Softw Syst Model (2014) 13:1529–1552
DOI 10.1007/s10270-013-0385-x

REGULAR PAPER

Transformation challenges: from software models to performance models

Murray Woodside · **Dorina C. Petriu** · **José Merseguer** ·
Dorin B. Petriu · **Mohammad Alhaj**

Received: 12 September 2011 / Revised: 29 January 2013 / Accepted: 25 September 2013 / Published online: 31 October 2013

Abstract A software model can be analysed for non-functional requirements by extending it with suitable annotations and transforming it into analysis models for the corresponding non-functional properties. For quantitative performance evaluation, suitable annotations are standardized in the "UML Profile for Modeling and Analysis of Real-Time Embedded systems" (MARTE) and its predecessor, the "UML Profile for Schedulability, Performance and Time". A range of different performance model types (such as queueing networks, Petri nets, stochastic process algebra) may be used for analysis. In this work, an intermediate "Core Scenario Model" (CSM) is used in the transformation from the source software model to the target performance model. CSM focuses on how the system behaviour uses the system resources. The semantic gap between the software model and the performance model must be bridged by (1) information supplied in the performance annotations, (2) in interpretation of the global behaviour expressed in the CSM and (3) in the process of constructing the performance model. Flexibility is required for specifying sets of alternative cases, for choosing where this bridging information is supplied, and for overriding values. It is also essential to be able to trace the source of values used in a particular performance estimate. The performance model in turn can be used to verify responsiveness and scalability of a software system, to discover architectural limitations at an early stage of development, and to develop efficient performance tests. This paper describes how the semantic gap between software models in UML+MARTE and performance models (based on queueing or Petri nets) can be bridged using transformations based on CSMs, and how the transformation challenges are addressed.

Keywords Software performance · Performance analysis · Model transformation · UML · MARTE profile · Layered queueing networks

1 Introduction

Model-driven engineering (MDE) uses *abstraction* to separate the model of the software from underlying platform models, and *automation* to generate code from models. Models also facilitate the analysis of non-functional properties (NFPs), such as performance, scalability, reliability, security, safety. MDE can be applied to a variety of models related to software, including workflow models. To evaluate a software model for NFPs, analysis models are ideally generated automatically by model transformations and become part of the model suite which is maintained with the product. This paper describes a framework called PUMA (Performance from Unified Model Analysis) that automatically derives a variety of performance models from UML software specifications.

For software performance evaluation, many modelling formalisms have been developed over the years, such as

---

Communicated by Prof. Hans Vangheluwe.

M. Woodside · D. C. Petriu (✉) · D. B. Petriu · M. Alhaj
Department of Systems and Computer Engineering, Carleton University, 1125 Colonel By Drive, Ottawa, ON K1S 5B6, Canada
e-mail: Dorina.Petriu@sce.carleton.ca

M. Woodside
e-mail: cmw@sce.carleton.ca

D. B. Petriu
e-mail: dorin@sce.carleton.ca

M. Alhaj
e-mail: malhaj@sce.carleton.ca

J. Merseguer
Departamento de Informática e Ingeniería de Sistemas, Universidad de Zaragoza, Zaragoza, Spain
e-mail: jmerse@unizar.es

queueing networks (QNs), Layered Queueing Networks (LQN) (a type of extended QN), stochastic Petri nets, stochastic process algebras and stochastic automata networks, as surveyed in [2]. Simulation is also widely used. This paper addresses the creation of software models in UML [25], for systems with stochastic workloads, to obtain performance measures such as capacity, throughput and response times. For brevity, we term the software models as *Smodels*, and the performance models as *Pmodels*.

The benefits of using Pmodels during the software development process include discovery of performance limitations in system architecture, scalability analysis, design of efficient performance tests, capacity planning for deployed systems, and model-based configuration optimization [8,16,40]. There is a well-established methodology called *software performance engineering* [34–36] using Pmodels derived from expert knowledge or from test data, throughout the software lifecycle. Unfortunately, its practical application is sometimes hindered by the effort of building the performance models by hand. PUMA is intended to automate this step.

To facilitate the generation of Pmodels, UML Smodels have been extended with standard performance annotations defined in the "UML Profile for Modeling and Analysis of Real-Time and Embedded Systems" (MARTE) [24] and its predecessor the "UML Profile for Schedulability, Performance and Time" (SPT) [26]. The PUMA framework (first developed by the authors for UML+SPT models [41]) integrates Pmodels into MDE as illustrated in Fig. 1. (The numbered circles represent different transformation steps required to bridge the gap between Smodel and Pmodel, as described in Sect. 3).

This paper describes a new version of PUMA for UML+MARTE models, which addresses the following transformation challenges:

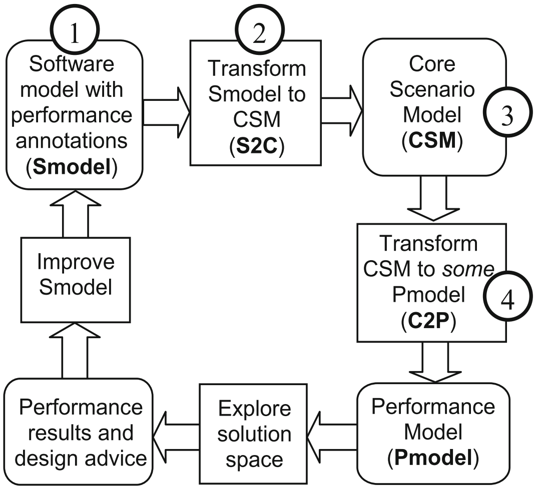


Fig. 1 The PUMA architecture, with four steps discussed in the paper

- bridging the semantic gap between Smodels and Pmodels, which is due to their different domains; performance models are centred on resources and abstract away from details of function and data [28];
- overcoming the complexity of dealing with several distinct kinds of Smodel and many kinds of Pmodel (an *N-by-M* problem);
- inferring behaviour patterns over extended patches of system scenarios, including patterns of interaction between system components, and patterns of resource holding, which require determination of *resource contexts* of behaviour [43];
- incorporating system elements which are indicated but not fully described in the Smodel.

These transformations are largely implemented in PUMA, covering Smodels expressed by Interaction, Activity and Deployment Diagrams (IDs, ADs, and DDs) and Pmodels in the form of QNs, LQNs, generalized stochastic Petri nets (GSPNs) and simulations. In this paper, we will focus on the transformation to two types of Pmodels, LQNS (Sect. 7) and Petri nets (Sect. 8).

PUMA addresses the N-by-M challenge by using an intermediate CSM model as illustrated in Fig. 2. CSM captures the necessary information about the use of resources by behaviour, which is the essence of all performance models. Now, to add a new type of Smodel or Pmodel requires only one additional transformation into or from CSM.

2 Related work

Many kinds of Pmodels can be used for performance analysis of software systems as described in [2] and [8]. The Pmodels are often constructed "by hand", based on analyst insights and interactions with designers. To fit into MDE, the present purpose is to automate the derivation of the Pmodel from the Smodel used for software development. Several approaches have been proposed for this.

In some research, a special restricted style of "performance Smodel" has been proposed, to specify only the software aspects that are relevant to performance models. An example is the pioneering "execution graph" of Smith [35,36], a kind of scenario model (as described in Sect. 4) with performance parameters. The execution graph, which may have a UML front-end [7,21], is transformed directly to a Pmodel. Other examples of "performance Smodels" include a constrained style of UML [18], including annotated structural definitions in code [22] and the Palladio Component Model (PCM) [3]. The latter is a modelling language intended for model-driven development of component-based software systems and for the early evaluation of NFPs such as performance and reliability, which captures the software archi-

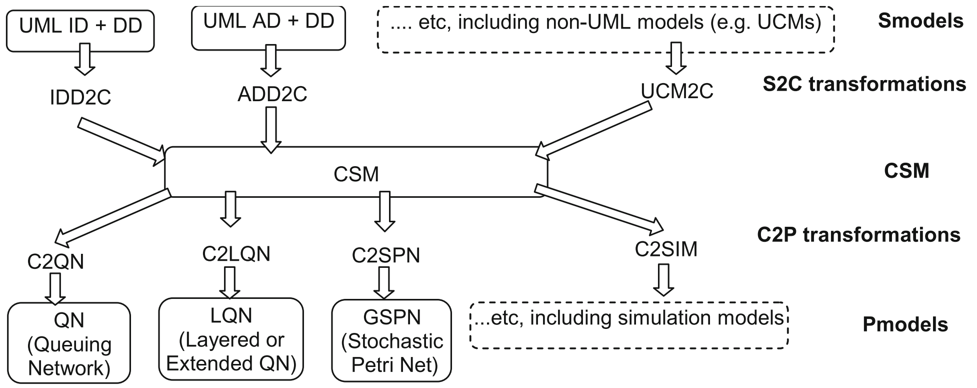


Fig. 2 Transformation architecture using the CSM intermediate model

Table 1 Automated transformation of UML Smodels to Pmodels

Target Pmodel	Source Smodel			
	UC + DD	SD + DD	AD + DD	SM + DD
Queueing network	[7,14]	[7,14,41]	[14,21,41]	
Layered QN		[14,18,27,41]	[14,27,29,41]	
Stochastic Petri net	[9]	[9,14,41]	[9,14,20,23,41]	[9,17]
Stochastic process algebra		[39]	[6]	
Markov model				[19]
Simulation		[1,27]	[21,27]	

UC use case, *SD* sequence diagram, *AD* activity diagram, *SM* state machine, *DD* deployment diagram

tecture with respect to static structure, behaviour, deployment/allocation, resource environment/execution environment, and usage profile. Although its metamodel is completely different from UML, the Palladio Component Model has a UML-like graphical notation representing component diagrams, deployment and individual service behaviour models (similar to activity diagrams).

The capabilities provided by some of the extensive research on automated transformation of UML Smodels to different PModels are summarized in Table 1, with references to papers.

Many of these approaches transform from *one* kind of UML behaviour diagram (plus deployment), to *one* kind of Pmodel. However, there are many benefits in being able to start from any kind of UML behaviour diagram and to choose the most suitable Pmodel for a given project. The PUMA strategy in [41] *unifies* performance evaluation in this sense, transforming multiple types of UML behaviour model into multiple types of Pmodel, via an intermediate (or pivot) language called Core Scenario Model (CSM) [30]. PUMA is capable of transformations in every cell of Table 1 and also supports non-UML Smodels (e.g. Use Case Maps [46]).

CSM represents sequences of operations, based on the concepts in the SPT/MARTE profiles, and exploits several standards: MARTE; UML and its model-interchange standard; performance model standards [15,33]; and the CSM metamodel [30]. Other intermediate models from literature include IM in [27] and PCM in [9], which are similar to CSM. KLAPER is another intermediate language that supports performance and reliability analysis of component-based systems [14]. KLAPER is more oriented towards representing calls and services rather than scenarios and has a more limited view of resources (i.e. no basic distinction between hardware/software, active/passive). It has also been applied as intermediate model for transformation from different types of Smodels to different types of Pmodels.

For PUMA, the preliminary paper [41] outlined transformations from sequence and activity diagrams extended with the SPT profile to CSM, and from CSM to queueing, layered queueing and stochastic Petri net models. The limitations in these original transformations mean that some valid designer options for expressing the Smodel cause failure to produce a Pmodel. This work describes a significantly enhanced PUMA framework based on MARTE, which addresses the transformation challenges listed in the Sect. 1 and detailed in Sect. 3.

3 Bridging the semantic gap between Smodel and Pmodel

The Smodel contains a wealth of design specification that is summarized or ignored in the Pmodel, and the Pmodel extends outside the normal content of an Smodel, in its focus

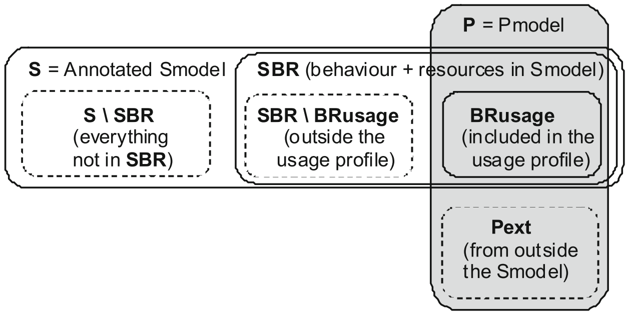


Fig. 3 Conceptual groupings of the semantic content of the Smodel and Pmodel

on the use of resources. There is overlap in the structural, behavioural and resource specifications that are common to both, but their central features are quite separate, creating a semantic gap between them. The Smodel is function-centric, while the Pmodel is resource-centric. This gap is crossed by using the common elements, which describe the resources and the units of behaviour that use these resources (called *steps* in this work). Starting from a typical Smodel, tone must first complete the description of behaviour and the execution platform, and then add performance annotations which specify how the behaviour uses the resources in executing the functions, and perhaps some additional resources. The relationships between the elements of a UML Smodel and its corresponding Pmodel are illustrated as subsets of model elements in Fig. 3.

SBR is the subset of the Smodel model elements that specifies behaviour and its use of resources, while **BRusage** is the subset of **SBR** that is related to the *usage profile* for the Pmodel (the set of system-level responses that is to be modelled). The Pmodel is extracted from **BRusage** plus additional specifications of system components outside the Smodel altogether, shown as **Pext**.

