ISSN 0167-6423, URL http://portal.acm.org/citation.cfm?id=607036.607042.

[39] V. Cortellessa, P. Pierini, and D. Rossi, Integrating Software Models and Platform Models for Performance Analysis, *IEEE Trans. Softw. Eng.*, vol. 33: pp. 385–401, June 2007, ISSN 0098-5589, URL http://portal.acm.org/citation.cfm?id=1263152.1263532.

[40] V. Cortellessa, P. Pierini, R. Spalazzese, and A. Vianale, MOSES: MOdeling Software and platform architEcture in UML 2 for Simulation-based performance analysis, in *Proceedings of the 4th International Conference on Quality of Software-Architectures: Models and Architectures*, QoSA '08, pp. 86–102, Springer-Verlag, Berlin, Heidelberg, 2008, ISBN 978-3-540-87878-0, URL http://dx.doi.org/10.1007/978-3-540-87879-7_6.

[41] A. D'Ambrogio, A model transformation framework for the automated building of performance models from UML models, in *Proceedings of the 5th international workshop on Software and performance*, WOSP '05, pp. 75–86, ACM, New York, NY, USA, 2005, ISBN 1-59593-087-6, URL http://doi.acm.org/10.1145/1071021.1071029.

[42] P. J. Denning and J. P. Buzen, The Operational Analysis of Queueing Network Models, *ACM Comput. Surv.*, vol. 10: pp. 225–261, September 1978, ISSN 0360-0300, URL http://doi.acm.org/10.1145/356733.356735.

[43] A. Deshpande, V. Apte, and S. Marathe, PerfCenter: a performance modeling tool for application hosting centers, in *Proceedings of the 7th international workshop on Software and performance*, WOSP '08, pp. 79–90, ACM, New York, NY, USA, 2008, ISBN 978-1-59593-873-2, URL http://doi.acm.org/10.1145/1383559.1383570.

[44] N. van Dijk, E. van der Sluis, R. Haijema, A. Al-Ibrahim, and J. van der Wal, Simulation and or (operations research) in combination for practical optimization, in *Proceedings of the 37th conference on Winter simulation*, WSC '05, pp. 274–284, Winter Simulation Conference, 2005, ISBN 0-7803-9519-0, URL http://portal.acm.org/citation.cfm?id=1162708.1162760.

[45] N. M. van Dijk, On hybrid combination of queueing and simulation, in *Proceedings of the 32nd conference on Winter simulation*, WSC '00, pp. 147–150, Society for Computer Simulation International, San Diego, CA, USA, 2000, ISBN 0-7803-6582-8, URL http://portal.acm.org/citation.cfm?id=510378.510403.

[46] F. Duarte, W. Hasling, W. Sherman, D. Paulish, R. M. Leão, E. S. Silva, and V. Cortellessa, Extending model transformations in the performance domain with a node modeling library, in *Proceedings of the 7th international workshop on Software and performance*, WOSP '08, pp. 157–164, ACM, New York, NY, USA, 2008, ISBN 978-1-59593-873-2, URL http://doi.acm.org/10.1145/1383559.1383580.

[47] P. J. Fortier and H. Michel, *Computer Systems Performance Evaluation and Prediction*, Butterworth-Heinemann, Newton, MA, USA, 2002, ISBN 1555582605.

[48] G. Franks, *Performance Analysis of Distributed Server Systems*, Ph.D. thesis, Department of Systems and Computer Engineering, Carleton University, Ottawa, Canada, 1999.

[49] G. Franks, P. Maly, M. Woodside, D. C. Petriu, and A. Hubbard, *Layered Queueing Network Solver and Simulator User Manual*, Department of Systems and Computer Engineering, Carleton University, Ottawa.

[50] H. Gomaa and D. A. Menascé, Design and performance modeling of component interconnection patterns for distributed software architectures, in *Proceedings of the 2nd international workshop on Software and performance*, WOSP '00, pp. 117–126, ACM, New York, NY, USA, 2000, ISBN 1-58113-195-X, URL http://doi.acm.org/10.1145/350391.350418.

[51] H. Gomaa and D. A. Menascé, Performance Engineering of Component-Based Distributed Software Systems, in *Performance Engineering, State of the Art and Current Trends*, pp. 40–55, Springer-Verlag, London, UK, 2001, ISBN 3-540-42145-9, URL http://portal.acm.org/citation.cfm?id=647640.733375.

