CSM2LQN – Transformations for the Generation of Performance Models from Software Designs

by
Dorin B. Petriu

A thesis submitted to the Faculty of Graduate and Postdoctoral Affairs in partial fulfillment of the requirements for the degree of

Doctor of Philosophy
in
Electrical and Computer Engineering

Carleton University
Ottawa, Ontario, Canada

ABSTRACT

This thesis presents a methodology for the tool-assisted generation of performance models from software designs. First, the author has defined the Core Scenario Model (CSM) as an intermediate model to capture performance information from a software behaviour specification. Second, he has examined in detail the problem of transforming CSMs derived from UML annotated with MARTE performance stereotypes into performance models.

The thesis presents a set of algorithms for transforming CSMs into other CSMs in order to: enforce the correctness of the associations defined in the metamodel; clean up CSMs with minor syntactic flaws; and normalize CSMs from heterogeneous software designs in order facilitate the generation of performance models. The thesis also presents algorithms for weaving CSM aspect sub-models and for generating Layered Queueing Network (LQN) and Queueing Network (QN) models from CSMs.

Three substantial case studies of service systems defined in UML, automatically generated as CSMs with tools developed by others, and then automatically generated LQNs using a CSM2LQN tool based on Eclipse and which implements the algorithms developed by the author.

The advantages of the methodology presented here are that it captures emergent system behaviour and its associated resource use in a manner that accounts for blocking interactions and does not lose the performance impact of the layered resource architecture.

ACKNOWLEDGEMENTS

This research would have never come to fruition without the support and guidance of the best supervisor I could imagine: Dr. C. Murray Woodside. His continuous support, encouragement and direction have been the building blocks for this research. Words cannot convey my deep appreciation for his understanding and friendship throughout the years.

My family has been an equally invaluable source of support. A big thank you to my parents, Dorina and Emil. You have my undying gratitude. Thank you too to my sister, Mihaela. You may not realize this, but your tenacity is truly a source of inspiration. Finally, thank you to all my friends and co-workers who have supported me and put up with hearing "*I can't. I have to work on my thesis.*" all these years. It actually paid off!

Carpe diem.

Dorin

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

GLOSSARY

Active Resource – CSM resource that can execute service demands

Activity – LQN element for modeling detailed service demands

Asynchronous call – LQN call where the caller continues executing and does not wait for a reply

Branch – CSM PathConnection with a single source Step and multiple target Steps; the target Steps are executed as alternatives

BSS – Building Security System

Closed Workload – CSM workload with a fixed population that cycles through the system

Refinement – CSM element contained by a Step which references a sub-Scenario

Component – CSM resource with access control and a host

CSM – Core Scenario Model

End – CSM PathConnection with a single source Step and no target; finishes a Scenario

Entry – LQN element denoting a service provisioning point

Fork – CSM PathConnection with a single source Step and multiple target Steps; the target Steps are executed in parallel

Forwarding Call – LQN call where a server forwards a synchronous call to another server, which then becomes responsible for sending a reply to the caller

Join – CSM PathConnection with multiple source Steps and a single target Step; the source Steps are executed in parallel

LogicalResource – CSM resource without access control

LMP – Labour Market Portal

LQN – Layered Queueing Network

MARTE – UML Profile for Modeling and Analysis of Real Time and Embedded systems

Merge – CSM PathConnection with multiple source Steps and a single target Step; the source Steps are executed as alternatives

Open Workload – CSM workload with external arrivals that execute once and do not cycle through the system

PathConnection – CSM element connecting Steps together in a Scenario

Path segment – a partial CSM scenario; a sequence of Steps

ProcessingResource – CSM resource that can be a host for software resources

PUMA – Performance though Unified Model Analysis

QN – Queueing Network

ResourceAcquire – CSM Step describing the acquisition of some resource

ResourceRelease – CSM Step describing the release of some resource

ResourcePass – CSM Step describing the explicit passing of some resource

Scenario – CSM element combining Steps in order

Semaphore Task – LQN Task which models access control to logical resources; has a synchronously called Entry for acquisition/locking and a paired asynchronously called Entry for release/unlocking

SPT – UML Profile for Schedulability, Performance and Timing

Sequence – CSM PathConnection with a single source Step and a single target Step

Start – CSM PathConnection with no source and a single target Step; begins a Scenario

Step – CSM element describing an operation; a Step is connected to a single predecessor PathConnection and a single successor PathConnection

Synchronous Call – LQN call where the caller blocks and waits for a reply

Task – LQN element providing services; process

TPC-W – Transaction Processing Council transactional Web benchmark

UML – Unified Modeling Language

1. INTRODUCTION

Software development is driven by both functional requirements (FR) and non-functional requirements (NFR). Functional requirements usually encompass the "what" attributes of the system (i.e. what the system does and what functions it provides) whereby the so-called non-functional encompass the "how" attributes (i.e. how fast the system performs, how many users the system can handle, how reliable the system is, etc.). Although the "how" system attributes do not necessarily provide first-order functionality, they do describe crucial attributes that are also critical for the system's proper operation. A unified software design methodology should thus include non-functional specification and analysis, and should consider performance concerns from the very outset of a project. Software Performance Engineering (SPE) is concerned with analyzing performance NFRs such as the response time, delay and throughput of software systems and also predicting the systems' scalability and sensitivity to increased workloads. The goal of SPE is to use predictive performance models to analyze the effect of software features on performance for systems with timing and capacity requirements. SPE aims to insure that software products under development will meet their performance requirements. In order to do so, SPE should begin early in the software lifecycle, before serious barriers to performance are frozen into the design and implementation [112]. The earliest decisions that can constrain the performance of a software system are made during requirements analysis, when the sequence of operations is developed.