The Pmodel is more abstract than the Smodel [28]:

- functional operations are abstracted using the MARTE annotations:
  - control decisions are abstracted to random choices governed by probabilities which must be supplied;
  - functional execution is represented abstractly by probability distributions or average demand values for CPU time, message lengths and sizes of storage operations.

  The parts that are kept are included in the set **SBR**.

- the effect of data on behaviour is abstracted, since the run-time data are not represented in the Pmodel. The effect of variations in the data is represented within the distribution of demand values noted above;
- some operations may be omitted from the Pmodel. Performance analysis focuses on the use cases which are regarded as important for performance, and for which there are performance requirements, called the *usage profile* of the system. This restricts to the Pmodel to the subset **BRusage** in Fig. 3;
- information may have to be added to the model, shown as set **Pext** in Fig. 3:
  - similar to a transformation to a platform-dependent model, the performance model must include abstractions of the execution platform, parts of which may be ignored in the Smodel (if it is platform-independent). Examples include middleware, databases and storage subsystems. These have been termed *performance completions* [42] and may be represented by additional overhead execution demand, or by pre-built Pmodel elements defined in **Pext**;
  - the system may include components that are already developed or are separately specified. These may also be represented by Pmodel elements defined in **Pext**.

Transformation steps and road map

The paper describes the transformation from SModel to Pmodel in four steps, indicated by numbered circles in Fig. 1:

Preliminary Step: identify the operations to be analysed (the usage profile) and ensure that the Smodel includes their behaviour description;

1. in the Smodel, add the performance annotations using MARTE stereotypes and attributes, to complete **SBR** (MARTE is described in the remainder of this section);
2. extract **BRUsage** from the Smodel into the CSM, which eliminates the unused parts of the Smodel (CSM in Sect. 4, the S2C transformations in Sect. 5);
3. analyse the CSM for extended resource properties (interaction patterns and resource use patterns across the scenario; they are needed by the LQN Pmodel, not by the QN or GSPN Pmodels) (Sect. 6);
4. transform the CSM to the chosen Pmodel (Sect. 7).

The preliminary step and Step 1 are manual, while Steps 2, 3 and 4 are automated in PUMA.

3.1 MARTE performance annotations

UML extensions to specify information about time and resources, to bridge the semantic gap, are defined in the MARTE standard profile [24]. Important packages of MARTE for our purposes are the NFPs, general resource model (GRM), generic quantitative analysis model (GQAM), and performance analysis model (PAM). Quantities are specified by *NFP*s (non-functional properties), which have a compact form *(value, units)*, where *value* may be a number, a variable, or an expression in the Value Specification Language ([24], Annex B), and *units* are described in Annex D.2. Some NFP types support ranges of values, or probability distributions. There is also a long form which specifies additional properties of the NFP value ([24], sec 8.3.3).

Highlights of MARTE will be introduced via the UML interaction diagram (ID) and deployment diagram (DD) in Figs. 4 and 5, which are based loosely on the TPC-W benchmark [38] representing an electronic bookstore. The ID in Fig. 4 defines behaviour to get the home page of the bookstore. This single response will make up the usage profile for this small example. The stereotype «*GaAnalysisContext*» identifies the ID as a subject for analysis and its *contextParams* attribute declares four parameters for the analysis:

- *Nusers*, the number of concurrent users in a closed workload,
- *thinkTime*, between the end of a response and the next request by the same user,
- *Images*, the average number of images in a web page,
- *R*, the required 95th percentile of the response time.

In the stereotype attributes, the “$” sign signifies the declaration of a variable; NFP_duration is the NFP type for time values, NFP_integer is for integers. These four parameters can be varied during the Pmodel evaluation to provide sensitivity analysis.

MARTE stereotypes are based mainly on the concepts of *scenarios*, *workloads* and *resources*. A scenario is a behaviour specified by an AD, ID or state machine diagram (SMD) (which are not considered here). A scenario is triggered by an event pattern defining its “workload” and is made up of Steps which are either elementary actions that take time and use resources, or containers for nested subscenarios. The software process instances (each of which gives one lifeline in the ID) are logical resources, while the hosts and the network are physical resources shown in the DD of Fig. 5. Other resources may be active or passive, logical or physical, software or hardware. In the example, we shall consider the MARTE annotations for the scenario and workload first, then consider the resources.

In Fig. 4, the scenario is implicitly the entire ID. Its workload is defined by the «*GaWorkloadEvent*» stereotype applied to the beginning of the scenario, with attributes *pattern* (describing the events that trigger responses) and *respT* (the response time to the event). The pattern defined here is *closed*, with a fixed population of *Nusers* users, who wait for *thinkTime* seconds between requests (notice the use of variables *Nusers* and *thinkTime*). An alternative is an *open* pattern, defining a flow of requests at a given rate. *respT* is defined with two values with different sources, one for the required value and one defining the variable *R* as a placeholder for the calculated value obtained from the Pmodel. To define the different sources, the long-form specification of *respT* is used. The *statQ* field declares the value to be a percentile (the 95th in this case).

The scenario is defined implicitly by the sequence of «*PaStep*», in which the stereotype may be attached to either an *ExecutionSpecification* (drawn as a narrow rectangle along the lifeline) or to the message which triggers it. A «*PaStep*» has an attribute *hostDemand* which defines its host execution time. «*PaStep*» is also applied to the *CombinedFragments* in Fig. 4, as a container for an implicit nested scenario representing the fragment content. «*PaStep*» has an attribute *prob* for the probability of optional or alternative fragments (prob is 0.2 for the `opt` fragment, and 0.4 and 0.6 for the two `alt` fragments in Fig. 4), or *rep* for repetitions of a loop (*rep* is the number of images to be retrieved, given by the variable *images*, for the `loop` fragment). In a `par` CombinedFragment, the attribute *noSync* on a fragment indicates that the joining of the parallel behaviour does not wait for this branch.

Some messages in the scenario may have an additional stereotype «*PaCommStep*» conveying an attribute *msgSize*, which may be used in the Pmodel to determine the message delay. The first message has a size of 2.7 KB; the final one has a size given by an expression depending on the number of images in the homepage (the variable *Images*).

The logical resources in this system are the «*PaRunTInstances*» (deployed processes) associated with each lifeline in the ID, with thread pools of size *poolSize* and an attribute *instance* that identifies the process instance (a «*SchedulableResource*» whose deployment is shown in the DD). The physical resources are «*GaExecHosts*» (compute nodes) and the «*GaCommHost*» (network). Each «*GaExecHost*» has attributes *resMult* (for its number of cores or processors), and transmission and reception overheads per message as shown. The «*GaCommHost*» has a transmission capacity and a latency attribute named *blockT*.

Going beyond this example, a «*PaStep*» may identify the invocation of additional behaviour by explicitly nesting a scenario defined by another behaviour diagram within it, or by defining demands for operations defined elsewhere using the «*PaStep*» attributes *behavDemand* (for nested scenarios), *servDemand* (for operations defined by a software component with its own scenarios), or *extOpDemand* (for operations defined in a library). Also a scenario may explicitly define the use of logical resources, with a stereotype «*PaAcqStep*» for a step which acquires a resource, and «*PaRelStep*» for a step which releases one.

UML ADs use the same annotations, with «*PaStep*» applied to Actions; an example is shown in Fig. 17. State machine diagrams can also be annotated (see e.g. [20]).

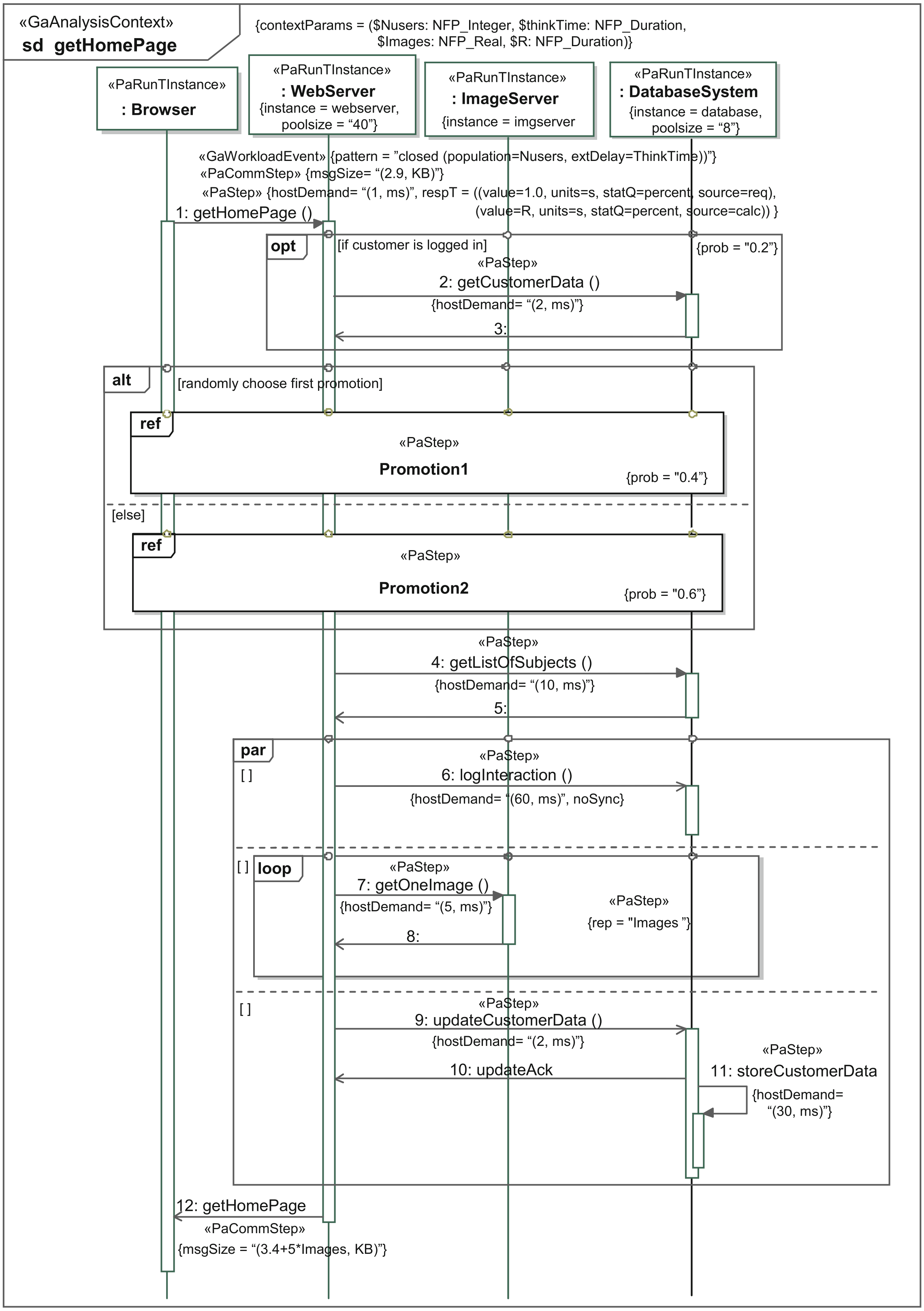


Fig. 4 A UML2 interaction diagram for the GetHomePage scenario of the TPC-W benchmark

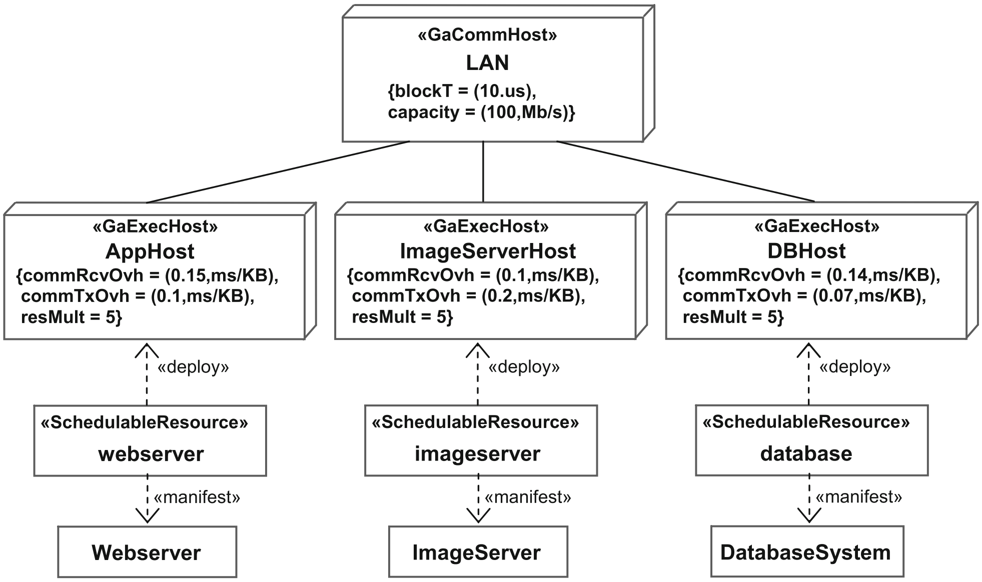


Fig. 5 Software components and their deployment

The many additional annotations in MARTE include identification of logical resources such as semaphores, locks or buffer pools. They can be modelled by declaring a logical resource, where it is acquired and released.

