A Hybrid Approach to Performance Evaluation of Distributed Systems

Ph.D. Report #1

Ph.D. Student: Cosmina CHIȘE
Scientific Advisor: Prof. Dr. Ioan JURCA

October 2009
## Contents

1. INTRODUCTION ................................................................................................................................................. 5

2. RELATED WORK ................................................................................................................................................. 7
   2.1. SPE Process EVOLUTION ............................................................................................................................... 7
   2.2. LQN METHODOLOGY AND MODEL SOLVERS ............................................................................................. 13
   2.3. METHODOLOGIES AND TOOLS FOR COMPONENT-BASED SYSTEMS ..................................................... 17
   2.4. UML 2 OR MARTE-COMPLIANT METHODOLOGIES AND TOOLS .......................................................... 17

3. HYBRID META-MODEL ..................................................................................................................................... 19
   3.1. META-MODEL DEFINITION .......................................................................................................................... 19
   3.2. MODEL EXTRACTION ...................................................................................................................................... 23
   3.3. MODEL SOLVING ........................................................................................................................................ 24

4. PHYMSS TOOL.................................................................................................................................................. 29

5. CASE STUDY. PERFORMANCE ANALYSIS USING PHYMSS.................................................................................. 33
   5.1. INPUT UML MODEL ..................................................................................................................................... 33
   5.2. PERFORMANCE ANALYSIS RESULTS ......................................................................................................... 35
   5.3. HYBRID METHOD EVALUATION ................................................................................................................... 35

6. CONCLUSIONS AND FUTURE WORK ............................................................................................................... 37
1. INTRODUCTION

While developing a software system, the focus is usually on the functional aspects and performance is neglected until later stages: it is measured during system validation. For regular, standalone software systems this is a correct approach, since only rarely performance becomes an issue. But when it does, the problem usually comes from system design; redesign leads to high costs, especially in late lifecycle stages, and the projects that encounter such problems may simply be dropped. For this reason, in case of real-time or distributed systems, where performance issues are likely to occur due to the environment in which they operate and where meeting deadlines or user-perceived delays are important, performance should be addressed as early as possible in their lifecycle, helping developers make informed design decisions.

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, such as response time, throughput and resource utilization. Complete information 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, only the number of requests and the performance parameters for steps needed to provide service are important.

There are several research directions in Software Performance Engineering (SPE), covering the entire performance prediction process, from input language, model extraction, to model solvers or simulators. However, there are few tools that actually cover the whole performance evaluation process, many tools implement only part of it or have restrictions.

Current performance prediction tools rely either on analytical or simulation models. Analytical models are mathematical representations of the system, solved in order to obtain performance parameters; simulation models are executable systems having a similar behavior to the original system, which are run in order to measure the performance results. Analytical approaches are fast, but not accurate, since they usually need simplifying assumptions and 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 in order to obtain relevant mean values for parameters.

Performance Engineering is a rather recent research field, based on model-driven development. Analytical and simulation models have been extensively studied, so the theoretical background is well-defined. However, standardization of description languages for these system models is still ongoing. UML (Unified Modeling Language) has been widely adopted as a
A Hybrid Approach to Performance Evaluation of Distributed Systems

universal modeling language, and extensions for specific purposes have been defined; performance related extensions are defined using two different UML profiles, SPT (Schedulability, Performance and Time) [28] and QoS&FT (Quality of Service and Fault Tolerance) [29], unified in the more comprehensive MARTE (Modeling and Analysis of Real Time and Embedded systems) profile [30].

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, since they have been developed only as proof-of-concept tools in order to support various modeling and performance evaluation techniques and verify their effectiveness.

This report presents the steps, theoretical and experimental, performed to define a hybrid methodology and to implement it in Phymss (Performance Hybrid Model Solver and Simulator), a tool that intends to encompass as much as possible from the performance analysis process. The tool accepts XMI (XML Metadata Interchange) [32] 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 Layered Queueing Network (LQN) model during the analytical solving process. Performance results are inserted into the UML model and can be exported as an XMI file. The tool and the underlying methodology have been the subject of a paper [6] that is accepted for publishing in the 1st Joint WOSP/SIPEW International Conference on Performance Engineering, which will be held in San Jose, California, USA, in January 2010.