Early performance analysis has been shown to be effective in avoiding performance disasters in software projects [114]. However, it takes time and effort to derive the

necessary performance models and the transfer of designer knowledge into the performance model is slow and expensive in practice [101].

The way to provide continuous, cost-effective, and timely performance analysis during the evolution of a project is by providing automation to the software designers. There needs to be automation in generating the necessary performance models from requirements and early design models. There must be some automation in setting up and running the performance analysis. Finally, there needs to be some automation in feeding back the performance results to the developers so they can make changes to the requirements and designs. This approach is in line with the principles behind Model-Driven Analysis (MDA) and Model-Driven Engineering (MDE) as described in [84].

The Unified Modeling Language (UML) Profile for Modeling and Analysis of Real-Time and Embedded Systems (MARTE) [87] has been developed to assist in the capture of performance data and automation of the model-building step. MARTE built on the earlier UML Profile for Schedulability, Performance and Time (SPT) [86]. MARTE and SPT make the analysis more accessible to developers using UML who are concerned about performance issues in their designs. MARTE covers a broad range of applications; from embedded systems with schedulability concerns, to business systems. MARTE v1.0 was ratified and released by the Object Management Group (OMG) in November 2009 and MARTE v1.1 was released in June 2011.

The research underpinning the work presented started with the SPT Profile and migrated to MARTE once it was officially released. The focus of the research is on applications with non-deterministic behaviour and statistical performance requirements. These are

common in distributed information processing such as telecommunications, business systems and web services.

Following the MDA/MDE approach, the Performance through Unified Model Analysis (PUMA) modeling architecture [125] has been developed to facilitate the automation of the SPE process with respect to UML. PUMA is an open tool architecture for attaching performance tools to UML tools. PUMA uses the Core Scenario Model (CSM) [93] to extract and audit performance information from different kinds of design models, and to support the generation of different kinds of performance models. PUMA involves several standards: UML and its interchange standard XMI, MARTE, CSM [93] [94], as well as other performance modeling languages [147] [136].

With suitable tools and model translators, PUMA can support *multiple* specification languages and *multiple* performance models.

1.1. CONTRIBUTIONS

The research tackles the problem of defining transformations for the generation of layered performance models from CSM, as well as providing mechanisms to report and feedback the results of the evaluation of those performance models. It concentrates on CSMs that are generated from UML specifications, but the LQN generation algorithms apply equally to CSMs from any other source. The contributions of this thesis are:

1. Design and definition of the CSM as an intermediate language
    - Semantic rules for the validation of CSM behaviours
2. Algorithms and rules for transforming CSMs into other CSMs in order to:
    - Clean up CSMs with minor syntactic flaws
    - Simplify CSMs

- Discover calling interactions in CSM scenarios
- Weave in CSM aspect sub-models

3. Algorithms for generating performance models from CSM

   - Algorithm to generate Layered Queueing Network (LQN) models
   - Algorithm to generate Queueing Network (QN) models

4. Case studies to demonstrate the application of the CSM transformations

   - A case study based on a building security system (BSS)
   - A case study based on a standardized transaction processing system (TPC-W)
   - A case study based on an enterprise-scale labour market information and job board website (LMP)

5. Co-authorship of published papers based on this thesis (listed in reverse chronological order of publication):

   - Murray Woodside, D.C. Petriu, J. Merseguer, Dorin B. Petriu, Mohammad Alhaj, "Transformation challenges: from software models to performance models", Software and Systems Modeling, published online, DOI: 10.1007/s10270-013-0385-x, 2013 [128]
   - Murray Woodside, D.C. Petriu, Dorin B. Petriu, J. Xu, T. Israr, G. Georg, R. France, J.M. Bieman, S.H. Houmb, J.Jürjens, "Performance Analysis of Security Aspects by Weaving Scenarios Extracted from UML Models", The Journal of Systems and Software Special Issue WOSP'2007, Vol.82, pp. 56–74, 2009 [125]
   - D.C. Petriu, Murray Woodside, Dorin B. Petriu, J. Xu, T. Israr, G. Georg, R. France, J. Jürjens, J.M. Bieman, S.H. Houmb, "Performance Analysis of Security Aspects in UML Models", Proc. of 6th ACM Workshop on Software and Performance, pp. 91-102, Buenos Aires, Argentina, February 5-8, 2007 [100]
   - Dorin B. Petriu, Murray Woodside, "An Intermediate Metamodel with Scenarios and Resources, for Generating Performance Models from UML Designs", SoSym Special Issue on UML'04, Software and Systems Modeling (SoSyM), Springer-Verlag, August 2006 [96]