As a minimum input for performance analysis, the annotated Smodel must include:

- the usage of the system, defined by «*GaAnalysis Contexts*» which define behaviour, with their «*GaWorkloadEvents*» and «*PaRunTInstances*»;
- annotations for *hostDemands* of «*PaSteps*»;
- deployment connecting «*PaRunTInstances*» to «*SchedulableResources*» and these to «*GaExecHosts*»;
- modelling of those logical resources that are expected to affect performance.

4 Intermediate modelling language: The Core Scenario Model

The CSM extracts the behaviour and resource information from the Smodel (called the subset **BRusage** in Fig. 3) using a metamodel shown (without attributes) in Fig. 6; details are described in [30] and its XML schema is available at [31]. The metamodel is based closely on MARTE, with corresponding elements as shown in Table 2.

The implicit sequence relationships in the Smodel map to explicit CSM *PathConnectors* (*Start, Sequence, Branch, Merge, Fork, Join, End*), called *PCs* here for brevity. Acquisition and release of process resources are implicit in MARTE and map to *ResourceAcquire* and *ResourceRelease* steps in CSM.

Figure 7 illustrates the mapping of sequence relationships and resource operations, using the shorthand *ra* and *rr* for *Resource Acquire* and *Resource Release* steps. Other scenario models lack the generality of CSM regarding resource modelling. For example, execution graphs in [36] indicate resource acquisition/release for processes and locks, but not for units of multiple resources like a pool of buffers. PCM [9] requires that fork/join sections join all branches, and fork/join and branch/merge sections be fully nested. KLAPER [14] has a more limited view of resources, considering that hardware and software resources offer services, which can be detailed in terms of behaviour. This represents process resources but not pure logical resources.

Nested subscenarios in MARTE and CSM

MARTE can associate a subscenario with a «*PaStep*» in three ways:

1. as a subscenario stereotyped «*GaScenario*» which refines the «*PaStep*»; the step is an abstraction for the subscenario;
2. as a behaviour included in the «*PaStep*», defined by an attribute *behavDemand,* with a repetition count *behavCount*;
3. as the behaviour of a service invoked by the «*PaStep*» with a demand *servDemand* and a repetition count *servCount*.

Normalized and flattened CSM

A *normalized* CSM has certain properties which make it easier to process further:

- there should be a *PC* between every pair of *Steps* (including *ResourceAcquire /Release/Pass Steps*);

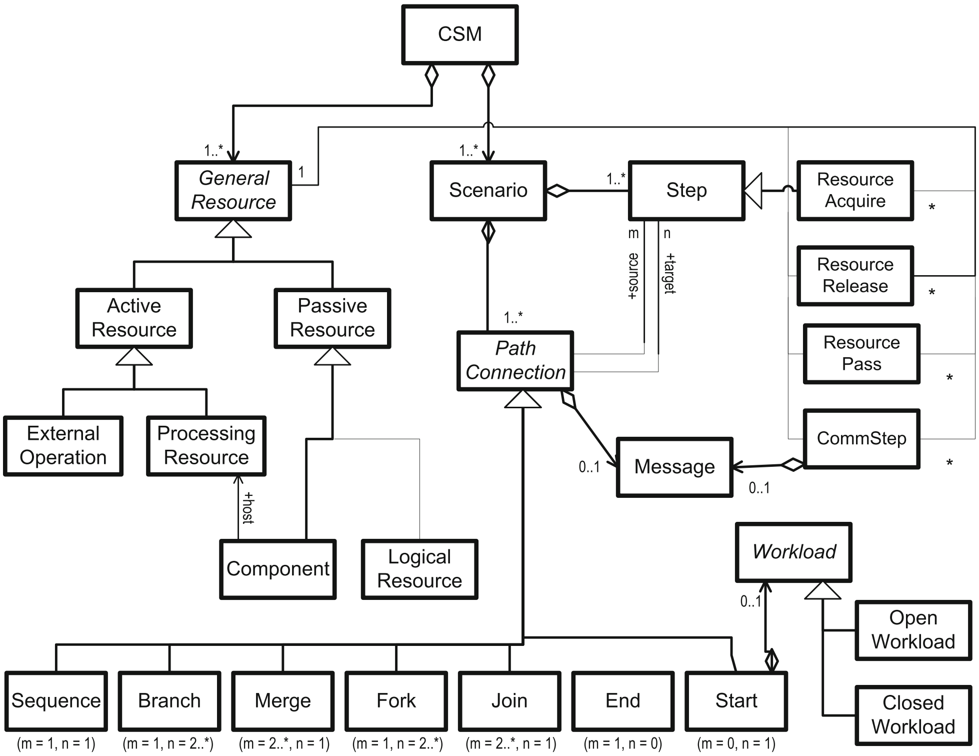


Fig. 6 Simplified Metamodel of the Core Scenario Model

Table 2 Correspondences between MARTE stereotypes and CSM elements

MARTE	CSM
«GaWorkloadEvent»	Closed/OpenWorkload
«GaScenario»	Scenario
«PaStep»	Step
«PaCommStep»[a]	CommStep
«GaResAcq»[a]	ResourceAcquire
«GaResRel»[a]	ResourceRelease
«PaResPass»[a]	ResourcePass
«GaExecHost»[b]	ProcessingResource
«PaCommHost»[b]	ProcessingResource
«PaRunTInstance»[b]	Component
«PaLogicalResource»[b]	LogicalResource

[a] Subtype of «PaStep» in MARTE
[b] Subtype of «Resource » in MARTE

- every primitive *Step* (which excludes *ResourceAcquire/Release/Pass Steps* and *Steps* with nested subscenarios) should have some nonzero execution demand and an associated Component to execute it;
- every *Component* (essentially, a process) should have an associated *host* processor.

A CSM which violates these properties can be *normalized* to satisfy them.

One CSM may include several separate independent *top-level* scenarios representing different externally available system operations, each with its own workload to describe how it is driven. If a top-level scenario is also used as a nested subscenario, then its workload is ignored when it is nested. A top-level scenario is *flattened* by recursively replacing its steps containing nested subscenarios with instances of the subscenarios.

5 Transformation from Smodel to Core Scenario Model (S2C)

One Smodel scenario is transformed at a time, by identifying a scenario and following it, using the causal implications from the UML scenario. In an AD, causality is implied by ActivityEdges between actions, in a SM by state transitions, but in an ID, causality is more complex and is addressed in Sect. 5.1. The implemented transformations cover IDs, ADs and their associated DDs. Instead of a DD, a MARTE user can define deployments using special allocation stereotypes (see chapter 11 in [24]).

5.1 Causality and sequence in a UML ID

In UML2, in activity diagrams and state machine diagrams, the sequence of steps is explicitly defined by transitions which establish causality. Interaction diagrams, however,

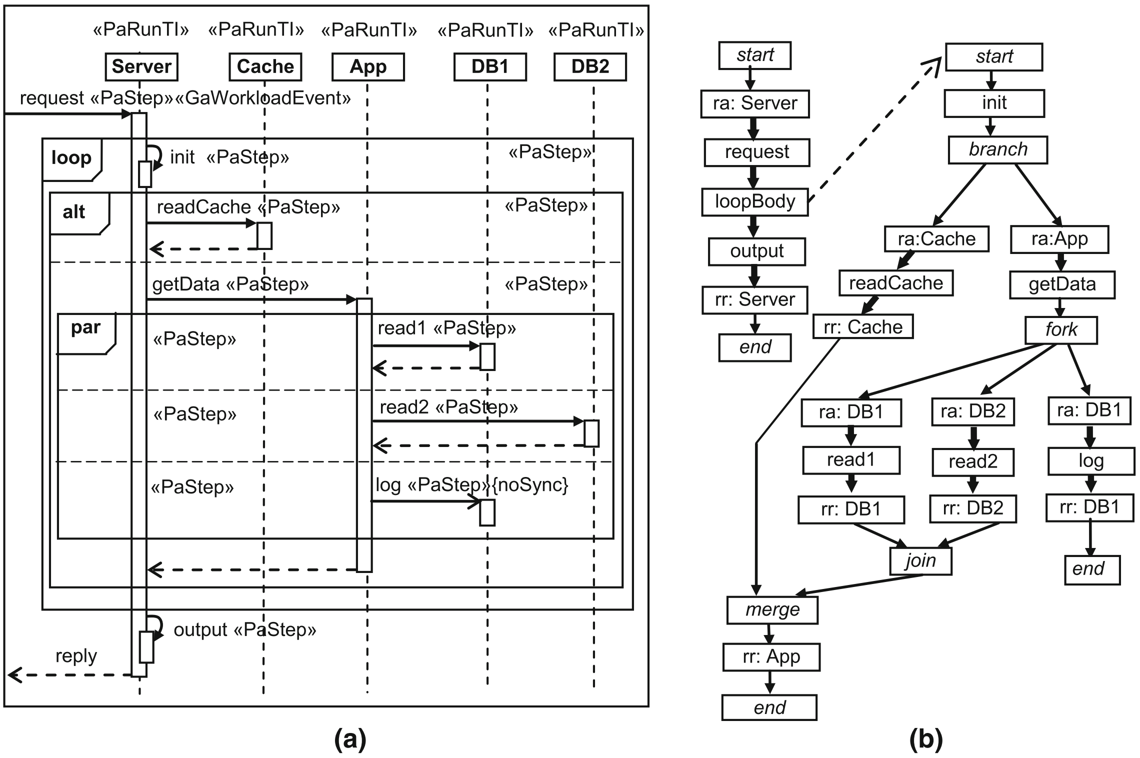


Fig. 7 An interaction diagram and the corresponding CSM (with subscenario loopBody). (*Notes*: «PaRunTI» means «PaRunTInstance». The CSM uses Roman font for Steps, *bold arrows* for *Sequence* PathConnectors, and *italics* for other PathConnectors. "ra/rr: Resource" specifies a ResourceAcquire/Release Step applied here to a named PaRunTI process resource). A *forked branch* that will not re-join is indicated with {noSync}. **a** ID, **b** CSM

only define event traces which must be satisfied in some sense by the behaviour; there may be events which are not shown in the ID. Several semantic interpretations of IDs are discussed in [32], and this work uses the "UML2 interpretation" defined there.

Transformation of an ID to a performance model treats the precedence relationships as causal, based on the time order which is given by their vertical position in the diagram. An ID is a list of *interaction fragments* (IFs) such as*MessageEnd*, *CombinedFragment* (CFs), *ExecutionSpecification*, and *OccurenceSpecification*. Fragment *IFa* is inferred as a causal predecessor of *IFb* in the following conditions:

- if *IFa* immediately precedes *IFb* on the same lifeline;
- or if *IFa* is the event of sending a message and *IFb* is the event of receiving the same message.

An IF with no predecessor is a *Start* fragment; IFs with no successor are *End* fragments.

Dubious causality

For a pair of IFs (*IFa, IFb*), if *IFb* is a CF with multiple operands, it may not be possible to infer causality from its vertical position, and we say the *causality is dubious*. This is analysed by the causality inference algorithm in Table 3, by considering the first IFs within each operand of the CF.

Table 3 Algorithm for establishing causality between interaction fragments in an ID

1	If CF has just one operand, or if all first IFs are on the same lifeline, then the causal predecessor *IFa* is the last IF before *IFb* (the CF) on that lifeline
2	Else if there is only one "active" lifeline with an IF within at least one operand, *IFa* is the last IF before the CF, on that lifeline. A lifeline is termed "active" after receiving a message, and it becomes "inactive" after a blocking message send, or the end of an ExecutionInstance
3	Else if CF is *par* or *seq*, *IFa* is taken arbitrarily to be the last IF before the CF, on those lifelines with an IF within at least one operand
4	Else the causality is dubious, and *IFa* is taken arbitrarily to be the last IF on any lifeline with operand IFs, before the CF

Figure 8 illustrates *dubious causality*. The first and third lifelines are both active after the asynchronous message (equivalent to a fork in the flow). Before the `alt` CF in the ID, the previous IF is *IFx*, and this will be taken as the predecessor. However, it is not clear how*IFx* causes *IFy*. *IFy* must be caused by means that are hidden in the diagram (such as by inspection of shared data set by the Printer). Dubious causal-

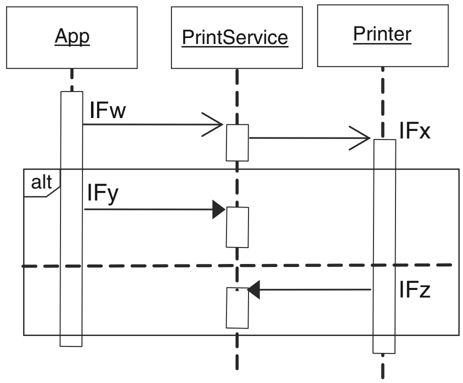


Fig. 8 Dubious causality for an `alt` CombinedFragment (CF)

ity does not prevent building Pmodels, but raises a question about behaviour completeness in the generated Pmodels.