In order to provide the background for notations and methodology regarding SPE and analytical model analysis, and also briefly review trends in tool development in the last few years, chapter 2 presents an overview of the main research directions and current developments that were not treated in the Ph.D. Project Proposal. In the following chapters, the personal contributions in the domain that have been addressed are described. A first personal contribution (chapter 3) is the definition of a performance hybrid meta-model, that relies on the UML MARTE profile. This model is the basis for the development of a performance analysis tool, Phymss, presented in chapter 4. Chapter 5 is a case study, which shows how the tool is used, and also evaluates the hybrid alternative by comparison to a pure simulation approach. Finally, in chapter 6, conclusions and future work directions are presented.
2. RELATED WORK

Two directions were set during the last decade in Software Performance Engineering (SPE): one initiated by C. Smith and L. Williams and developed by V. Cortellessa which covers the whole performance analysis process, starting from UML diagram parsing and ending with feedback mechanisms (result interpretation); another research direction led by D.C. Petriu, D.B. Petriu and M. Woodside focused on modeling system performance (including platform details usually) by using Queuing Networks (QN) and defining improved extensions such as Layered Queuing Networks, for which a comprehensive solver was developed by G. Franks in his Ph.D. Thesis [12].

In the last few years, a new direction emerged from SPE focusing on analysis of component-based systems; these techniques and tools are briefly reviewed.

In order to cover the latest performance annotations, UML 2 and MARTE related techniques and tools are presented in the last paragraph of this section.

2.1. SPE Process Evolution

A methodology to automatically extract a performance model from UML diagrams is defined in [10]. The most difficult part of the SPE process is considered to be the definition of a software model, because platform models can be extracted by other tools.

The performance model is based on Queueing Networks (QN) and can be solved by existing tools that implement numerical solutions. 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 the paper, 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.

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 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.

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 [7]. 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 1. 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 1. NFP integration framework architecture [7]

Starting from the SPE (Software Performance Engineering) approach, defined by C.U. Smith [24], which divides performance analysis in two phases – software execution model
RELATED WORK

analysis and system execution model analysis – a tool called XPRIT [9] was built. 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 2.

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) [22] 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.
A Hybrid Approach to Performance Evaluation of Distributed Systems

- 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 the QNM details required to solve the model, a picture of the model among tools that support this could be exchanged.
- Model results - exchanging results derived from the modeling tool should be also 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 that have 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 3, is used to define the transfer format that enables the exchange of information specified in the meta-model between tools that support the format.

![Queueing Network Meta-Model](image)

Figure 3. Queueing Network Meta-Model as defined in PMIF [22]
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 [23]. Using 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 4. Since the approach involves two phases, the UML model is converted to an S-PMIF model, by using a tool such as XPRIT, and then the S-PMIF model can be transformed into a PMIF model in order to perform system performance model analysis.

Integrating performance analysis within the software lifecycle also assumes interpreting feedback from SPE regarding system architecture or design. A framework that automatically interprets performance output parameters and suggests improvements by identifying performance anti-patterns is described in [8]. The high-level flow chart of the proposed process is shown in Figure 5.

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).
A Hybrid Approach to Performance Evaluation of Distributed Systems

- **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.

![Diagram](image)

Figure 4. The SPE interchange process [23]
2.2. LQN Methodology and Model Solvers

Essentially, an LQN is made of client requests, tasks and hardware devices [20]. 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 6.

Figure 5. Results interpretation and feedback generation process [8]

Figure 6. Layered Queueing Network Example
Franks defines in [12] a methodology to assess performance for client-server systems that cannot be solved by simple Mean Value Analysis. 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, 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 the following:

* Read the input and construct an object database consisting of the tasks, processors and entries and the calls between them.
* Perform processor, forwarding and think-time transformations.
* Generate the layer submodels, and then construct 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:

\[ s_{ml} = \sum_{i \in L, i \neq l} \sum_{j \in S_i} w_{mj} , \]  

(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:

\[ Z_{i,l} = N_{i,l} \cdot \frac{1 - \rho_{i,l-1}}{\lambda_{i,l-1}} , \]

(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 [19]. A specific notation is defined for replicated servers, as shown in Figure 7. The proposed solution relies on the hierarchical approach in solving LQNs and defines an “inner loop” 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.

LQN input description language is extended in [27] 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 8. Parameterization of instances allows for
flexibility regarding multithreaded processing (number of supported threads) or other possible internal differences, while maintaining the common interface for the component class.