[52] V. Grassi and R. Mirandola, Towards automatic compositional performance analysis of component-based systems, in *Proceedings of the 4th international workshop on Software and performance*, WOSP '04, pp. 59–63, ACM, New York, NY, USA, 2004, ISBN 1-58113-673-0, URL http://doi.acm.org/10.1145/974044.974052.

[53] V. Grassi, R. Mirandola, E. Randazzo, and A. Sabetta, The Common Component Modeling Example, chap. KLAPER: An Intermediate Language for Model-Driven Predictive Analysis of Performance and Reliability, pp. 327–356, Springer-Verlag, Berlin, Heidelberg, 2008, ISBN 978-3-540-85288-9, URL http://dx.doi.org/10.1007/978-3-540-85289-6_13.

[54] V. Grassi, R. Mirandola, and A. Sabetta, Filling the gap between design and performance/reliability models of component-based systems: A model-driven approach, *J. Syst. Softw.*, vol. 80: pp. 528–558, April 2007, ISSN 0164-1212, URL http://portal.acm.org/citation.cfm?id=1225950.1226107.

[55] G. P. Gu and D. C. Petriu, XSLT transformation from UML models to LQN performance models, in *Proceedings of the 3rd international workshop on Software and performance*, WOSP '02, pp. 227–234, ACM, New York, NY, USA, 2002, ISBN 1-58113-563-7, URL http://doi.acm.org/10.1145/584369.584402.

[56] G. P. Gu and D. C. Petriu, From UML to LQN by XML algebra-based model transformations, in *Proceedings of the 5th international workshop on Software and performance*, WOSP '05, pp. 99–110, ACM, New York, NY, USA, 2005, ISBN 1-59593-087-6, URL http://doi.acm.org/10.1145/1071021.1071031.

[57] A. Hennig, A. Hentschel, and J. Tyck, Performance Prototyping – Generating and Simulating a Distributed IT System from UML Models, in *Proceedings of the 17th European Simulation Multiconference*, ESM '03, pp. 502–508, 2003.

[58] A. Hennig, D. Revill, and M. Ponitsch, From UML to Performance Measures – Simulative Performance Predictions of IT Systems using the Jboss Application Server with OMNET++, in *Proceedings of the 17th European Simulation Multiconference*, ESM '03, pp. 509–513, 2003.

[59] E. Ignall and P. Kolesar, On Using Simulation to Extend OR/MS Theory: The Symbiosis of Simulation and Analysis, *Current Issues in Computer Simulation*, pp. 223–233, 1979.

[60] P. Inverardi and A. L. Wolf, Formal Specification and Analysis of Software Architectures Using the Chemical Abstract Machine Model, *IEEE Trans. Softw. Eng.*, vol. 21: pp. 373–386, April 1995, ISSN 0098-5589, URL http://dx.doi.org/10.1109/32.385973.

[61] A. C. Johnson and N. T. Thomopoulos, Characteristics and Tables of the Left-Truncated Normal Distribution, in *Proceedings of the Midwest Decision Sciences Institute*, pp. 133–139, 2002, URL http://www.stuart.iit.edu/shared/shared_stuartfaculty/whitepapers/thomopoulos_char-left.pdf.

[62] P. Kahkipuro, UML-Based Performance Modeling Framework for Component-Based Distributed Systems, in *Performance Engineering, State of the Art and Current Trends*, pp. 167–184, Springer-Verlag, London, UK, 2001, ISBN 3-540-42145-9, URL http://portal.acm.org/citation.cfm?id=647640.733374.

[63] P. J. B. King and R. Pooley, Derivation of Petri Net Performance Models from UML Specifications of Communications Software, in *Proceedings of the 11th International Conference on Computer Performance Evaluation: Modelling Techniques and Tools*, TOOLS '00, pp. 262–276, Springer-Verlag, London, UK, 2000, ISBN 3-540-67260-5, URL http://portal.acm.org/citation.cfm?id=647809.737825.

[64] C. Kirkegaard, *Dynamic XML Processing with Static Validation*, Master's thesis, Department of Computer Science, University of Aarhus, Denmark, 2003.

[65] C. Kirkegaard, A. Mller, and M. I. Schwartzbach, Static Analysis of XML Transformations in Java, *IEEE Trans. Softw. Eng.*, vol. 30: pp. 181–192, March 2004, ISSN 0098-5589, URL http://portal.acm.org/citation.cfm?id=972216.972292.