PathGraph for Navigation in IDs

For the IDD2C transformation, the causal sequences in an ID are represented by a notional directed *PathGraph*, with a node (e.g. node a) for each interaction fragment *IFa* and a directed arc (a, b) if *IFa* is connected by a causal predecessor sequence to *IFb*. If *IFa* has multiple stereotypes which are subtypes of «*PaStep*», then it is treated as if it were a sequence of separate interaction fragments in this order: «*PaCommStep*», «*PaResourceAcquire*», «*PaStep*», «*PaResourceRelease*». (It is assumed that missing stereotypes have been inserted as described in Sect. 5.2). Interaction fragments within a combined fragment (CF) are treated separately, and each operand gives a separate PathGraph. A node with no predecessor is the *Start* node of a PathGraph, and a node with no successor is an *End* node. If for node a, *IFa* is a «*PaStep*» with a subscenario, node a is linked to the PathGraph for the subscenario.

Let inOrder(a) and outOrder(a)) be the number of arcs into and out of node a, respectively. An arc (a, b) may not have inOrder $(a) > 1$ and outOrder$(b) > 1$ at the same time. If this occurs, then the condition is enforced by replacing arc (a, b) by a dummy node a' and single arcs (a, a') and (a', b). Each PathGraph generates a CSM scenario.

5.2 Scenario preprocessing exceptions and special cases

Missing information

In practice users may forget to insert some annotations or attributes. Before performing the actual transformation, a robust transformation process should detect and report missing information, but continue on as far as possible, and provide the richest possible diagnostics. Some missing MARTE stereotypes and attributes may simply be provided:

- many attributes of MARTE stereotypes have default values which are used if no value is assigned;
- the «*PaStep*» stereotype can be assigned to those entities that may normally support it, if it is not defined (e.g. *Message*, *ExecutionSpecification* and combined fragment operands in IDs, Actions in ADs};
- the «*ExecHost*» stereotype can be assigned to any *NodeInstance* or *Node*, and «*CommHost*» to any link in a DD.

Additionally, at user's discretion, some more aggressive fill-ins may also be desirable:

- to interpret all behaviour diagrams as *AnalysisContexts* with scenarios;
- to supply a «*GaWorkloadEvent*» stereotype to a scenario that lacks one, with attributes *{pattern = closed (population = 1, thinkTime = (0.0, s))}*. This defines an artificial workload that will at least provide a solvable model, which can be corrected later;
- to supply a «*PaRunTInstance*» stereotype to any ID lifeline or AD partition that lacks one. The name attribute can be assigned from the lifeline/region, and an artificial host DefaultHost can be introduced as its deployment. We have found this artifice to be useful; DefaultHost has infinite multiplicity, so it can host any number of «*PaRunTInstances*» without introducing artificial congestion in the resulting performance model.

Nested Behaviour in a «PaStep»

Besides a *hostDemand* indicating CPU execution demand, a *Step* has three other attributes which (if defined) indicate additional behaviour in the form of a nested scenario. One is a direct reference to a nested scenario by the association *behaviour*; the second is an invocation (n times) of a scenario named by the attribute *behavDemand* with n given by attribute *behavCount*; the third is the invocation (n times) of an operation named by the attribute *servDemand* (which will normally in turn have a behaviour defined by a scenario in the Smodel), with n given by the attribute *servCount*.

Multiple resource demands in one «PaStep»

If more than one of the attributes *hostDemand*, *behaviour*, *behavDemand*, and *servDemand* is defined, then a separate Step is created in the CSM for each of them (in arbitrary order). The *Steps* created for the nested scenarios for *behavDemand* and *servDemand* have *rep* set to the value:

(*rep* of the original «*Step*»)*(*behavCount* or *servCount* for the invoked behaviour).

5.3 Transformations to CSM (IDD2C, ADD2C)

The IDD2C and ADD2C transformations are implemented separately because the UML metamodel for interaction and activity diagrams are very different. However, the transformations follow the same high-level approach, which is described in this section. The transformation begins by creating a CSM *ProcessingResource*, *Component* or *Logical-Resource* for each «*GaExecHost*», «*SchedulableResource*» or «*PaLogical-Resource*» respectively, and associating each *Component* to a *ProcessingResource*. Then, the starting points of scenarios are identified as entities with a «*WorkLoadEvent*» stereotype which are also the start of a PathGraph in an ID, or a Start node of an AD. One UML model may contain both ADs and IDs.

One scenario is transformed at a time. The behaviour is traced forwards along the PathGraph (in an ID) or following the *ActivityEdges* (in an AD), with PCs inferred from the UML presentation. For an AD, the CSM *Start/End/Branch/Merge/Fork/Join* PCs correspond to the AD elements of the same type, while for an ID, they must be inferred. For an ID, the *Start* is inserted before the first Smodel entity, the *End* is inserted after the last, *Branch/Merge* are implied by an `opt` or `alt` CF and *Fork/Join* are implied by a `par` CF or by sending/receiving an asynchronous message. In a CF, the operand(s) generate CSM *Steps* with nested subscenarios for the operand behaviour. Nested subscenarios are inferred from a `ref` CF (in an ID) or a *StructuredActivityNode* (in an AD).

Smodel «*PaSteps*» and «*PaCommSteps*» are translated to CSM *Steps* and *CommSteps* with the corresponding attributes, except where multiple CSM Steps are created, as described above. Implicit resource acquisition and release of process resources (e.g. threads) is inferred wherever the behaviour crosses from a «*PaStep*» in one process to a «*PaStep*» in another (from one lifeline to another (for an ID), or from one *ActivityPartition* (swimlane) to another (for an AD). Figure 9 shows the pseudocode for the IDD2C transformation algorithm.

Figure 10 shows a screen shot of the generated CSM for the UML GetHomePage scenario given in Fig. 4, with comments showing the transformation of CFs, and indicating six subscenarios for CF operands, which are not shown in detail. Notice the R_Acquire and R_Release Steps to acquire and release the process resources for the SchedulableResources, inferred from a message from one lifeline to another.

6 CSM analysis for resource-holding and component interactions

For an LQN Pmodel, additional properties of the CSM are needed, which are described in this section. For other types of Pmodel, these analyses are not required.

6.1 Logical resource context of a step

To determine the holding time of a logical resource, the operations that are carried out during its holding times must be identified. This is done by first finding what resources are held in executing each *Step* (defined as *resource context* [43]). Resource context *inconsistencies*, which may be logical errors in defining the resource use, are discussed below.

The resource context **R**(S) of *Step* S in a CSM is an ordered set (a stack) of logical resources held during the execution of S, including blocked and held process threads and pure logical resources. For a context with n resources:

$$\mathbf{R}(\mathrm{S}) = \{(\mathrm{r}_1, m_1), (\mathrm{r}_2, m_2), \ldots(\mathrm{r}_n, m_n)\},$$

where resource r_1 is the first one acquired, r_n is the last one acquired, and for resource r_i, m_i units are held at *Step* S.

R is readily determined by traversing a CSM which has been normalized and flattened, adding/removing resources as they are acquired/released. At a Fork, the previous context is normally passed to all successors. However, a special case has been provided in MARTE for a parallel subpath which has a resource like a lock or buffer explicitly passed into it: the first Step has a «*PaResPassStep*» stereotype, which leads to a *ResourcePass* entity in the CSM. The identified resource then enters the context only in the one subpath. In the special case of a parallel subpath resulting from an asynchronous message, *only* explicitly passed resources enter the context.

Resource context inconsistencies

Context inconsistency at a Merge. Before a *Merge* PC, the contexts may be different, giving a resource context inconsistency. This can occur in a specification, but represents bad practice. For example, suppose a certain condition gives a branch path in which a buffer is obtained, and the same condition gives a later distinct branch path in which it is filled, used and released. If the buffer is accessed between these branches, there will be an error in the cases where it has not previously been obtained. Our solution is to abort the transformation and treat the inconsistency as a specification error. The Smodel can be corrected by extending the alternative paths to cover both branches, or by obtaining the buffer just before using it.

Resources in Parallel Subpaths. Parallel subpaths inherit the resource context from before the fork. This creates consistency questions if a subpath releases a resource; do the other subpaths retain it, or not? If they also release it, are two units released? To resolve this question, one subpath is chosen as the “owner” of the resource, and only this subpath can release it (the other subpaths, however, retain it in their contexts). Passing a resource to one subpath explicitly designates its owner.

Non-deterministic order of some resources at a Join: Resources obtained on different subpaths are not ordered among themselves. When one of these resources is released, its holding time is arbitrarily determined to be nested (see

```
1. For each PathGraph create a CSM Scenario. The C(a) of its Start node a is a Start PC. If IFa has a
   «GaWorkloadEvent» stereotype, then it is a top-level Scenario;
2. Traverse the PathGraph and process its nodes; after processing node a, consider each successor
   node b;
   2.1. if IFb is of type «PaStep» or one of its subtypes as shown in Table 2, C(b) is the
        corresponding CSM type in the Table 2, with attributes copied from IFb (including any nested
        sub-scenario);
   2.2. if IFb is of type «PaCommStep» then C(b) is the following sequence:
        2.2.1. Step (for transmission overhead by the sending Component), with hostDemand
               commTxOvhd * msgSize;
        2.2.2. Seq, ResourceAcquire for «CommHost», Seq;
        2.2.3. Step (for communication link transmission) on component «CommHost», with
               attributes delay = blockT, hostDemand = msgSize/capacity, (msgSize from
               «CommStep», blockT (= link latency) and capacity from «CommHost»);
        2.2.4. Seq, ResourceRelease for «CommHost», Seq;
        2.2.5. Step (for receive overhead by the receiving Component), with hostDemand
               commRxOvhd * msgSize, where «CommHost» is the link between the sending and
               receiving «ExecHosts» in the deployment. There are two special cases:
                   (i) if (msgSize = 0 and blockT = 0), or if the hosts are the same, C(b) is null;
                   (ii) if msgSize = 0 and blockT > 0, only item 2.2.3 is created to show the latency;
   2.3. if IFb is an alt CF, C(b) is a Step with an associated subscenario with a Branch PC attached
        to a Step for each operand, and a Merge. The CSM Step for each operand has the probability
        attribute of the operand «PaStep» and is associated to the subscenario for the operand;
   2.4. if IFb is a par CF, C(b) is a Step with an associated subscenario with a Fork PC attached to a
        Step for each operand, and a Join connected to all those Steps which do not have the attribute
        noSync set true. The CSM Step for each operand is associated to the subscenario for the
        operand;
3. Between C(a) and C(b):
   3.1. if C(a) ends with a PC, nothing more is necessary;
   3.2. if outOrder(a)>1, then C(a) should be followed by a PC of  type Fork (create if it does not exist
        already);
   3.3. elseif inOrder(b)>1, then C(a) should be followed by a PC of  type Join (create if it does not
        exist already);
   3.4. else C(a) is followed by a Sequence PC.
4. If outOrder(b) = 0, C(b) is followed by an End PC.
```

Fig. 9 IDD2C transformation algorithm

below) unless the determined part of the resource order contradicts it.

Non-determinism of resource context due to nested scenarios: A subscenario that is nested in a Step can modify the resource context due to probabilistic behaviour in the subscenario. Thus, it is preferred that a nested scenario should release any logical resources that it acquires, so it ends with the same resource context that it starts with. Without this "well-structured resource usage," the transformation is aborted.

6.2 Nesting of holding times and ordered use of resources

Whenever resources are released in the reverse order to which they were acquired, their holding times are nested (each resource holding time is contained within the holding times of resources acquired earlier and released later). *Full nesting* also has a global ordering of resources that is respected by all resource acquisitions, and guarantees freedom from resource deadlock. Thus, full nesting may be regarded as a "well-structured" resource discipline, although it is often