Figure 8. Component model for an application server [27]

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, as illustrated in Figure 9.

Figure 9. Component assembly model [27]
2.3. Methodologies and tools 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 a larger system.

A reference paper in the field of component-based SPE is [2] – it presents a methodology and a tool that implements it. The proposed technique describes how to apply SPE analysis to components in the process of component selection and reuse. The tool receives input from Argo UML [33] and the system is modeled as EG and QN, according to the SPE methodology; solvers are integrated for both models.

KLAPER (Kernel LAnguage for PErformance and Reliability analysis) [13] 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) [1] is a domain specific modeling language for component-based software architectures. PCM does not use annotated UML as design model, but defines its own metamodel. This reduces the model to concepts 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.

2.4. 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 [25]. 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 [15] 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 Discrete Event Simulation (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) [11], 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 a software architectural model and a 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.

Since UML 2 and MARTE were recently adopted and are still subject to improvements, there are few analysis tools relying on them. 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 – this is the subject of the following section.
3. HYBRID META-MODEL

Performance meta-models have been defined for analytical solving and simulation, the intention of the ongoing research is to define a hybrid meta-model that should be easy to refine either into an analytical model or into a simulation one. However, only the relevant information from the UML input model should be stored in the performance model, in order to improve analysis efficiency. The outcome of this research activity is building a performance meta-model that combines flexibility and conciseness; its actual purpose is to allow implementation of both analytical methods and simulation on each system model based on it.

Regarding existing hybrid models, one has been used to combine software design evaluation with network specific architecture in [26]. The software model (LQN) and network model (NS-2 [34]) are solved iteratively, thus the analysis results are refined.

An iterative algorithm that applies alternatively analytical techniques and simulation ones has been defined for network processor design [3], but the methods are separately defined, have distinct input and output parameters and they are only ordered one after the other. In the following section, a hybrid approach, combining the two approaches within the same algorithm will be described.

In the first subsection, the process of defining the meta-model is described, the second subsection shows how an instance model can be extracted from input UML models and in the third the proposed solving methodology is presented.

3.1. Meta-model definition

Reference models from both approaches were chosen: the simulation model from UML–Ψ [17] and an intermediate analytical meta-model, which can further be transformed into several types of analytical performance models (such as LQN and STPN) – Core Scenario Model (CSM) [21].

CSM 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 10; it is consistent with MOF (Meta Object Facility) [31].

Scenario flows are described as ordered sequences of steps and PathConnection objects which connect each pair of steps. Sequence objects connect consecutive steps, Branch and Merge objects define alternative steps (ORfork, OR-join), while for parallel activities (AND-fork,
A Hybrid Approach to Performance Evaluation of Distributed Systems

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 could have been used related to network communication, but they were not 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 handle passive resources, including processes or threads in an operating system, hosted by ProcessingResources. Several step types can be distinguished, 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.

Figure 10. Classes in the Core Scenario Model meta-model [21]

An important simulation tool developed by Marzolla as the application part of his PhD thesis [16] is UML-PSI (UML Performance Simulator). UML Performance Simulator is a tool that builds a simulation model and runs it in order to collect performance information. The overall structure of the tool is presented in Figure 11.
Tool input consists in XMI (XML Metadata Interchange [32]) files exported by ArgoUML [33] (or the commercial version Poseidon), a visual tool for defining UML diagrams. The diagrams are annotated with performance information according to the UML SPT profile [28]. UML-Ψ extracts relevant information from the XMI input file and generates a performance process oriented simulation model, based on three main types of entities, corresponding to scenario actions (from Activity diagrams), the resources of the software system (from Deployment diagrams) and the workloads (from Use Case diagrams).

The simulation model execution can be customized from a configuration file, written in Perl, 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. 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 12.
The two selected meta-models are somewhat similar, so the new performance meta-model is derived in a straightforward manner in a research paper [5], written by the author of this report, published this year at the SACI’09 Conference; the meta-model can easily be extracted from system specifications, expressed using the MARTE Profile for UML. An advantage of this meta-model is simplicity, eliminating redundant elements, while maintaining a clear structure for system components and their interactions. An improved version of the meta-model in [5] is presented in Figure 13.

The main three entities are the same: workloads, resources, and steps – 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 ResourceAction items 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 contained by ProcessingSteps, meaning that 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 needed in order to be able to call a scenario within another scenario.