- Dorin B. Petriu, Murray Woodside, “Software Performance Models from System Scenarios”, Journal on Performance Evaluation (PEVA), vol. 61, issue 1, pp. 65-89 Elsevier, June 2005 [95]
    - appeared on the ScienceDirect “Top 25 Hottest Articles for Performance Evaluation” list
        - #1 Apr-Jun 2005
        - #7 Jul-Sep 2005
        - #17 Oct-Dec 2005
        - #23 Jan-Mar 2006
- Murray Woodside, D.C. Petriu, Dorin B. Petriu, H. Shen, T. Israr, J. Merseguer, “Performance by Unified Model Analysis (PUMA)”, Proc. WOSP 2005, Palma de Mallorca, pp. 1-12, July 2005 [125]
- Dorin B. Petriu, Daniel Amyot, Murray Woodside, B. Jiang, “Traceability and Evaluation in Scenario Analysis by Use Case Maps”, in Scenarios, LNCS 3466, pp. 134-151, Springer-Verlag, 2005 [94]
- Dorin B. Petriu, Murray Woodside, "A Metamodel for Generating Performance Models from UML Designs, Proc UML 2004, Lisbon, LNCS 3273, pp. 41-53, Springer-Verlag, October 2004 [93]

2. BACKGROUND

There has been considerable effort expended on methods for Software Performance Engineering (SPE), and an overview can be obtained from the proceedings of the international Workshops on Software and Performance (WOSP'98 [132], WOSP'00 [134], WOSP'02 [135], WOSP'04 [136], WOSP'05 [137], WOSP'07 [138]) and its successor, the International Conference on Performance Engineering (ICPE'10 [140], ICPE'11 [141], ICPE'12 [142], ICPE'13 [143], ICPE'14 [144]) after WOSP joined with SPEC International Performance Evaluation Workshop (SIPEW) in 2010. A standard reference is the book by Smith and Williams [114], and more recently Cortellessa, Di Marco and Inverardi authored a reference text that surveys a variety of SPE concepts and methodologies [28] in 2011.

2.1. FROM DESIGN TO PERFORMANCE EVALUATION – AN OVERVIEW

The main tenet of SPE is that systems need to be evaluated for performance early in the software development cycle. Various approaches for capturing requirements and design information for performance evaluation have been explored.

Scenario specifications, which describe the system behaviour during a response, provide a powerful starting point for system design and for analysis of various kinds of requirements. The sequence of operations for the system is developed during requirements analysis and that is often when the earliest decisions that can constrain the performance of a software system are made.

Many different notations have been used to capture scenarios, and this masks their common features. From a historical perspective, scenarios have only been adopted as an

approach to software design relatively recently. Carroll [21] gives a broad discussion of the significance of scenarios and argues that scenario-based techniques avoid premature commitment, contain complexity and maintain focus on the essential problem in a new application. Scenarios also facilitate performance evaluation during design, rather than after, since they identify end-to-end use cases of the system and can thus be used to isolate the performance-sensitive responses. It should be noted that Carroll's work is focused on user interface design, but his arguments also apply to software development in general. Buhr *et al.* utilized a scenario-based approach for capturing design information for performance when building their "software CAD" tool described in [20].

One of the originators of SPE, Smith, uses a scenario-based approach for creating performance models. Smith developed an approach to building QNs based on scenario-like Execution Graphs [112] [114] that are specially created for performance analysis. Use Case Maps (UCMs) are a graphical language specifically used for expressing scenarios, and for experimenting with scenario interactions and architecture [18] [53]. The UCM notation is part of the User Requirements Notation, standardized by the International Telecommunication Union Telecommunication Standardization Sector (ITU-T) as Z.151 [52]. Among the numerous scenario notations surveyed in [1], UCMs are notably fit for many requirements engineering activities and for transformations to other modeling languages. A UCM tool, the *Java UCM Navigator* (jUCMNav) [79] (and its C++ predecessor UCMNav [74]) has been augmented with CSM export capabilities to assist with the early analysis of performance questions from scenario specifications [146]. Both tools have already been used in various SPE case studies [92] [94] [109] and in SPE

graduate courses. The author has also developed a method of generating LQN performance models from Use Case Maps (UCMs) as part of his Master's thesis [88]. Cortellessa's comparison of various performance meta-models in [27] provides a useful overview of these various approaches.

Another possible starting point for the SPE process occurs a little later in the design process after the architectural design stage. Balsamo, Inverardi and Mangano have developed an architecture modeling technique [7], and a general framework for modeling called "resource architecture", is described in [122].