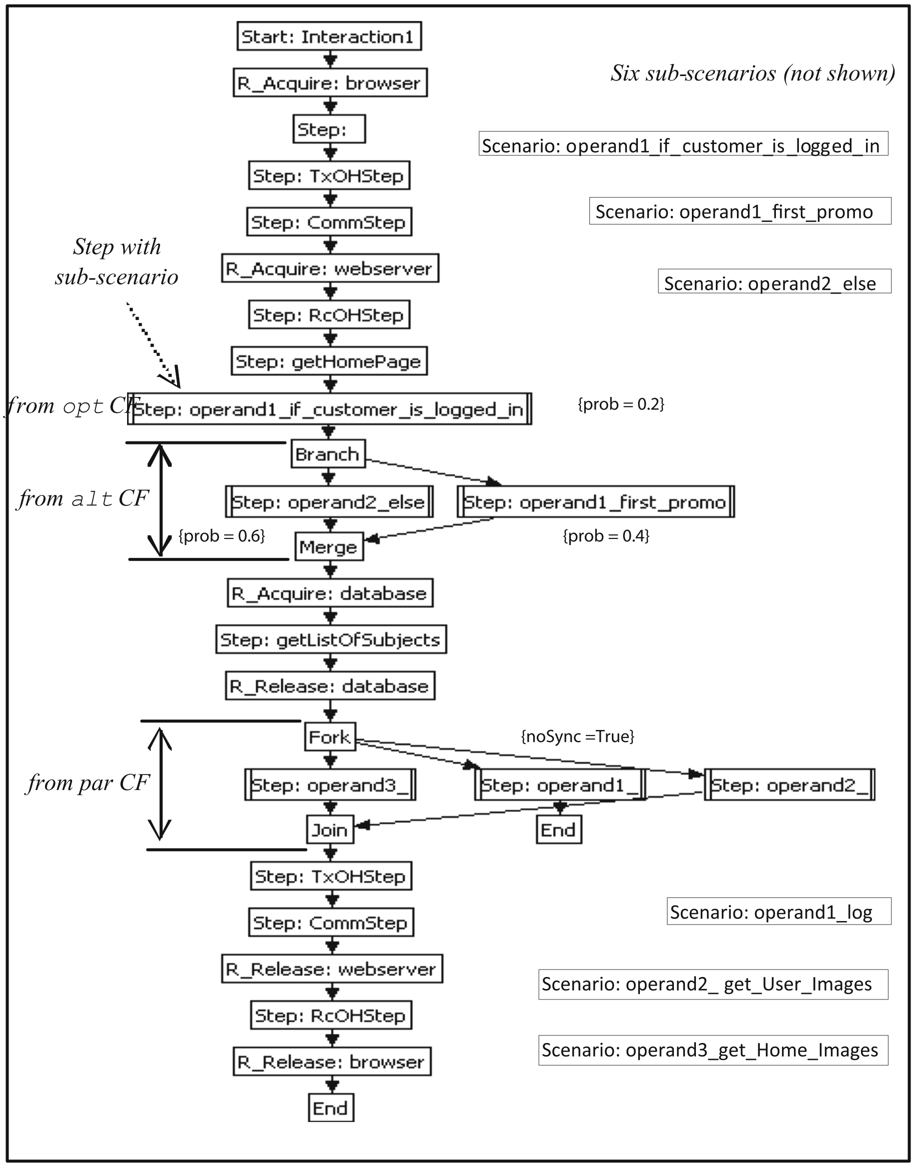


Fig. 10 CSM for the GetHomePage scenario, excluding resources. (The CSM on the *left* is a screenshot from the tool, which uses different presentation conventions from Fig. 7. *Arrows* represent Sequence PCs *except* for those surrounding Branch/Merge/Fork/Join/Start/End PCs, which just indicate associations)

not the case in correct software. For example, when a buffer manager returns a buffer, the holding time of the buffer is not nested in the holding time of the manager process.

Full nesting also corresponds to layering of resource queues in LQN. However, even without it, a correct LQN model can be constructed and solved. The algorithm for generating LQN models detects full nesting as a standard simple case and accommodates exceptions either as “second phases” of a service time which gives analytical solutions [10–12], or by using a special resource-token task (which requires simulation for solving the model).

6.3 Discovering calls between components

Wherever the CSM makes a transition between Components, there is implicitly a message passed (there may or may not be an explicit message description attached to a CommStep). An important feature of LQN is its ability to estimate the performance effect of *blocking calls*, in which the caller waits for a reply. A blocking call/reply pair of messages is identified where:

- the message is explicitly identified by attributes in the CSM;
- the scenario transfers from one Component to another, with a given prior **R**(S), and later returns to the first Component, with the identical **R**(S), and there is no parallel subpath between these points defining nonzero execution by the sending Component.

An advanced LQN feature that arises in real software is a *forwarded request*, in which a sequence of messages traverses several tasks, ending with a reply back to the originating task,

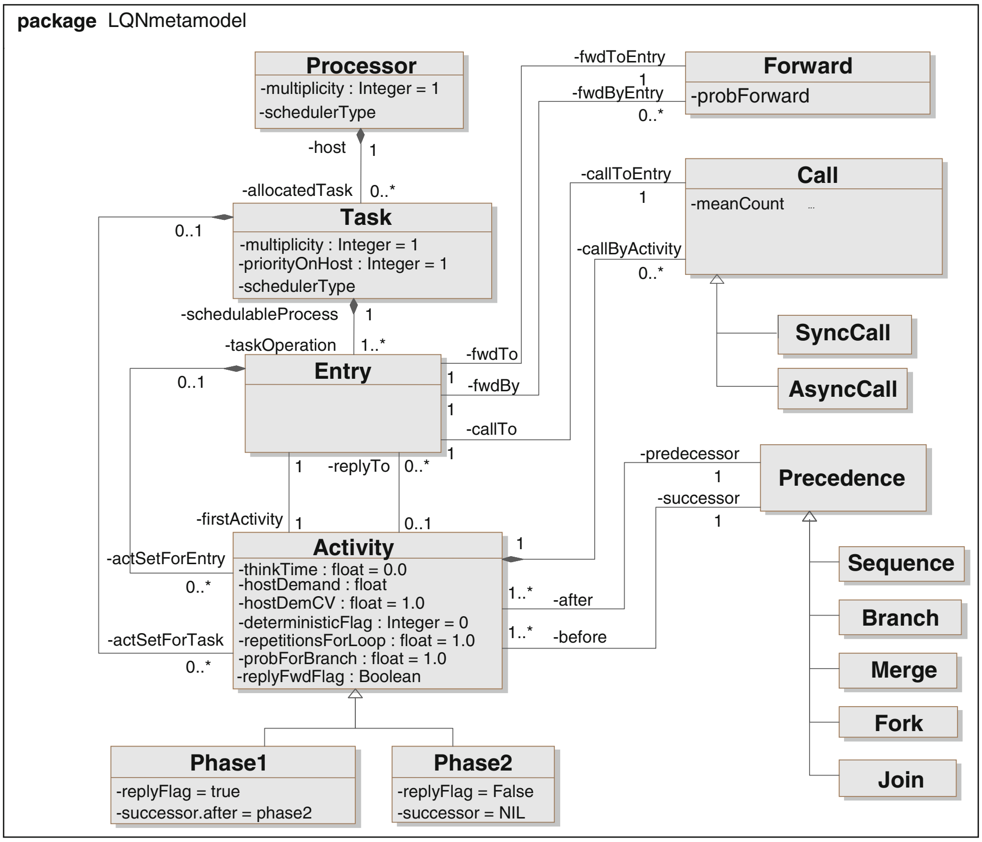


Fig. 11 Simplified LQN metamodel

again with no parallel subpath defining nonzero execution by the sending Component.

Messages are thus categorized as part of blocking call/replies or of forwarding chains, or as asynchronous (the remainder).

7 Transformation from CSM to LQN (C2LQN)

The types of Pmodels used involve different approximations to the behaviour and to contention management, which should be considered by the user but are outside the present scope. The viewpoint of PUMA is that a user should be free to use the performance formalism of their choice, perhaps the one they are most used to, or with available tooling. This paper focuses on transformations from CSM to two Pmodels: LQN presented in this section and Generalized Stochastic Petri Nets (GSPNs) in Sect. 8. These two PModels differ greatly in the way they model resources and how the resources are used by the behaviour, so each section concentrates on the nature of the challenges that had to be addressed for the particular transformation.

Whereas the CSM is a kind of projection of the Smodel extended with MARTE annotations, the Pmodel is formulated in different terms altogether (a large semantic gap). In particular, resources, which are a small part of the Smodel, are central to the Pmodel.

7.1 LQN Pmodel and Metamodel

The LQN model [11,12] is a form of extended QN particularly designed to represent software systems. A simplified LQN metamodel is shown in Fig. 11, and the concepts are illustrated by the example in Table 4 below. Software resources (e.g. process thread pools) are represented as *Tasks* (in the graphical notation, the bold rectangles labelled by the thread pool size) each providing a set of operations called *Entries* (shown as attached rectangles). Each task has a host *Processor* (shown as an oval). The detailed execution of an entry is described by *Activities* (a graph of small rectangles inside the task), with the same precedence relationships as CSM. For each entry, there is a *firstActivity* to begin the execution and a *replyFwdActivity* to send a reply to the caller, or to forward the request to another entry. An *Activity* has execution attributes similar to CSM steps: processing demand, loop repetition, branching probability, and calls for other operations. *Calls* are shown as arrows from an activity to an entry, labelled with the mean number of requests. A

Table 4 C2LQN transformation algorithm: from CSM to LQN

1	Optionally bind aspects, then flatten and normalize the CSM
2	Find and remove simple cycles; stop if there are complex cycles
3	Find the resource context **R**(S) of each Step S, (Sect. 6.1)
4	Traverse the CSM and discover blocking calling interactions and other (asynchronous) calls
5	Create an LQN processor for each CSM ProcessingResource, and an LQN Task from each CSM Component, with corresponding multiplicities
6	For a closed workload definition create a task with multiplicity representing the number of users and an entry with the think time; for an open workload the arrival rate is attached to an entry created for the first Step as in step 7
7	Create calls and entries recursively starting from the Workload tasks; for each call a target entry with a first Activity is created on the task indicated by the first Step of the call (one entry per call). By construction, blocking calls always return to the same entry that made the call. The call frequency is the product (repetitions* probability) of the first Step of the call
8	Create LQN Activities to represent the entry internals from the sequence of additional Steps for the call. The Activities simply copy the CSM Steps and PCs, and where a nested call is discovered, an Activity is created to make it

Call may be blocking (the caller waits for a reply, indicated by a solid arrowhead), asynchronous (no reply) or forwarding (after providing an operation, the receiver forwards the request to another task entry). An operation may be executed in two phases, with the second phase following the reply.

Service requests may be produce a chain of tasks waiting for replies; this chain is called *resource context* of an operation, and the operation duration is part of the service time of each blocked task in the calling chain. Pure logical resources are also modelled as tasks. An LQN model can be solved either with the numerical solver LQNS [11] or by a simulator.

7.2 C2LQN transformation details

A high-level description of the transformation algorithm from CSM to LQN is given in Table 4.

The implemented CSM-to-LQN transformation begins by generating a LQN *Task* or *Host* for each CSM *Component* or *ProcessingResource*, and a user*Task* for each *Workload* of each top-level scenario, with the given *population* and *thinkTime* for closed workloads. For open workloads, the user task has a given *arrivalRate*, an infinite *population* and zero *thinkTime*.

The scenarios are normalized and flattened as described in Sect. 4. In [44], the concept of subscenario was enlarged to include *aspects*, defined as a kind of parameterized subscenario with roles and role bindings. Aspect subscenarios are bound into the CSM using the approach of [44], before flattening, and then treated as normal subscenarios.

CSM cycles constructed with *Branch/Merge* are reduced to subscenarios nested in a repeated *Step*. This is always possible if the loops are structured (that is, fully nested within each other, as provided by structured programming languages). Structured loops that start with a *Merge* and end at a *Branch* back to the *Merge* are found by inspection and reduced, until no structured loop can be found; if there is still a cycle, an LQN cannot be produced.

Starting from the CSM top-level scenario *Start* points, and from the LQN User tasks, for each inferred message that is not a reply there is a *Call* created to a target *Entry* (call it entry E) created in the task corresponding to the target *Component* of the message (call it task T). The *Call* frequency is the product of the *repetition* attribute of the last *Step* before the call, and the *probability* attribute of the first *Step* after. If the*Call* is in a forwarding chain, its forwarding probability is set to unity. From the first *Step*, after the message, the first *Activity* is created in the entry E.

Subsequent *Steps* and PCs (until the next message) generate activity graph entities in the task T which mirror one-to-one the CSM entities. When a message is detected, an *Activity* A is created in task T to be its sender, and (if the message is not asynchronous) the traversal of the CSM proceeds to the next *Component*; eventually, it will return with a reply message and further LQN additions will continue from *Activity* A. If the message from A is asynchronous, there is a fork before A and one subpath continues in task T, while the other subpath proceeds along the message to the next *Component*.

7.3 LQN Pmodel as assembly of multiple scenarios

Figure 12 shows the LQN model obtained by applying the algorithm to the GetHomePage scenario in Fig. 4. An LQN task is generated for each concurrent component corresponding to the lifeline roles stereotyped «*PaRunTInstance*». Note that four LQN tasks in Fig. 12 correspond to the lifeline roles from Fig. 4, while the fifth LQN task, PromoProc, corresponds to a lifeline role inside the two `ref` CFs, Promotion 1 and 2. Each task has one or more entries and for each entry, the first activity is shown within the entry rectangle. The graph of additional activities (if any) is shown in a shaded rectangle attached to the task.

Real systems include several scenarios for different responses, modelled in separate behaviour diagrams, and they are converted separately. TPC-W for example defines 14 scenarios, with a fraction of requests being directed to each one [38]. These scenarios share common resources and may have a performance impact on each other.