Figure 13. Performance meta-model
3.2. Model extraction

From the multitude of ways to define a system, only specific combinations are allowed in order to provide a meaningful system representation for performance analysis. In the following paragraphs, for each of these sets of diagrams, the process of extracting the performance model will be described, as presented in [5] published in the proceedings of SACI’09.

The resources, both active and passive, are specified by Deployment Diagrams, the ones important for performance analysis should be stereotyped as GaExecHost (GQAM), SchedulableResource (GRM), or just Resource (GRM) – for passive resources. An example of a deployment diagram is presented in Figure 14.

The performance model will include two objects of type ActiveResource: Service1 and App1 (these are software applications); they run on specific platforms (GaExecHost), which may be considered resources too, but they will only be used by the previously mentioned active resources; this is a two-layer deployment.

![Figure 14. Deployment Diagram with two layers](image)

In order to maintain compatibility with models defined in SPT, the tool will accept simplified Deployment Diagrams, such as the one in Figure 15.

In this case, each object stereotyped as GaExecHost will become an ActiveResource, and each object stereotyped as Resource will become a PassiveResource.

In order to add behavior to these resources, there are two possibilities: Use Case Diagram or Sequence Diagram.

A Use Case Diagram connects actors to use cases (behavior). An actor stereotyped WorkloadEvent 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 each step is extracted from the 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.

Figure 15. Deployment Diagram with one layer

A Sequence Diagram is a scenario itself, because it provides successive calls (steps) among instances of resources. Each message stereotyped as PaStep is a step that executes on the resource given by the lifeline towards which it is headed. This 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 (for single-layered deployment). The workload is usually defined by the first message in the sequence, stereotyped as WorkloadEvent.

The complete performance model is obtained by applying the above mentioned mapping rules to diagrams and to the properly stereotyped items and their specific tagged values.

3.3. Model solving

The hybrid solver starts from the two-step iterations in LQNS, and includes a simulation step between them, obtaining a three-step iteration: top-down analytical computations to propagate think time values, a simulation step that measures all parameters from level k downwards, and bottom-up computations, which do not start from the last level (as in the case of LQNS), but from level k and propagate service time values upwards. When these service time values are computed for level 1, the whole system is solved and the iteration is complete. The iterations are stopped either when convergence is obtained for all parameters, or when a predefined number of iterations have been performed.

After performing experiments with simple case studies, the results showed that throughput values rise above normal values for levels approaching the simulation submodel,
when level $k$ is high; otherwise – if $k$ has a low value – simulation starts early and does not propagate estimation errors, which usually occur in MVA. In order to adjust the situation, the formula regarding the propagation of think times has been reevaluated and reduced to:

$$Z_{ij} = \frac{N_{ij}}{\lambda_{i,j-1}}, \quad (2.3)$$

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

The utilization of task $i$, when it is acting as a server in submodel $l-1$, $\rho_{i,j-1}$ is not directly related to think times of requests issued by the server as a client to the lower level servers; indeed, these requests are issued while the server is not busy, but the duration between two consecutive forwarded requests only relies on the rate requests are being handled (the throughput). For two consecutive requests belonging to the same class of requests, that during processing will require requests to lower level servers, the time between requests (each one issued at a specific moment during processing) only depends on the throughput rate with which requests of this particular type are being handled. The mean throughput at server level is computed not by MVA, but by applying a weighted average to the individual throughput rates for each input class, the weights being the probabilities that the input classes issue requests to the server. Another factor that influences the think time is the population count of requests. At each level, it is computed as a weighted average of population counts for each client class on the previous upper level, the weights being the earlier mentioned probabilities. As the number of requests increases, waiting times are larger and this leads to higher values for the think time.