[66] N. Knaak and B. Page, Applications and extensions of the Unified Modeling Language UML 2 for discrete event simulation, *International Journal of Simulation*, vol. 7: pp. 33–43, September 2006, ISSN 1473-8031, URL http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.112.4746&rep=rep1&type=pdf.

[67] H. Koziolek, S. Becker, R. Reussner, and J. Happe, Model-Driven Software Development: Integrating Quality Assurance, chap. Evaluating Performance of Software Architecture Models with the Palladio Component Model, pp. 95–118, 2009, ISBN 9781605660066.

[68] E. D. Lazowska, J. Zahorjan, G. S. Graham, and K. C. Sevcik, *Quantitative system performance: computer system analysis using queueing network models*, Prentice Hall, 1984, URL http://www.cs.washington.edu/homes/lazowska/qsp/.

[69] Y. H. Lee and S. H. Kim, Optimal productiondistribution planning in supply chain management using a hybrid simulation-analytic approach, in *Proceedings of the 32nd conference on Winter simulation*, WSC '00, pp. 1252–1259, Society for Computer Simulation International, San Diego, CA, USA, 2000, ISBN 0-7803-6582-8, URL http://portal.acm.org/citation.cfm?id=510378.510559.

[70] C. Lindemann, A. Thümmler, A. Klemm, M. Lohmann, and O. P. Waldhorst, Performance analysis of time-enhanced UML diagrams based on stochastic processes, in *Proceedings of the 3rd international workshop on Software and performance*, WOSP '02, pp. 25–34, ACM, New York, NY, USA, 2002, ISBN 1-58113-563-7, URL http://doi.acm.org/10.1145/584369.584375.

[71] J. P. López-Grao, J. Merseguer, and J. Campos, From UML activity diagrams to Stochastic Petri nets: application to software performance engineering, in *Proceedings of the 4th international workshop on Software and performance*, WOSP '04, pp. 25–36, ACM, New York, NY, USA, 2004, ISBN 1-58113-673-0, URL http://doi.acm.org/10.1145/974044.974048.

[72] M. Marzolla, libcppsim: A SIMULA-like, Portable Process-Oriented Simulation Library in C++, in G. Horton, ed., *Proceedings of the 18th European Simulation Multiconference, First International Conference*, ESM '04, SCS Europe, 2004.

[73] M. Marzolla, *Simulation-Based Performance Modeling of UML Software Architectures*, Ph.D. thesis, Dipartimento di Informatica, Universitá Ca Foscari di Venezia, Italy, 2004.

[74] M. Marzolla and S. Balsamo, UML-PSI: The UML Performance Simulator, Tech. rep., Dipartimento di Informatica, Universitá Ca Foscari di Venezia, Venice, Italy, 2004.

[75] M. Marzolla and S. Balsamo, UML-PSI: The UML Performance Simulator, in *Proceedings of the The Quantitative Evaluation of Systems, First International Conference*, pp. 340–341, IEEE Computer Society, Washington, DC, USA, 2004, ISBN 0-7695-2185-1, URL http://portal.acm.org/citation.cfm?id=1025129.1026112.

[76] D. McMullan, Components in Layered Queuing Networks, Tech. rep., Department of Systems and Computer Engineering, Carleton University, Ottawa, Canada, 2001, URL http://www.sce.carleton.ca/rads/lqns/lqn-documentation/component3.pdf.

[77] D. A. Menascé, V. A. F. Almeida, and L. W. Dowdy, *Performance by Design: Computer Capacity Planning by Example*, Prentice Hall, Upper Saddle River, New Jersey, USA, 1st ed., 2004, ISBN 0-13-090673-5.

[78] J. Merseguer and J. Campos, Exploring Roles for the UML Diagrams in Software Performance Engineering., in *Software Engineering Research and Practice'03*, pp. 43–47, 2003.

[79] Microsoft, Introduction to WPF, http://msdn.microsoft.com/en-us/library/aa970268.aspx, URL http://msdn.microsoft.com/en-us/library/aa970268.aspx.

[80] Z. Micskei and H. Waeselynck, A survey of UML 2.0 sequence diagrams' semantics, Tech. rep., Budapest University of Technology and Economics, Université de Toulouse, 2008, URL http://home.mit.bme.hu/~micskeiz/sdreport/uml-sd-semantics.pdf.