Several performance evaluation frameworks use SDL and in particular scenarios expressed as Message Sequence Charts (MSCs) as their starting point for capturing the design information. Dulz *et al*. [34] and Kerber [59] supplemented MSCs [51] with performance information to generate SDL [48] specifications. Mitschele-Thiel [76] also augments MSCs with workloads, processors and processor mappings to generate QNs from SDL. An overview of other SDL-based performance evaluation frameworks includes the following:

- Petri Nets approach – SDL to SDL-net, SDL to Queueing Petri Nets [12]
- attaching time to actions – SPECS: SDL Performance Evaluation of Concurrent Systems [21]
- execution of generated code on emulated target hardware – SPEET: SDL Performance Evaluation Tool [118]
- usage of full-fledged performance evaluation environments – HIT [15], SteelCentral NetModeler Suite (formerly OPNET Modeler Suite) [148]
- attaching timed machines to actions – QUEST [102] and Queueing SDL (QSDL) [30]
- MSC-based performance evaluation and optimization (DO-IT Toolbox)
- coupling of SDL specification with simulation tools – Easy-Sim (combines Geode-SDL and SES Workbench) [107]

Similarly, UML behaviour models [83] can also have performance annotations [86]. Kähkipuro uses performance-oriented UML models (in addition to the design model) to generate performance models [58], whereas Woodside *et al*. extract performance models from UML-like designs in a CASE tool [123].

To evaluate a software or system design for non-functional properties such as performance, reliability, and security, it is necessary to attach suitable annotations to the design. The UML profiles SPT and MARTE, which have this purpose, are described later in this chapter. Such annotations have been used to generate many different kinds of performance models, based on queues [26] [112], layered queues [96] [98], stochastic Petri nets [15] [32] [65], stochastic process algebra [24], and simulation [8]. A survey of methods for building performance models from UML specifications is given in [9]. Translations from annotated UML into different kinds of performance models have been described, for example:

- into queueing models, by Smith [112]
- into layered queueing models [87][94]
- into stochastic Petri nets [8][29][61]
- into stochastic process algebra models [17]
- directly into simulation models [1]

The model translation can be somewhat intricate. For example, D.C. Petriu and Shen use a technique based on graph grammars in [97].

The approaches these authors have taken to interpreting UML are affected by the target performance semantics. Further, each contribution addresses one kind of UML sub-model, such as sequence diagrams, activity diagrams and state machines, and these diagrams do not express behaviour in the same way. This limits the applicability of these methods.

A different performance evaluation approach is to look at the system in a qualitative way that analyses the possible impacts of different system aspects on product qualities such as performance. For example, Chung *et al.* have developed a general approach for analyzing non-functional requirements that they call NFR [25], which has been applied for performance by their co-author Nixon [81]. A similar qualitative analysis, applied to the slightly later stage of system architectural design, is described by Bass, Clements, and Kazman [11]. On a slightly different note, Sancho *et al.* [105] define an ontology for reasoning about performance based on Chung's Softgoal Interdependency Graphs [25]. The next section compares some pre-existing approaches for evaluating performance based on system designs, using scenarios.

2.2. OTHER SCENARIO-BASED PERFORMANCE MODELS

The present research extracts performance models from scenarios for software behaviour, and this section describes other work that takes a similar approach.

2.2.1. Execution Graphs

Smith, as one of the originators and first practitioners of SPE, developed an approach to building QNs based on Execution Graphs (EGs) [112] [114] that are specially created for performance analysis. EGs are created by the analyst from knowledge of the system or its requirements and a systematic procedure is used to generate extended QN performance models.

In [134], Cortellessa and Mirandola described a methodology for deriving QN performance models from UML by using Execution Graphs as an intermediate model, which is then refined with hardware-specific parameters in order to generate extended

QNs that can be solved to obtain performance metrics. Their methodology uses UML Use Case Diagrams, Sequence Diagrams and Deployment Diagrams to generate EGs and extended QNs as follows:

- The Use Case Diagram is used to extract the user profile for the system which specifies which scenarios are executed and with what frequency (this corresponds to the workload characterization in CSM)
- The Sequence Diagrams are used the create meta-EGs which are EGs composed of interaction tuples that are instantiated later based on deployment
- The Deployment Diagrams are used to instantiate EGs from the meta-EGs and thus tailor them to the hardware platform that hosts the system. The Deployment Diagram is also used to assign values to the service centres of the extended QN.

After the EG instances and the extended QN have been created, Cortellessa and Mirandola assign probabilities to the edges of the EGs in order to derive the weight of each execution block and thus get the block's resource demands. These fully specified EGs are then used to add software-based parameters to the extended QN, at which point the extended QN can be solved.

The use of EGs as a means to create QN performance models from system designs predates the research presented in the thesis but does not model layered resource usages particularly well. As Cortellessa and Mirandola note in [134], the EG-based approach provides a valuable delay-based solution but does not capture synchronization effects in the system or the complexity in communication introduced by complex components.

2.2.2. KLAPER

The Kernel LAnguage for PErformance and Reliability analysis (KLAPER) was introduced by Grassi et al. in [43] to focus on the performance and reliability modeling of

systems developed using a Component Based Software Engineering (CBSE) approach. KLAPER was developed about the same time as the initial research on CSM presented in this thesis but was introduced after CSM. KLAPER was also defined as a Meta-Object Facility (MOF) metamodel [84] so that the transformations to/from KLAPER can make use of the Query/Views/Transformations (QVT) [85] model transformation language. Similarly to the use of CSM as an intermediate model, KLAPER also addresses the problem that having M multiple design models and N multiple target performance models requires M*N one-step transformations to cover all situations. By using KLAPER as an intermediate model, the necessary transformations can be broken down into a two-part process with M transformations from design models to KLAPER and N transformations from KLAPER to the target performance models. The overall complexity of the transformations is thus reduced and the number of required transformation methods goes down from N*M to N+M.