In the pseudo-code for the hybrid solver, presented in Figure 16, the following notations are used:

$Z_c = \{Z_{ij} \mid 1 \leq i \leq c\}$, where $c$ is the number of requests at level $l$, level 1 means pure clients (request generators),

$\rho_t = \{\rho_{ij} \mid 1 \leq i \leq t\}$, where $t$ is the number of server tasks from level $l-1$ that act as clients in level $l$,

$\lambda_t = \{\lambda_{ij} \mid 1 \leq i \leq t\}$, where $t$ is the number of server tasks from level $l-1$ that act as clients in level $l$,

$S_t = \{S_{ij} \mid 1 \leq i \leq t\}$, where $t$ is the number of server tasks from level $l$,

$P_k = \bigcup_{l \in L,l \neq k} \rho_{ij}$, includes all utilization values for tasks on level $k$ and lower levels,

$\Lambda_k = \bigcup_{l \in L,l \neq k} \lambda_{ij}$, includes all throughput values for tasks on level $k$ and lower levels,
A Hybrid Approach to Performance Evaluation of Distributed Systems

\[
T_k = \bigcup_{j \in S_j, m \in S_j, \forall i < k, i \geq 1} [w_{ij}], \text{ includes all response time values for tasks on level } k \text{ and lower levels, which provide service to tasks on higher levels.}
\]

1. Before the actual iterative process, \textit{Cummulate} adds service time values for all servers, without considering contention delays, in order to get an initial estimation for \(S_j, i = 1..k\).

2. Each iteration starts with a call to a random number generator function that follows a given distribution and generates think time values for each request chain. Distributions are provided as request arrival distributions, specified as tagged values in the annotated UML input model.

3. The simulator runs model \(M_{k+1}\), which consists of all submodels starting from level \(k+1\) down to the lowest level, and receives as input the set of propagated think time values.

4. High level submodels \(1..k\) are solved by Mean Value Analysis (MVA) twice per iteration: downwards and upwards. The first call, in the first iteration, relies on the cumulated service times computed in the initial step, but these values are approximations, since contention delays are ignored; afterwards, the computed (analytically) or measured (by simulation) values are propagated.

5. Exact MVA applied for multiple classes of requests can lead to state space explosion. Even for small populations, the complexity of the solution algorithm exceeds manageable boundaries. For this reason, approximate MVA (A-MVA) has been defined [18] and this is the method that has been implemented as the analytical solver, where there are several classes of clients on a certain level (if only one class is present, exact MVA is applied).

6. 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 performed for the simulation step, it has a different meaning: values are collected continuously and sets of values are obtained 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.

Starting from the performance meta-model in Figure 13, additional items are needed in order to allow layered solving: submodels are defined as groups of servers on a particular level and all their clients, regardless of the level the clients belong to. Each submodel will provide an interface that allows both analytical computation of parameters and simulation of its layer and all subsequent layers, below the current level. In order to facilitate working with submodels and groups of items from the complete model, the notion of PerformanceObject is defined – it provides a common interface for all components in the performance model, in order to have easy access to their performance specific parameters, as shown in Figure 17.

![PerformanceObject class diagram](image)

Each object has its own ID, Name and Level; SetLevel is used to propagate level changes among objects while the level values are still changing (they are set recursively by stepping through the performance model). The lists of Clients and CallProbabilities are built recursively too, in the same pass through the hierarchy and are used within submodels at a particular level. Performance parameters, such as ResponseTime, Utilization and Throughput are also present, in order to store intermediate results (before convergence is achieved). PopulationCount and ThinkTime are propagated downwards by calling the method ComputeThinkTime at each level, and ServiceTime is propagated upwards by calling ComputeServiceTime at each level. The Convergent method implements the convergence test for the analytical iterations – it checks the performance results passed as parameters versus the internal values and the differences should be lower than delta.
4. PHYMSS TOOL

Phymss (Performance Hybrid Model Solver and Simulator) [6] is a tool that intends to encompass as much as possible from the performance analysis process and provide flexible analysis options.

The tool accepts XMI (XML Metadata Interchange) files with the UML (Unified Modeling Language) representation of the system model, annotated using the MARTE (Modeling and Analysis of Real Time and Embedded systems) 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 Layered Queueing Network (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 18.

Figure 18. Phymss block diagram [6]
Tool input is represented in XMI format and can be obtained as output from visual design editors for UML diagrams. Papyrus UML [35] is such an editor; it is open-source and supports extensions for the MARTE profile. Simulation and analytical solving parameters, such as duration, confidence interval relative width for 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 changed.

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 a previous section – 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 node will have values specified for parameters such as response time, throughput or utilization.

A partial screenshot of the Phymss main window, with the File menu expanded, is presented in Figure 19. The only option available at startup is Import Parameters. After parameters are imported, the UML model option becomes active. And after the UML model is imported, the Run menu becomes active – it has only two options: Simulation and Hybrid. When choosing the Hybrid option, a window appears that allows selecting the level k for the simulated submodel.