[81] M. de Miguel, T. Lambolais, M. Hannouz, S. Betgé-Brezetz, and S. Piekarec, UML extensions for the specification and evaluation of latency constraints in architectural models, in *Proceedings of the 2nd international workshop on Software and performance*, WOSP '00, pp. 83–88, ACM, New York, NY, USA, 2000, ISBN 1-58113-195-X, URL http://doi.acm.org/10.1145/350391.350411.

[82] G. Nemes, New asymptotic expansion for the $\gamma(x)$ function, December 2008, URL http://dx.doi.org/10.3247/sl2math08.005.

[83] T. Omari, G. Franks, M. Woodside, and A. Pan, Solving layered queueing networks of large client-server systems with symmetric replication, in *Proceedings of the 5th international workshop on Software and performance*, WOSP '05, pp. 159–166, ACM, New York, NY, USA, 2005, ISBN 1-59593-087-6, URL http://doi.acm.org/10.1145/1071021.1071038.

[84] O. M. G. (OMG), Introduction to OMG's Unified Modeling Language (UML), URL http://www.omg.org/gettingstarted/what_is_uml.htm.

[85] O. M. G. (OMG), MARTE Tutorial - Part 8: MARTE and AADL, URL http://www.omgmarte.org/node/28/.

[86] O. M. G. (OMG), Meta Object Facility (MOF) 2.0 Query/View/Transformation (QVT), URL http://www.omg.org/spec/QVT/.

[87] O. M. G. (OMG), Meta Object Facility (MOF) Core, URL http://www.omg.org/spec/MOF/.

[88] O. M. G. (OMG), OMG Model Driven Architecture, URL http://www.omg.org/mda/.

[89] O. M. G. (OMG), UML Profile for Modeling and Analysis of Real-time and Embedded Systems, URL http://www.omg.org/spec/MARTE/1.0/.

[90] O. M. G. (OMG), UML Profile for Modeling Quality of Service and Fault Tolerance Characteristics and Mechanisms, URL http://www.omg.org/spec/QFTP/1.1/.

[91] O. M. G. (OMG), UML Profile for Schedulability, Performance and Time, URL http://www.omg.org/spec/SPTP/.

[92] O. M. G. (OMG), UML Version 2.0 Documents, URL http://www.omg.org/spec/UML/2.0/.

[93] O. M. G. (OMG), XML Metadata Interchange (XMI), URL http://www.omg.org/spec/XMI/.

[94] D. B. Petriu, *Layered Software Performance Models Constructed from Use Case Map Specifications*, Master's thesis, Department of Systems and Computer Engineering, Carleton University, Ottawa, Canada, 2001.

[95] D. B. Petriu and C. M. Woodside, Software Performance Models from System Scenarios in Use Case Maps, in *Proceedings of the 12th International Conference on Computer Performance Evaluation, Modelling Techniques and Tools*, TOOLS '02, pp. 141–158, Springer-Verlag, London, UK, 2002, ISBN 3-540-43539-5, URL http://portal.acm.org/citation.cfm?id=647810.737983.

[96] D. B. Petriu and M. Woodside, Generating a performance model from a design specification, August 2001, URL ftp://ftp.sce.carleton.ca/pub/cmw/gen-perf.pdf.

[97] D. B. Petriu and M. Woodside, A metamodel for generating performance models from UML designs, *UML 2004 The Unified Modeling Language: Modeling Languages and Applications 7th International Conference*, (3273): pp. 41–53, 2004, URL http://www.springerlink.com/index/D92HHDB5A21VBDUX.pdf.

[98] D. B. Petriu and M. Woodside, An intermediate metamodel with scenarios and resources for generating performance models from UML designs, *Software and Systems Modeling*, vol. 6: pp. 163–184, 2007, ISSN 1619-1366, URL http://dx.doi.org/10.1007/s10270-006-0026-8, 10.1007/s10270-006-0026-8.

[99] D. C. Petriu, Performance Analysis Based on the UML SPT Profile, 2004.

[100] D. C. Petriu, Software Model-based Performance Analysis, in *Post-Proceedings of the MDD4DRES Summer School*, Hermes Science Publishing Ltd., London, UK, 2009, URL http://portal.acm.org/citation.cfm?id=647810.737982.

[101] D. C. Petriu and H. Shen, Applying the UML Performance Profile: Graph Grammar-Based Derivation of LQN Models from UML Specifications, in *Proceedings of the 12th International Conference on Computer Performance Evaluation, Modelling Techniques and Tools*, TOOLS '02, pp. 159–177, Springer-Verlag, London, UK, 2002, ISBN 3-540-43539-5, URL http://portal.acm.org/citation.cfm?id=647810.737982.