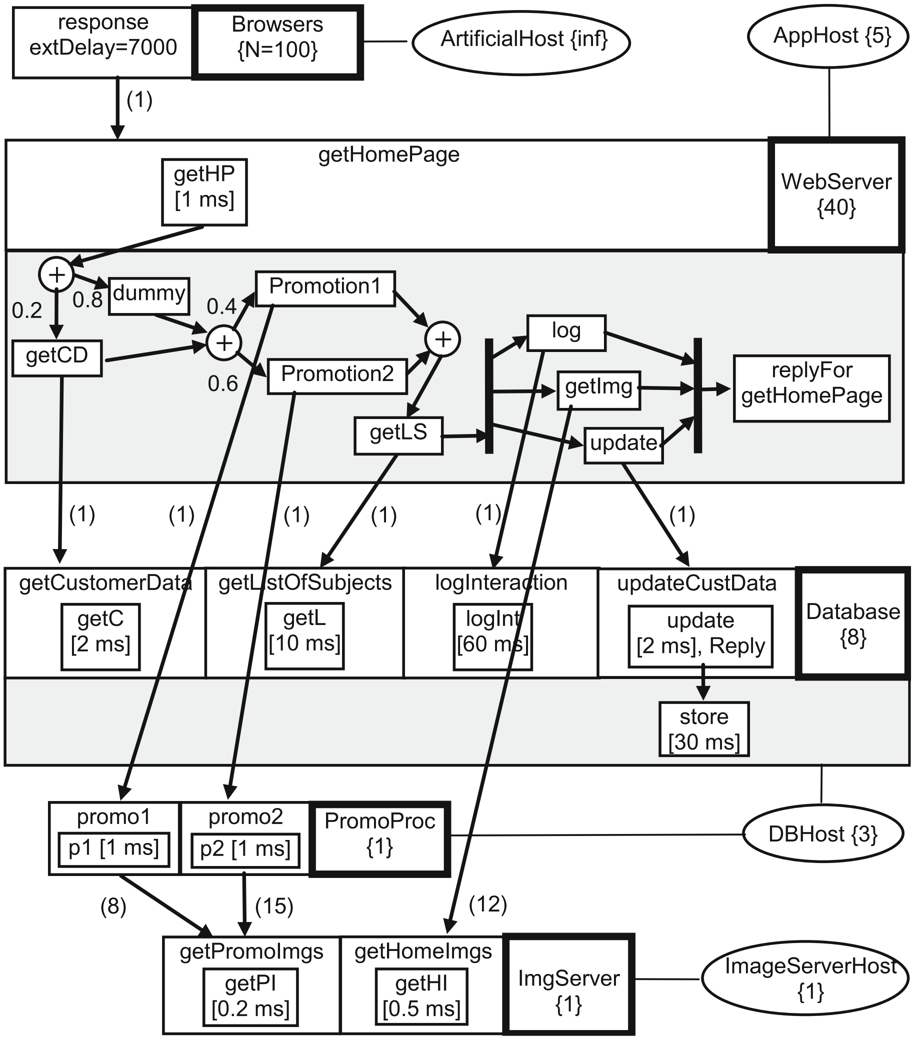


Fig. 12 LQN model for the GetHomePage scenario

To cover the usage profile, the Pmodels for all the scenarios should be combined into a single Pmodel. In LQN for a closed workload, the User tasks can be combined together, using the request fractions to derive a weighted average think time and the probability of requesting each scenario; then, the LQN models found separately are attached to these requests. Each task collects together its entries that were found from the separate scenarios.

This approach was used to model 10 of the 14 scenarios in TPC-W, with the same software components and deployment as shown in Fig 5, giving the LQN Pmodel shown in Fig. 13 in a simplified form without parameters and processors. The LoadGenerator task chooses the scenario in the proportions defined in [38], and drives the Browser task (called here EB) with an entry corresponding to each of the 10 scenarios. It would be a long and error-prone process to produce such a large model manually.

To provide an illustration of the end-to-end application of a Pmodel, experiments were performed on the LQN shown in Fig. 13 for different numbers of users (*Nusers* from 1 to 2,000). The LQN solver solves this model in less than one second. Some model results are shown in Fig. 14. The first case with single processors, single-threaded tasks, an external user delay (think time) of 7 s, and up to 2,000 users gives the curves for high response times and low throughputs in the two graphs. Examination showed that WebServer saturation limited the throughput to about 23 responses/second and a capacity of about 30 users (for the desired 1-s mean response time), which was unsatisfactory. An improved “base” case was defined with 10 WebServer threads, 2 DB threads and 2 DB processors, giving the other curves (lower response time, higher throughputs, and a capacity of about 1,200 users). The additional concurrency gave a satisfactory solution.

A deeper use of the model is to evaluate design changes such as execution in parallel, replication, modified concurrency, and reduced demands and delays. The results evaluate the potential of these changes, which can then be mapped to possible software solutions [36,45]. The choice of the greatest performance improvement for the smallest cost or effort is finally made by the designer.

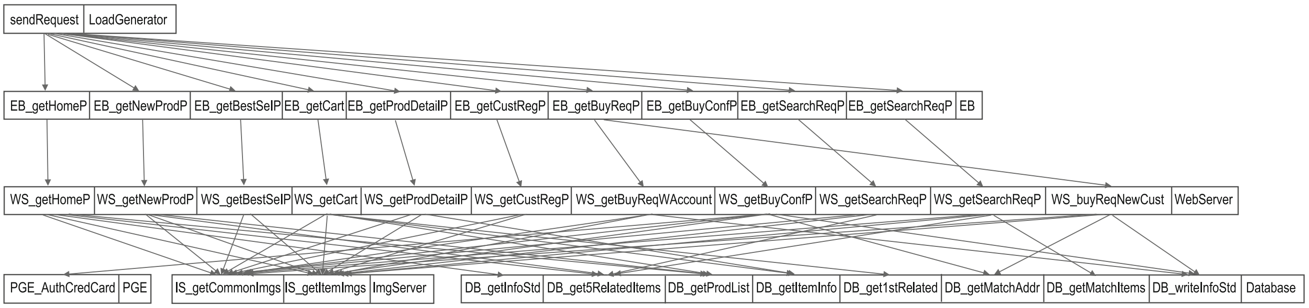


Fig. 13 The LQN model created by merging submodels for ten TPC-W scenarios

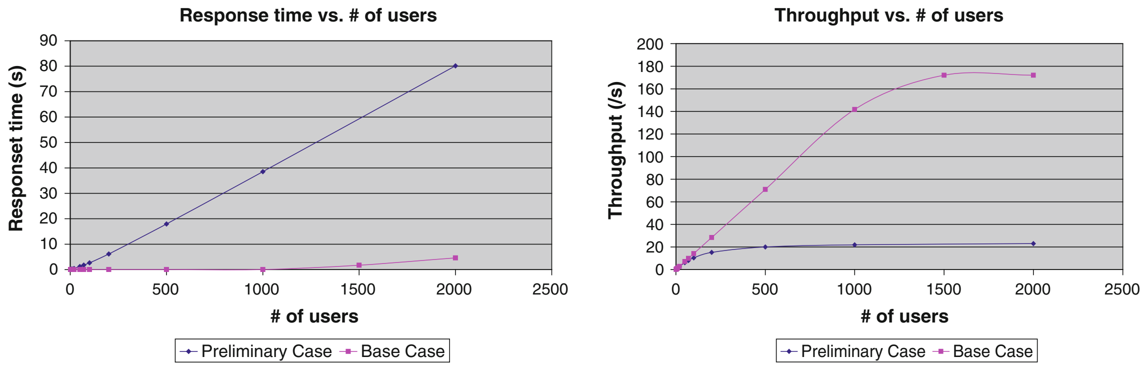


Fig. 14 Results for the “preliminary case” with limited concurrency, and the improved “base case”

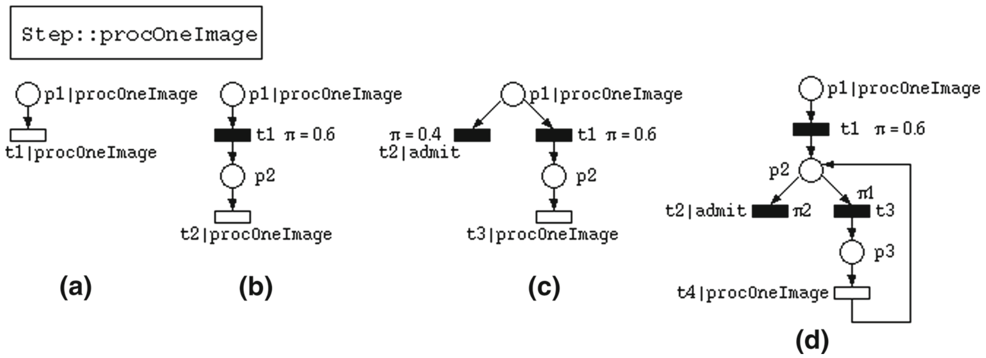


Fig. 15 LGSPN patterns for a step

8 Transformation from CSM to Stochastic Petri Nets (C2PN)

This section presents the transformation from CSM to Stochastic Petri nets and illustrates it with an example of bioinformatics workflow model.

8.1 Transformation approach

Petri net models can represent complex logic, which is impossible in queueing models. They represent system state by tokens in places, and model behaviour by transitions which fire and move tokens from place to place. Time delays are modelled in Generalized Stochastic Petri Nets (GSPNs) [37] by stochastic firing delays. An algorithm has been created for Labelled GSPNs in which subnets are composed based on labels attached to places and transitions [4,5,37]. The GSPN model must be solved by generating its state space, and its main disadvantage is state explosion in the solver. Petri net tools can also carry a variety of correctness analysis, which are beyond the scope of this work.

8.2 Patterns for translation

The algorithm is based on subnets, illustrated in Fig. 15, for translating a Step. Places and transitions are labelled as

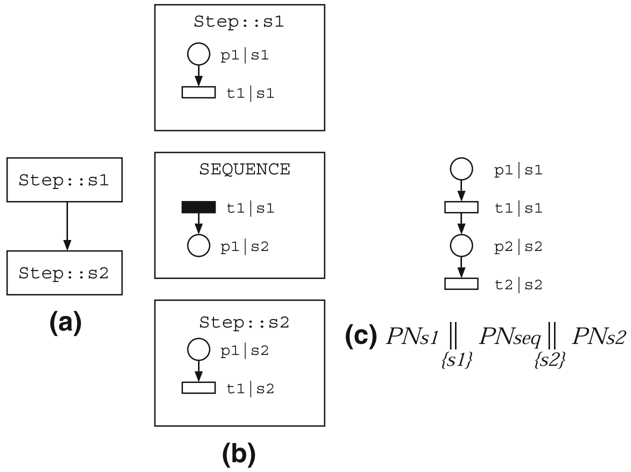


Fig. 16 LGSPN patterns for sequence, and for composing it

"patternName|CSMName". Part (a) shows the subnet for a simple Step, part (b) includes a probability of execution with the additional transition t1, (c) shows a preceding multiway branch, and (d) shows repetition.

Figure 16 shows the sequential composition of two Steps s1 and s2. Part (a) is the CSM, (b) the subnets for the two Steps and the Sequence PathConnector, and (c) the composition in LGSPN terms, based on labels t1|s1 and p2|s2. The patterns for the *Branch, Merge*, *Fork* and *Join* are similar. A Start PathConnector for a closed workload gives a pattern that cycles tokens from the end of a response back to the beginning, after the external delay. Open workloads give infinite state spaces and are not modelled.

Each resource subnet has a place with tokens equal to its multiplicity, and transitions for each requester to allocate and de-allocate tokens. It is composed with patterns representing

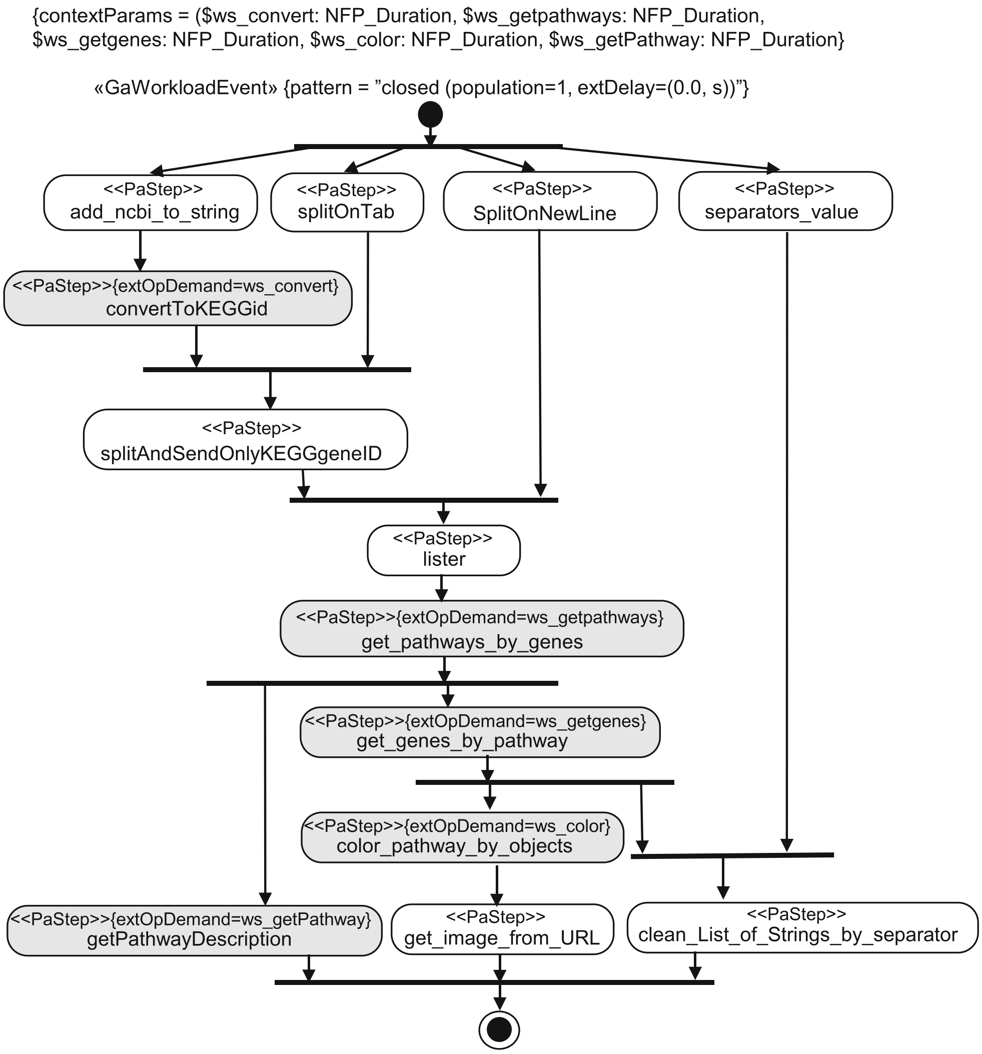


Fig. 17 Annotated activity diagram for the GPGE workflow

acquisition and release. Processor resources are handled in the same way, by introducing acquisition and release transitions before and after each Step.