A partial screenshot of the Phymss main window, with the File menu expanded, is presented in Figure 19. The only option available at startup is Import Parameters. After parameters are imported, the UML model option becomes active. And after the UML model is imported, the Run menu becomes active – it has only two options: Simulation and Hybrid. When choosing the Hybrid option, a window appears that allows selecting the level k for the simulated submodel.

After the UML model is imported, another two windows are displayed – one of them allows selection of performance parameters to be monitored during the simulation (the analytical
stage is too fast to be able to display intermediate values) and the other one displays simulation status (selected by default to be displayed) and the selected parameters. No actual values are present until one of the methods is run. These windows with actual values for parameters are illustrated in Figure 20.

Figure 20. Runtime parameter monitoring windows

Phymss and the methodology it implements have been considered relevant and with potential by experts in the field of SPE – the paper describing Phymss [6] was accepted at the 1st Joint WOSP/SIPEW International Conference on Performance Engineering.
5. CASE STUDY. PERFORMANCE ANALYSIS USING PHYMSS

5.1. Input UML Model

A simple UML model was considered in order to show how the method is applied. Only active resources are present, they have a FIFO scheduling policy. Also, the steps involved in activity diagrams contain no loops, in order to have a straightforward conversion from UML diagrams to an LQN model. A more advanced transformation technique is described in [14].

The model depicts a web application that may be used as part of the website for a university. Students can either find out what their grades are or update their profile information (with preferences on what should be displayed when they login). Two database servers are used: one for grades and one for profiles. Two additional servers provide secured access to the information and to the services: a web server and an authentication server. The use case diagram is shown in Figure 21.

The Login and Logout use cases are never called directly by students (their probabilities are zero), since they have no utility by themselves; they are defined as use cases just to group related steps. The most frequently used is GetGrades, and rarely UpdateProfile is also called. UpdateProfile gets information from the profile database, and saves back the changes. Login uses the authentication server to validate credentials and the web server to encrypt and interpret the response, while Logout saves environment data to the profile database.

![Figure 21. Use case diagram](image)
GetGrades, illustrated in Figure 22, has a more complex structure: it includes steps that call two other use cases (scenarios), Login and Logout, and also includes a fork node, in order to perform the queries to database servers in parallel.

Figure 22. Activity diagram for GetGrades

The essence of the LQN model for the previously described system was already shown in Figure 6, as an example of an LQN model. The model that the tool actually builds from the UML diagrams is more complex: use cases are replaced by their scenarios, so that workloads (clients) issue requests directly to the first step in a scenario, each step in an activity diagram becomes a task, and active resources become processors. After applying the batched partitioning strategy, the model is divided into 12 layers. In Figure 6, for conciseness, only the use cases and the active resources were represented and the interactions between them. Information is shared among different nodes within separate layers by means of the system model, which implements the hybrid meta-model, shown in section 3, HYBRID META-MODEL.

The UML model was defined in Papyrus UML [35] using its extension for MARTE, in order to have access to performance annotations and specific stereotypes (such as PaStep and GaScenario).

The contents of the configuration JavaScript file used for running both the simulation and the hybrid solver is listed in Figure 23.

```javascript
var simDuration=100000;
var iterationCount=10;
var confRelWidth=0.95;
var getGradesProb=0.85;
var updateProfileProb=0.15;
var loginProb=0.0;
var logoutProb=0.0;
var studentRequestPattern=["closed", 8, ["exponential",20]];
```

Figure 23. Configuration file
5.2. Performance Analysis Results

Table 1 summarizes the analysis of the previously presented system with the following configuration: simulation duration = 100000, confidence interval relative width = 0.95, number of iterations = 10. The level k from which simulation starts has been modified, in order to study the effects it has on results. The hybrid model think time propagation formula (2.2 and 2.3 respectively) is also mentioned, in order to illustrate the experimental observation that led to reevaluating the original formula.