[102] D. C. Petriu and X. Wang, Deriving Software Performance Models from Architectural Patterns by Graph Transformations, in *Selected papers from the 6th International Workshop on Theory and Application of Graph Transformations*, TAGT'98, pp. 475–488, Springer-Verlag, London, UK, 2000, ISBN 3-540-67203-6, URL http://portal.acm.org/citation.cfm?id=645872.671531.

[103] C. Rad, BS Degree Project: C# Library for Process-Oriented Simulation, 2009.

[104] M. Reiser and S. S. Lavenberg, Mean-Value Analysis of Closed Multichain Queuing Networks, *J. ACM*, vol. 27: pp. 313–322, April 1980, ISSN 0004-5411, URL http://doi.acm.org/10.1145/322186.322195.

[105] R. G. Sargent, A historical view of hybrid simulation/analytic models, in *Proceedings of the 26th conference on Winter simulation*, WSC '94, pp. 383–386, Society for Computer Simulation International, San Diego, CA, USA, 1994, ISBN 0-7803-2109-X, URL http://portal.acm.org/citation.cfm?id=193201.194085.

[106] R. G. Sargent, Verification and validation of simulation models, in *Proceedings of the 41th Conference on Winter Simulation*, WSC '09, pp. 162–176, Winter Simulation Conference, 2009, ISBN 978-1-4244-5771-7, URL http://www.informs-sim.org/wsc09papers/014.pdf.

[107] H. S. Sarjoughian, D. Huang, G. W. Godding, K. G. Kempf, W. Wang, D. E. Rivera, and H. D. Mittelmann, Hybrid discrete event simulation with model predictive control for semiconductor supply-chain manufacturing, in *Proceedings of the 37th conference on Winter simulation*, WSC '05, pp. 256–266, Winter Simulation Conference, 2005, ISBN 0-7803-9519-0, URL http://portal.acm.org/citation.cfm?id=1162708.1162757.

[108] J. G. Shanthikumar and R. G. Sargent, A Unifying View of Hybrid Simulation/Analytic Models and Modeling, *OPERATIONS RESEARCH*, vol. 31 (6): pp. 1030–1052, 1983, URL http://or.journal.informs.org/cgi/content/abstract/31/6/1030.

[109] E. d. S. e Silva, A. P. C. da Silva, A. A. de A. Rocha, R. M. M. Leão, F. P. Duarte, F. J. S. Filho, G. D. G. Jai, and R. R. Muntz, Modeling, analysis, measurement and experimentation with the Tangram-II integrated environment, in *Proceedings of the 1st international conference on Performance evaluation methodologies and tools*, valuetools '06, ACM, New York, NY, USA, 2006, ISBN 1-59593-504-5, URL http://doi.acm.org/10.1145/1190095.1190103.

[110] P. P. d. Silva, A. H. F. Laender, and P. B. Golgher, A Simulation Model for the Performance Evaluation when Migrating Legacy Systems, in *Proceedings of the Fifth European Conference on Software Maintenance and Reengineering*, CSMR '01, pp. 210–215, IEEE Computer Society, Washington, DC, USA, 2001, ISBN 0-7695-1028-0, URL http://portal.acm.org/citation.cfm?id=794203.795298.

[111] C. U. Smith, *Performance Engineering of Software Systems*, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st ed., 1990, ISBN 0201537699.

[112] C. U. Smith and C. M. Lladó, Performance Model Interchange Format (PMIF 2.0): XML Definition and Implementation, in *Proceedings of the The Quantitative Evaluation of Systems, First International Conference*, pp. 38–47, IEEE Computer Society, Washington, DC, USA, 2004, ISBN 0-7695-2185-1, URL http://portal.acm.org/citation.cfm?id=1025129.1026073.

[113] C. U. Smith, C. M. Lladó, V. Cortellessa, A. D. Marco, and L. G. Williams, From UML models to software performance results: an SPE process based on XML interchange formats, in *Proceedings of the 5th international workshop on Software and performance*, WOSP '05, pp. 87–98, ACM, New York, NY, USA, 2005, ISBN 1-59593-087-6, URL http://doi.acm.org/10.1145/1071021.1071030.

[114] C. U. Smith and L. Williams, *Performance Solutions: A Practical Guide to Creating Responsive, Scalable Software*, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st ed., 2001, ISBN 0-201-72229-1.