The translation algorithm first creates a LGSPN pattern subnet for every CSM element. Then, it composes Steps with their host resources, and resources with acquisition/release. Finally, it composes subnets for Start, End and Sequence, followed by other PathConnectors.

8.3 Workflow case study

Petri nets can capture well-workflow models, which can fork/join or branch/merge concurrent branches at will. We have chosen as an example a real workflow model from bioinformatics, found at the archive website myExperiment [13]. The workflow represents the computation of "Get Pathway-Genes by Entrez gene id" (GPGE). Given a specific "entrez" gene id, GPGE returns the set of pathways that this gene participates in, a pathway map, and the genes associated with each pathway.

The workflow model is represented as a UML activity diagram with MARTE annotations in Fig. 17. The activities are stereotyped as PaStep; the ones with white background are executed by the Taverna workflow engine at the user's site, while five activities shown in grey are executed by external web services. These five PaSteps were identified as external operations with the attribute *extOpDemand* (with a count of 1, not shown).

The stereotype «*GaAnalysisContext*» identifies the AD as a subject for performance analysis and its *contextParams* attribute declares five parameters which correspond to the external services delays. A closed workload with a population of 1 and zero think time is associated to the *Start* node corresponds to repeated executions of the workflow, one at a time. The transformation to CSM and then to GSPN had the results shown in Figs. 18 and 19. The sequences of steps in the workflow can easily be traced in each of these models.

The GSPN model was validated against measurements, in order to see how effective it is in predicting the end-to-end

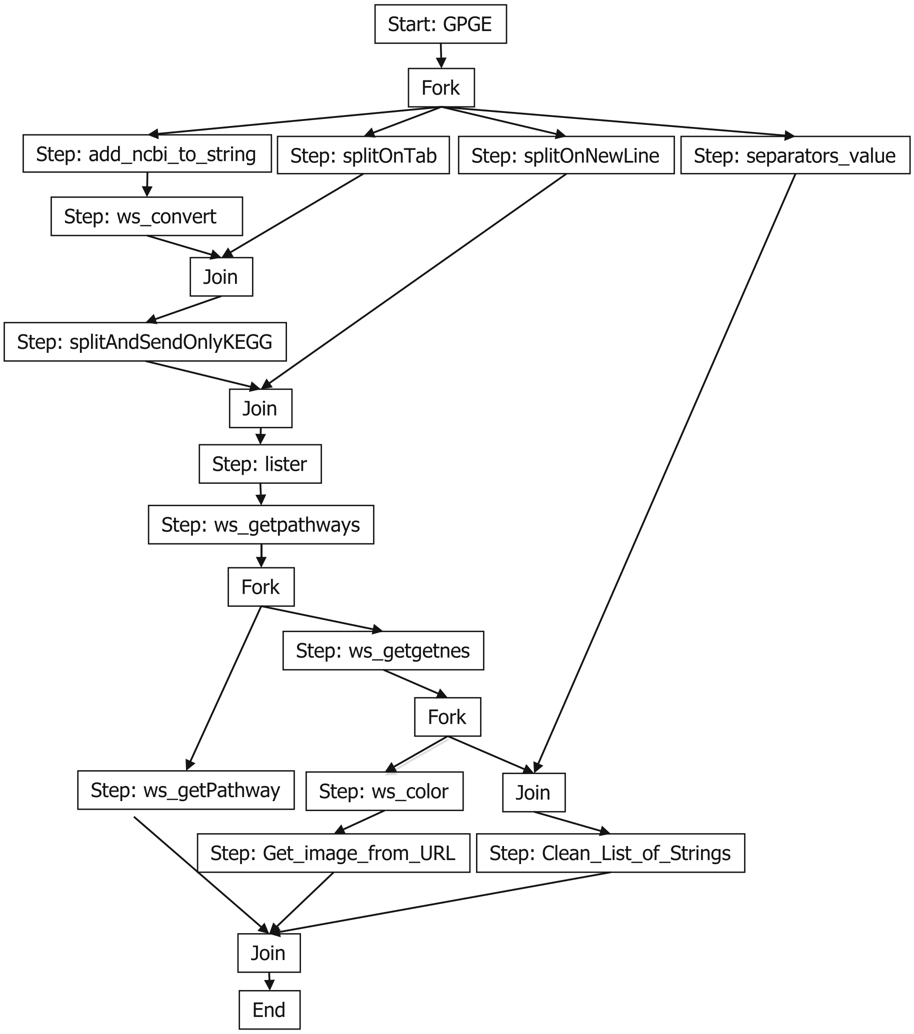


Fig. 18 CSM model of the GPGE workflow

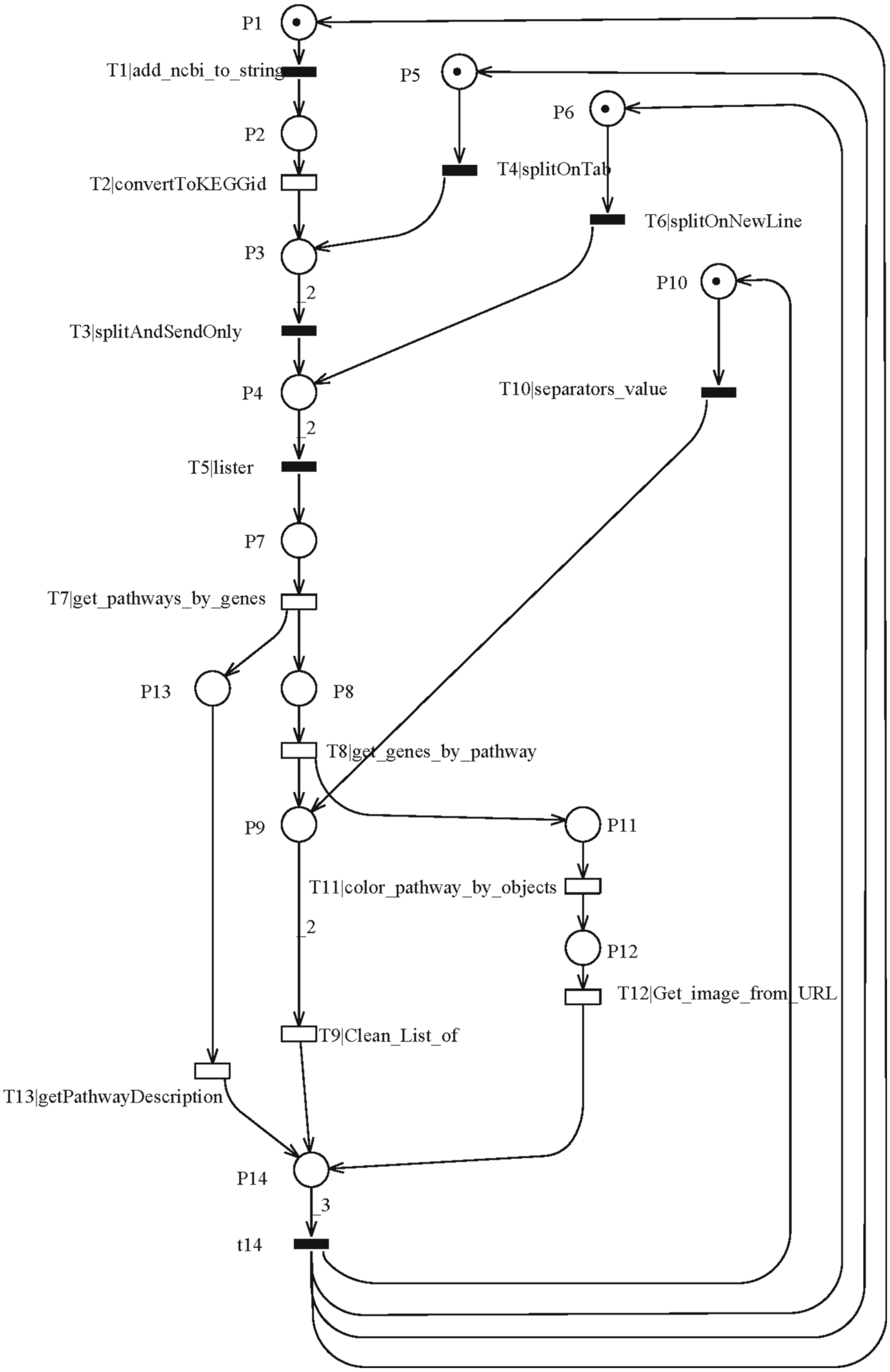


Fig. 19 GSPN Pmodel for the GPGE workflow

delay. The workflow computation was executed and measured, with some tens of executions five times-of-day, at times labelled T1 to T5. The average delay found for each step in the workflow is shown in Table 5. Clearly, the mean values varied greatly with the time of day. The delays for all the workflow steps are given in Table 5.

The measured step delays were inserted in the GSPN model as the average delay to fire the transitions corresponding to the external services. The model was solved for each of the five times-of-day. The results in Table 6 show that the error was always less than 10 %. Given that the external web services used had other unknown workloads, that we knew only their mean delay but not the distribution and that we approximated each delay with exponentially distributed transitions in GSPN, the results are reasonably good.

Table 5 Average workflow operation times and end-to-end delays at five different times-of-day

Web service or local activity	T1	T2	T3	T4	T5
add ncbi to string	111	1700	42	96	75
convertToKEGGid	1.000	1400	982	1100	1300
splitOnTab	14	2000	1	15	21
splitAndSendOnlyKEGGgeneID	74	116	50	94	48
splitOnNewLine	33	1.900	1	35	3
lister	72	84	52	78	60
get_pathways_by_genes	997	1000	897	1000	931
get_genes_by_path_way	1100	1000	918	1100	946
separators_value	21	1900	1	7	2
colour_pathway_by_objects	1800	1700	1700	1800	1500
getPathwayDescription	972	983	867	923	894
Clean_List_of_Strings_by_separator	520	450	108	340	92
Get_image_from_URL	1700	1800	2100	1400	1700
End-to-end workflow delay	7800	11600	8800	7400	9900

Table 6 Prediction error of the GSPN model

	Response time (second)				
Real system (average)	7.8	11.6	8.8	7.4	9.9
GSPN model (prediction)	8.16	10.98	8.14	7.22	9.12
% Prediction error	4.41 %	−5.56 %	−8.11 %	−2.49 %	−8.55 %

9 Conclusions

The PUMA transformations have successfully automated the creation of the types of Pmodel described here (LQN or GSPN) from a UML Smodel and the information in its MARTE annotations, for systems with statistical workloads and performance measures. PUMA can do what a performance specialist would do with the same information; user judgment is still required in determining the annotations and the choice of Pmodel. The transformations can claim a useful level of "completeness" in covering the problem of building a Pmodel because

- the CSM captures and PUMA uses all the information in the Smodel and the annotations which is relevant to Pmodels for these systems (with a few exceptions noted below), and
- the transformations then extract from the CSM all the properties that can be applied in building the target Pmodel.

Some useful properties in the MARTE annotations are not yet included in CSM (such as Step priority and arrival patterns other than open and closed); however, extensions to CSM are planned to cover these and pose no difficulty. Many other properties in MARTE are less useful for the class of systems with stochastic timing properties that we model; such examples are discrete-time clocked behaviour, time-bounded nondeterministic hostDemands, host clock overhead, and many detailed properties for describing operating systems and systems on chip. These are not planned for inclusion.

Performance modelling transformations like PUMA are unusual in that they transform between quite different semantic domains, with different levels of abstraction. The differences have been described in some detail in the context of each stage of the transformations. A second aspect which is not common in software transformation is the need to analyse extensive CSM properties to identify types of messaging interactions between components, and relationships between resource holding times (the analysis in Sect. 6).

The transformation scalability is good. The complexity of the transformations is dominated by the cost of traversing the Smodel and the CSM, which is linear in the number N of CSM Steps in a single top-level scenario (or the number of annotated elements in the Smodel scenario, which is roughly proportional to N). So the complexity of S2C transformations is $O(PN)$ where P is the number of scenarios in the usage profile and N the number of steps in a scenario, which means it is linear in the number of Steps in the Smodel.