KLAPER is similar to CSM in that it models system behaviour and resources but both the behaviour and the resource sub-models are less rich than in CSM. However, in keeping with its CBSE focus, KLAPER provides a richer description of services. Services can be associated to both resources and behaviour, as well as to the usage of other services. In this way, KLAPER provides the capability to define intermediate performance models that retain a component framework that can be composed later.

KLAPER's component-based focus has the advantage that services can be modeled quite well and the related drawback that the system's behaviour is broken up between standalone behaviour specifications and component behaviour instead of having a unified behaviour model that captures the system's emergent behaviour.

2.2.3. Palladio Component Model

The Palladio Component Model (PCM) was developed by Becker, Koziolek and Reussner and introduced in [12] and [14]. Similarly to KLAPER, PCM is also focused on the description of component based software architectures and was introduced after CSM. The "Palladio-Bench" tool is available online at www.palladio-simulator.com and allows users to model PCM instances, simulate models, view simulation results, and derive software design optimisations.

The Palladio approach (and CBSE generally) identifies four developer roles producing design artifacts, as follows:

- *Component developers* who specify and implement the components. The specification contains an abstract, parameterized description of the component and its behaviour.
- *System architects* who assemble the components to build applications. They can retrieve component specifications by component developers from a repository.
- *System deployers* who model the resource environment and afterwards the allocation of components from the assembly model to different resources of the resource environment.
- *Business domain experts* who are familiar with the customers or users of the system and who provide usage models describing critical usage scenarios and parameter values.

Each developer role has its domain-specific modeling language and only sees and alters the parts of the model that are its responsibility. The complete system model is then derived from those partial models.

The PCM resource model has active resources, passive resources and linking resources. The PCM behaviour model is similar to CSM but, like KLAPER, it allows behaviour to

be assigned to services. The PCM component models can then be composed at a later stage.

Palladio shares KLAPER's advantage that it is well-suited for modeling component-based systems as well as the drawback that the behaviour model is broken up among the components. Ultimately both Palladio and KLAPER provide a less general intermediate model than CSM.

2.2.4. Tool Independent Performance Model (TIPM)

The Tool Independent Performance Model (TIPM) is an extension of CSM developed by Fritzsche [41] [42]. TIPM extends CSM with resource scheduling and with a performance specification and measurement sub-model that is similar to the CSM performance annotation sub-model explained in section 3.3.2.

The TIPM performance specification and measurement sub-model is composed of Experiments and Monitors. Experiments are used to indicate which kind of performance analysis should be performed and where. There are three kinds of Monitors that save the results from the performance analysis tool:

- A *LatencyMonitor* holds information about the latency between two Steps (defined through *entry* and *exit* associations).
- A *CountingMonitor* observes how often a Step is executed.
- An *UtilizationMonitor* observes the utilization of a resource.

In [41], Fritzsche *et al.* defined a *Tool2TIPM* mechanism to trace the performance results from a performance analysis tool (AnyLogic simulation tool) back to the TIPM intermediate model. They also define a *TIPM2DevelopmentModel* mechanism that uses

rules written as Higher-Order Transformations for the Atlas Transformation Language (ATL). Using the resource scheduling extension, they can simulate the TIPM directly. More recently in [104], Redlich *et al.* adapted and extended the workbench using TIPM developed by Fritzsche in order to create a workbench for Model-Driven Business Impact Analysis, which can model the impact of various disruptions on an organization's business processes.

2.3. DESIGN ANNOTATION FOR PERFORMANCE

2.3.1. UML Profile for Schedulability, Performance and Time (SPT)

The UML Profile for Schedulability, Performance and Time (SPT) [86] was developed to assist in the capture of performance data, and in the automation of model-building. This was intended to make performance analysis more accessible to developers who are concerned about performance issues in their designs. Fig 1 illustrates the type of processing that was envisaged by the SPT Profile.

The SPT Profile addresses a broad range of applications, from embedded systems with schedulability concerns (as described by Liu in [63]), to applications with statistical performance requirements, such as telecom, business systems and web services.

The SPT Profile extends UML with stereotypes and tags, which can be applied to object instances, and to instances of action executions in behaviour specifications. The UML specification together with the stereotypes determines the structural properties of a performance model, and the tags provide parameter values. The profile is based on domain sub-models for resources and for performance, which are also the basis of the CSM metamodel described below.