[115] C. U. Smith and L. Williams, Best Practices for Software Performance Engineering, in *Proc. CMG*, 2003, URL http://www.perfeng.com/papers/bestprac.pdf.

[116] C. U. Smith and L. G. Williams, A performance model interchange format, *J. Syst. Softw.*, vol. 49: pp. 63–80, December 1999, ISSN 0164-1212, URL http://portal.acm.org/citation.cfm?id=340287.340332.

[117] R. Suri, S. Sahu, and M. Vernon, Approximate Mean Value Analysis for Closed Queuing Networks with Multiple-Server Stations, in *Proceedings of the 2007 Industrial Engineering Research Conference*, IERC'07, 2007.

[118] M. Tribastone and S. Gilmore, Automatic extraction of PEPA performance models from UML activity diagrams annotated with the MARTE profile, in *Proceedings of the 7th international workshop on Software and performance*, WOSP '08, pp. 67–78, ACM, New York, NY, USA, 2008, ISBN 978-1-59593-873-2, URL http://doi.acm.org/10.1145/1383559.1383569.

[119] T. Verdickt, B. Dhoedt, F. De Turck, and P. Demeester, Hybrid performance modeling approach for network intensive distributed software, in *Proceedings of the 6th international workshop on Software and performance*, WOSP '07, pp. 189–200, ACM, New York, NY, USA, 2007, ISBN 1-59593-297-6, URL http://doi.acm.org/10.1145/1216993.1217026.

[120] R. P. Verlekar, V. Apte, P. Goyal, and B. Agarwal, PerfCenter: A Methodology and Tool for Performance Analysis of Application Hosting Centers, in *Proceedings of the 2007 15th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*, pp. 201–208, IEEE Computer Society, Washington, DC, USA, 2007, ISBN 978-1-4244-1854-1, URL http://portal.acm.org/citation.cfm?id=1474555.1475491.

[121] R. Wagh, U. Bellur, and B. Menezes, Transformation of UML design model into performance model: a model-driven framework, in *Proceedings of the the 8th International Conference on Enterprise Information Systems (ICEIS)*, ICEIS (3) 2006, pp. 576–580, 2006, URL http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.108.8326&rep=rep1&type=pdf.

[122] J. Westall and R. Geist, A hybrid tool for the performance evaluation of NUMA architectures, in *Proceedings of the 29th conference on Winter simulation*, WSC '97, pp. 1029–1036, IEEE Computer Society, Washington, DC, USA, 1997, ISBN 0-7803-4278-X, URL http://dx.doi.org/10.1145/268437.268736.

[123] L. G. Williams and C. U. Smith, Performance evaluation of software architectures, in *Proceedings of the 1st international workshop on Software and performance*, WOSP '98, pp. 164–177, ACM, New York, NY, USA, 1998, ISBN 1-58113-060-0, URL http://doi.acm.org/10.1145/287318.287353.

[124] S. Winitzki, A handy approximation for the error function and its inverse, February 2008, URL http://issuu.com/julianprice/docs/erf-approx.

[125] M. Woodside and G. Franks, Tutorial Introduction to Layered Modeling of Software Performance, May 2002, URL http://sce.carleton.ca/rads/lqns/lqn-documentation/tutorialg.pdf.

[126] M. Woodside, D. C. Petriu, D. B. Petriu, H. Shen, T. Israr, and J. Merseguer, Performance by unified model analysis (PUMA), in *Proceedings of the 5th international workshop on Software and performance*, WOSP '05, pp. 1–12, ACM, New York, NY, USA, 2005, ISBN 1-59593-087-6, URL http://doi.acm.org/10.1145/1071021.1071022.

[127] X. Wu, D. McMullan, and M. Woodside, Component Based Performance Prediction, in *Proceedings of the6th ICSE Workshop on Component-Based Software Engineering*, CBSE '03, Carnegie Mellon University, USA, and Monash University, Australia, 2003, URL http://www.csse.monash.edu.au/~hws/cgi-bin/CBSE6/Proceedings/papersfinal/p24.pdf.

[128] J. Xu, M. Woodside, and D. C. Petriu, Computer Performance Evaluation. Modelling Techniques and Tools, Lecture Notes in Computer Science, chap. Performance Analysis of a Software Design Using the UML Profile for Schedulability, Performance, and Time, pp. 291–307, Springer Berlin / Heidelberg, 2003.