The analysis of the CSM described in Sect. 6, which is needed for creating the LQN Pmodel, flags some cases that cannot be handled. These are not shortcomings of PUMA, but warnings of possible problems in the Smodel. One of these, *non-deterministic resource contexts*, is due to a part of a scenario where some resource may or may not be allocated to the process, depending on its history, and may lead to resource allocation errors or deadlocks. A second case, *dubious causality*, is due to a limitation in UML in identifying causality in `alt` CFs. The third case, called *unstructured loops*, is simply due to a limitation of LQN in modelling looping behaviour; some other Pmodel types can be applied instead. However, unstructured loops are essentially the "goto" behaviour that was eliminated by structured programming long ago, and perhaps, they should be eliminated in these cases too.

The key to real progress in software performance engineering lies in the more intelligent use of performance models [36], by themselves and in combination with measurements [40]. Practical automation of performance model-building as achieved by PUMA is an important step towards this goal.

Acknowledgments This research was supported by grants from NSERC, the Natural Sciences and Engineering Research Council of Canada, through its Discovery and Strategic Projects programs.

References

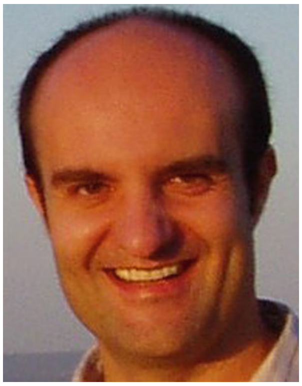
1. Balsamo, S., Marzolla, M.: Simulation modeling of UML software architectures. In: Proceedings of the ESM'03, Nottingham, UK, June 2003
2. Balsamo, S., DiMarco, A., Inverardi, P., Simeoni, M.: Model-based performance prediction in software development. IEEE Trans. Softw. Eng. **30**(5), 295–310 (2004)
3. Becker, S., Koziolek, H., Reussner, R.: The Palladio component model for model-driven performance prediction. J. Syst. Softw. **82**(1), 3–22 (2009)
4. Bernardi, S., Donatelli, S., Merseguer, J.: From UML sequence diagrams and statecharts to analysable Petri net models. In: Proceedings of the 3rd International Workshop on Software and Performance, Rome, July 2002, pp. 35–45
5. Bernardi, S., Merseguer, J.: Performance evaluation of UML design with stochastic well-formed nets. J. Syst. Softw. **80**(11), 1843–1865 (2007)
6. Cavenet, C., Gilmore, S., Hillston, J., Kloul, L., Stevens, P.: Analysing UML 2.0 activity diagrams in the software performance engineering process. In: Proceedings of the 4th International Workshop on Software and Performance, Redwood City, CA, Jan 2004, pp. 74–83
7. Cortellessa, V., Mirandola, R.: Deriving a queueing network based performance model from UML diagrams. In: Proceedings of the Second International Workshop on Software and Performance, Ottawa, Sept. 17–20, pp. 58–70 (2000)
8. Cortellessa, V., Di Marco, A., Inverardi, P.: Model-Based Software Performance Analysis. Springer, Berlin (2011)
9. DiStefano, S., Scarpa, M., Puliafito, A.: From UML to petri nets: the PCM-based methodology. IEEE Trans Softw. Eng. doi:10.1109/TSE.2010.10 12, Jan 2010
10. Franks, G., Petriu, Dorina, Woodside, M., Xu, J., Tregunno, P.: Layered bottlenecks and their mitigation. In: Proceedings of the International Conference on Quantitative Evaluation of Systems, Riverside, CA, Sept 2006
11. Franks, G., Woodside, M.: Multiclass multiservers with deferred operations in layered queueing networks, with software system applications. In: Proceedings of the 12th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems (MASCOTS, 2004), pp. 239–248, Volendam, The Netherlands, Oct 2004
12. Franks, G., Al-Omari, T., Woodside, M., Das, O., Derisavi, S.: Enhanced modeling and solution of layered queueing networks. IEEE Trans. Softw. Eng. **35**(2), 148–161 (2009)
13. Goble, C.A., et al.: Myexperiment: a repository and social network for the sharing of bioinformatics workflows. Nucleic Acids Research 38 (suppl 2) (2010). See also http://www.myexperiment.org/workflows/
14. Grassi, V., Mirandola, R., Sabetta, A.: From design to analysis models: a kernel language for performance and reliability analysis of component-based systems. In: Proceedings of the 5th International Workshop on Software and Performance, pp. 25–36, Palma, Spain, July 2005
15. International Standards Organization, Software and System Engineering: High-level Petri nets-Part 1: concepts, definitions and graphical notation. ISO/IEC document 15909–1 (2004)
16. Jain, R.: The Art of Computer Systems Performance Analysis. Wiley, New York (1991)
17. Jansen, D., Hermanns, H.: QoS modeling with UML statecharts. Proc. ACM Sigmetrics **32**, 28–33 (2005)
18. Kahkipuro, P.: UML based performance modeling framework for object oriented systems. In: UML99, The Unified Modeling Language, Beyond the Standard, LNCS 1723, pp. 356–371, Springer (1999)
19. Khan, R.H., Heegaard, P.E.: Translation from UML to Markov model: a performance modeling framework for managing behavior of multiple collaborative sessions and instances. In: 3rd IEEE International Conference on Computer Science and Information Technology (ICCSIT), pp. 677–686 (2010)
20. López-Grao, J.P., Merseguer, J., Campos, J.: From UML activity diagrams to Stochastic Petri Nets. In: Proceedings of the 4th International Workshop on Software and Performance, pp. 25–36, Redwood City, CA, Jan. 2004
21. Marzolla, M., Balsamo, S.: UML-PSI: the UML performance simulator. In: Proceedings of the First International Conference on the Quantitative Evaluation of Systems (QEST'04), Enschede, pp. 340–341 (2004)
22. Menasce, D., Gomaa, H.: A method for design and performance modeling of client/server systems. IEEE Trans. Softw. Eng. **26**(11), 1066–1085 (2000)
23. Merseguer, J.: Software performance engineering based on UML and Petri nets. Ph.D. thesis, University of Zaragoza, Spain, March 2003
24. Object Management Group. UML Profile for MARTE Modeling and Analysis of Real-Time and Embedded Systems, Version 1.0, OMG doc. formal/2009-11-02, Dec 2009
25. Object Management Group. Unified Modeling Language (UML), v2.3—Superstructure, document formal/10-05-05, May 2005
26. Object Management Group: UML profile for schedulability, performance, and time specification, version 1.1, OMG document formal/05-01-02, Jan 2005
27. Petriu, D.C., Gu, G.: From UML to LQN by XML algebra-based model transformations. In: Proceedings of the 5th ACM Int Workshop on Software and Performance (WOSP 05), pp. 99–110, Palma, June (2005)
28. Petriu, D.C., Sabetta, A.: From UML to performance analysis models by abstraction-raising transformation. In: Babau, J.P., Champeau, J., Gerard, S. (eds.) From MDD Concepts to Experiments and Illustrations, pp. 53–70, ISTE Ltd. (2006)
29. Petriu, D.C., Shen, H.: Applying the UML performance profile: graph grammar-based derivation of LQN models from UML specifications. In: Proceedings of the 12th International Conference on Modeling Tools and Techniques for Computer and Communication System Performance Evaluation, London, England (2002)
30. Petriu, D.B., Woodside, C.M.: An intermediate metamodel with scenarios and resources for generating performance models from UML designs. Softw. Syst. Model. **6**(2), 163–184 (2007)
31. PUMA project web page http://sce.carleton.ca/rads/puma
32. Sibertin-Blanc, C., Hameurlain, N., Tahir, O.: Ambiguity and structural properties of basic sequence diagrams. **4**(3), 275–284 (2008)
33. Smith, C.U., Lladó, C.M.: Performance model interchange format (PMIF 2.0). In: Proceedings of the QEST 2004 (First International Conference on Quantitative Evaluation of Systems) Enschede, Sept 2004

34. Smith, C.U., Williams, L.G.: Software performance AntiPatterns. International CMG Conference, pp 797–806, Dec 2001
35. Smith, C.U.: Introduction to software performance engineering: origins and outstanding problems. In: International Summer School on Formal Methods for Performance Evaluation, pp. 395–428, Bertinoro, Springer (2007)
36. Smith, C.U., Williams, L.G.: Performance Solutions. Addison-Wesley, Boston (2002)
37. The GreatSPN tool. Available at http://www.di.unito.it/~greatspn
38. Transaction Processing Council: TPC Benchmark W (Web Commerce) Specification, Version 1.8 (2002)
39. Tribastone, M., Gilmore, S.: Automatic translation of UML sequence diagrams into PEPA models. In: Proceedings of 5th International Conference on Quantitative Evaluation of Systems (QEST 2008), pp. 205–214, St Malo, France (2008)
40. Woodside, M., Franks, G., Petriu, D.C.: The future of software performance engineering. In: Proceedings of the Future of Software Engineering 2007, pp 171–187, IEEE Computer Society, May 2007
41. Woodside, M., Petriu, D.C., Petriu, D.B., Shen, H., Israr, T., Merseguer, J.: Performance by unified model analysis (PUMA). In: Proceedings of the 5th International Workshop on Software and Performance (WOSP'2005), pp. 1–12, Palma de Mallorca, July 2005
42. Woodside, M., Petriu, D.B., Siddiqui, K.H.: Performance-related completions for software specifications. In: Proceedings of the 24th International Conference on Software Engineering, May (2002)
43. Woodside, M.: Software resource architecture. Int. J. Softw. Eng. Knowl. Eng. (IJSEKE) **11**(4), 407–429 (2001)
44. Woodside, M., Petriu, D.C., Petriu, D.B., Xu, J., Israr, T., Georg, G., France, R., Bieman, J.M., Houmb, S.H., Jurjens, J.: Performance analysis of security aspects by weaving scenarios from UML models. J. Syst. Softw. **82**(1), 56–74 (2009)
45. Xu, J.: Rule-based automatic software performance diagnosis and improvement. Perform. Eval. **67**(8), 585–611 (2010)
46. Zeng, Y.X.: Transforming Use Case Maps to the Core Scenario Model Representation. MASc thesis, University of Ottawa, May (2005)

Author Biographies

Murray Woodside does research in all aspects of performance and dependability of software. Much of this work is based on a special form of queueing analysis called layered queueing (also known as "active servers") which he and his co-workers have applied to distributed systems of many kinds. This analysis supports software engineering through the concept of resource architectures, and performance engineering in general. He received the PhD degree in Control Engineering from Cambridge University, England, and has taught and done research in stochastic control, optimization, queuing theory, performance modelling of communications and computer systems, and software performance. He taught at Carleton University in Ottawa until his recent retirement, and now continues research and teaching with an appointment as Distinguished Research Professor. He is an Associate Editor of the IEEE Transactions on Software Engineering. In the period 1995–1999, he was Vice-Chair and Chair of Sigmetrics, the ACM Special Interest Group on performance. He is a Fellow of IEEE.

Dorina C. Petriu is a full professor in the Department of Systems and Computer Engineering at Carleton University, Ottawa, ON, Canada. Her main research interests are in the areas of software performance and dependability modelling and model-driven development, with emphasis on integrating the analysis of non-functional properties into the software development process. She served in the steering and programme committees of numerous international conferences and workshops. Dr. Petriu is a Fellow of the Canadian Academy of Engineering and of the Engineering Institute of Canada, senior member of IEEE and member of ACM.



José Merseguer received BS and MS degrees in computer science and software engineering from the Technical University of Valencia, and a PhD degree in computer science from the University of Zaragoza. He is currently the Director of the Master in Computer Science and Systems Engineering at the University of Zaragoza, Spain. He teaches software engineering courses at graduate and undergraduate levels at the University of Zaragoza. Dr. Merseguer has developed postdoctoral research with Prof. Murray Woodside at Carleton University, Ottawa, Canada, and with Prof. Robyn Lutz at Iowa State University, USA. He has also been a Visiting Researcher at the Universities of Torino and Cagliari in Italy, and Politecnico di Milano, Italy. His main research interests include performance and dependability analysis of software systems, UML semantics, and service-oriented software engineering. Dr. Merseguer has been serving as a referee for international journals and as a programme committee member for several international conferences and workshops. He is programme co-chair for ICPE'14. He is also co-author of the book "Model-driven Dependability Assessment of Software Systems" by Springer.

Dorin B. Petriu holds Bachelor's and Master's degrees in computer system engineering from Carleton University, Ottawa. Dorin is completing his doctoral programme part-time while he works as a policy analyst at the Office of the Commissioner of Lobbying of Canada and bicycles around Ottawa, Canada. Dorin's research interests are modelling and automatic model transformations.

Mohammad Alhaj received the B.Sc. degree in Electrical Engineering from University of Jordan, Amman, Jordan, in 1994, the M.A.Sc. degree in System and Computer Engineering - Software Engineering from Carleton University, Ottawa, Canada, in 2008. He is currently working towards the PhD degree in System and Computer Engineering at Carleton University. His research interests include software performance engineering, performance models (queuing networks and extensions), model-driven development (UML) and model transformations. His current research topic is focused on performance analysis of Service-Oriented Architecture (SOA) systems using model-driven SOA.