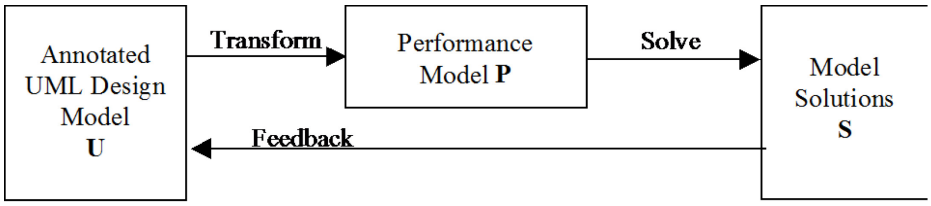


Fig 1 Transformations and performance model solutions from UML

The SPT Profile domain model for performance is summarized in Fig 2. It is centered on a Scenario class, representing behaviour for one kind of system response. A scenario is an ordered sequence of Steps, each of which can be further refined by a sub-scenario. The ordering supports forks, joins and loops in the flow. Stereotypes and tagged value names defined in the SPT Profile are prefixed by P for "Performance", PA for "Performance Analysis" or GRM for "General Resource Model".

Each scenario has a *workload*, which describes the intensity of its use. It may be an *open workload*, with arrivals from the environment (described by their rate), or a *closed workload* in which a fixed number of potential arrivals are either executing the scenario, or are waiting to arrive again.

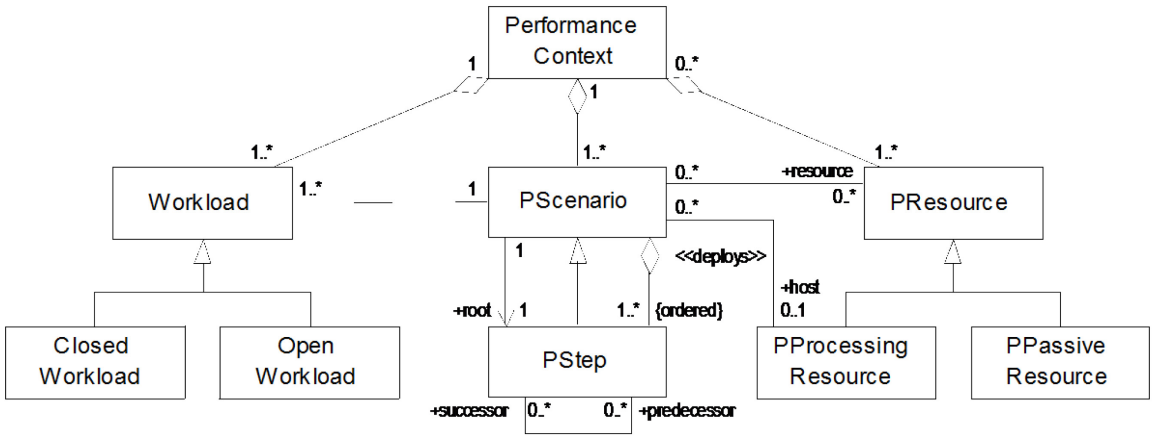


Fig 2 Performance domain model of the SPT Profile (from Figure 8-1 of [86])

A *simple Step* (without refinement) is the basic unit of work. It is executed by the processor identified as its *host* processing resource, with a specified demand parameter for its CPU time. A *complex Step* defined by a sub-scenario refinement can also be used. Resources may be attached to a scenario. The domain sub-model for resources in chapter 4 of [86] distinguishes between *active* resources (such as a user), which spontaneously generate events, and *passive* resources that respond to requests. Both types of resource may be *protected* (in which case a client gets exclusive use of one or more units of the resource), or *unprotected* (in which case they can be shared without control). For performance analysis, only the protected resources are significant. Chapter 8 of [86] distinguishes between ProcessingResources, representing devices, and PassiveResources, representing logical resources (created by the software, such as buffers, tasks, or semaphores).

The SPT Profile was defined for UML 1.4, but with only a little additional interpretation the Performance Analysis part of it was applied to UML2 behaviour models in chapter 8 of [86]. The features of UML2 that affect the use of the Profile and their interpretations are as follows:

- behaviour fragments and combined fragments in Interaction Diagrams, for loop, alt (alternative paths), par (parallel paths), and ref (reference to another diagram, expressing a sub-scenario). Annotations that are needed on the first Step of a sub-path (for alt and loop fragments) can be attached to the first <<PAstep>> stereotype in the UML 2 fragment or as a note on the fragment itself.
- the <<PAstep>> stereotype can be applied to an execution-occurrence in an Interaction Diagram, as well as to the other entities specified in the Profile for UML 1.4.

- deployment is specified in terms of *artifacts* rather than components; a one-to-one correspondence between components and some of the artifacts is assumed, so that the deployment of components can be specified equally well using the artifacts.

Thus, there are no conceptual barriers to the use of the SPT Profile with UML 2 [94].

2.3.2. UML Profile for Modeling and Analysis of Real-Time and Embedded Systems (MARTE)

The UML Profile for Modeling and Analysis of Real-Time and Embedded Systems (MARTE) has replaced the SPT Profile as of November 2009. MARTE v1.0 was ratified and released by OMG in November 2009 and the current MARTE v1.1 was released in June 2011.