Table 1. Response time values obtained using each approach (in simulated time units)

<table>
<thead>
<tr>
<th>Performance model component</th>
<th>Simulation</th>
<th>Level k for simulation within hybrid approach</th>
<th>Hybrid (2.2)</th>
<th>Hybrid (2.3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logout use case</td>
<td>163.86</td>
<td>3</td>
<td>136.06</td>
<td>136.06</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5</td>
<td>158.50</td>
<td>153.48</td>
</tr>
<tr>
<td></td>
<td></td>
<td>6</td>
<td>354.95</td>
<td>243.61</td>
</tr>
</tbody>
</table>

5.3. Hybrid Method Evaluation

The values differ from the simulation values, but the reason is the fact that the MVA algorithm implied in the analytical solver is rather simple; no advanced approximations are used, so that errors are propagated along high level submodels. It seems that simulation should start at levels above the middle layer, but it may depend on the particular system model being analyzed.

Regarding the speed, the pure simulation runs for 100000 time units, while the hybrid method performs 10 iterations, each including a 100000 time unit simulation for the simulated submodel. The actual durations are presented in Table 2.

Table 2. Analysis duration in each approach (in minutes and seconds)

<table>
<thead>
<tr>
<th>Simulation</th>
<th>Level k for simulation within hybrid approach</th>
<th>Hybrid</th>
</tr>
</thead>
<tbody>
<tr>
<td>2’2”</td>
<td>3</td>
<td>2’44”</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>2’37”</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>13’49”</td>
</tr>
</tbody>
</table>

In case of the considered case study, the simulation results converge after 30000 time units, so the time is less than the equivalent for 100000 simulated time units.
For the hybrid solution, for k=3 and 5, the solutions for the simulation step converge during a few iterations, while for k = 6, the solution does not converge, and this is also proven by the result which is far from the simulation result.

The simulation submodel contains fewer elements than the simulation model for the entire system; and secondly the MVA analytical technique yields results faster than simulation of the high levels of the system. Simulation at high levels is very expensive – the number of possibilities that are to be experimented is greater than at lower levels. This is a problem for analytical models too, materialized as state space explosion, an example being exact MVA applied for multiple classes of requests. With the proposed hybrid approach, the analytical solver starts from the results provided by the simulator, so the complexity remains at a low level for both steps involved in each iteration.

When k has small values, the hybrid method could take more time than the simulation because of the complexity for simulation at high levels.

When k has higher values, the hybrid method should take less because (A)MVA runs faster, even with the simulated submodel being evaluated several times.

Several other, more complex case studies will be built and run for many more iterations and the tool will undergo an iterative development, adjusting its behavior, where it is the case, in order to obtain more accurate results.

When running multiple case studies, the appropriate values for level k will be established heuristically. In the previously presented case, obviously the best value for k is 5.
6. CONCLUSIONS AND FUTURE WORK

The proposed method does not have the desired accuracy, but it is a starting point for future developments. The speed of the method is as expected, the simulation submodel together with the analytical solver perform faster than the simulator by itself.

The analysis tool will be subject to improvements regarding the analytical method and also to generalization, in order to accept mixed requests, both open and closed as workloads for the analyzed system models.

A formal approach concerning transformation of UML diagrams to LQNs was defined in [14]; it could be applied to transform high level submodels for more complex diagrams, in order to be able to apply the MVA method. Since this paper is only concerned with modeling and solving the model, the model transformation techniques were reduced to minimal conversions, with the sole purpose of obtaining a layered model structure on which the solver can operate.

A straightforward application of the technique described in this paper is related to the declaration of components inside LQNs [27]. Component templates (classes) are described using specific language constructs; when reusing them inside particular LQN models, their parameters are customized (the classes are instantiated). Tasks are slots of LQN models where components can be instantiated, by providing appropriate binding sections. Performance parameters for component instances can be obtained by simulation, in case they do not further rely on lower level tasks in the model. The rest of the model could then be solved analytically.

The proposed method can further be improved by establishing an appropriate level k, from which simulation should be performed. Given the total number of layers and their complexity, k can be heuristically computed depending upon each system model.

The tool can also be applied in case of real-time systems analysis, since the MVA solver module is separated from the hybrid meta-model, so the analytical solver and the propagation rules can be easily changed in order to obtain worst-case values instead of mean values for performance parameters.

All the previously mentioned improvements require major experimentation with a wide range of case studies that will provide relevant results in order to evaluate, improve and validate the proposed approach.

The first step was to define the methodology: analytical and simulation approaches were not combined before within the same algorithm, this research direction will further be pursued by examining thoroughly each component approach.
REFERENCES


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