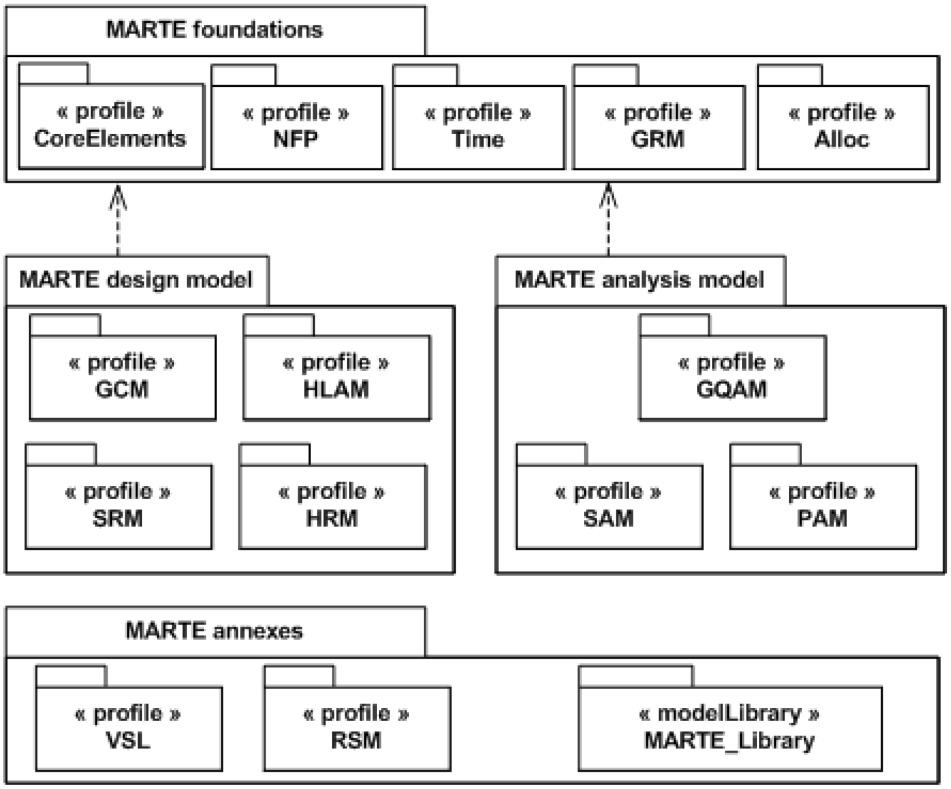


Fig 3 MARTE profile architecture

MARTE extends the performance concepts from SPT and covers an even broader range of applications than SPT; from embedded systems with schedulability concerns, to business systems. The development of MARTE was informed by performance annotation and modeling concerns with SPT that came to light during the development of CSM, particularly with respect to the treatment of messages and related annotations.

The MARTE profile architecture, shown in Fig 3, is composed of four packages that group the shared MARTE foundations, the MARTE design model for the features of real-time and embedded system, the MARTE analysis model to annotate application models so as to support analysis of system properties, and the MARTE annexes and model library.

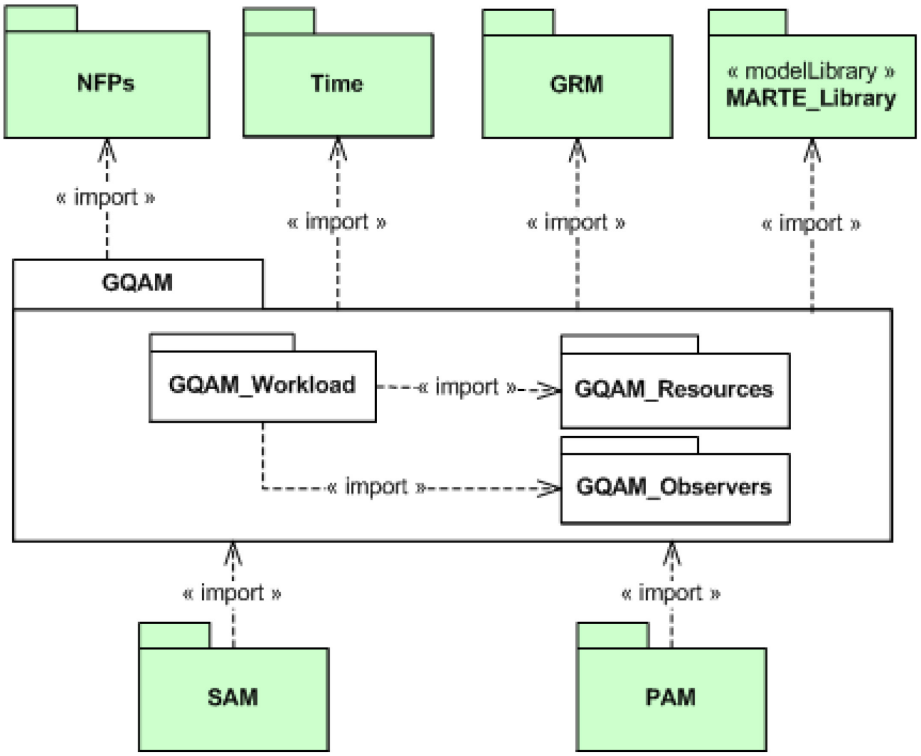


Fig 4 Dependencies of the GQAM profile

Given the breadth of the MARTE specification, the research presented in this thesis focuses on MARTE analysis model; specifically the Performance Analysis Modeling

(PAM) sub-profile and the Generic Quantitative Analysis Modeling (GQAM) sub-profile it depends on.

The GQAM profile is used to annotate models subject to quantitative analysis. Fig 4 shows the dependencies of the GQAM domain model. The GQAM elements are shown in white while the elements that GQAM depends on, or which depend on GQAM, are in green.

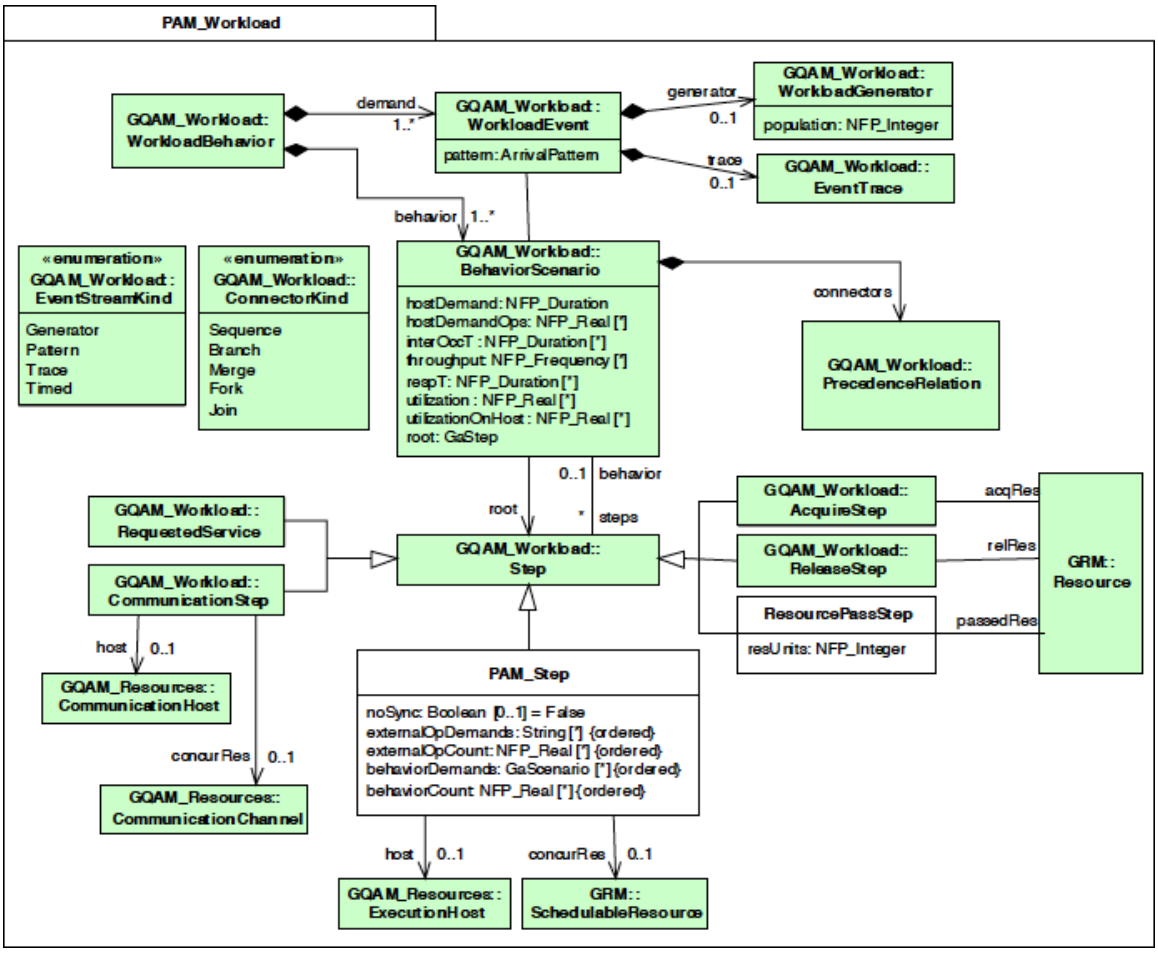


Fig 5 PAM behaviour model

The PAM behaviour domain model shown in Fig 5 extends the GQAM behaviour model with the addition of PAM_Step and ResourcePassStep. Behaviour in PAM, and by extension MARTE, is modeled in much the same way as in SPT. Behaviour is modeled

through scenarios which are ordered sequence of PAM_Steps, each of which can be further refined by a child sub-scenario. The scenario ordering supports parallel forks and joins, alternate branches and merges, and loops in the flow. Scenarios are driven by WorkloadEvents generated by WorkloadGenerators. Special step types include AcquireStep and ReleaseStep for resource handling similar to SPT, as well as additional ResourcePassStep for resource handling and CommunicationStep for explicit messaging communication.

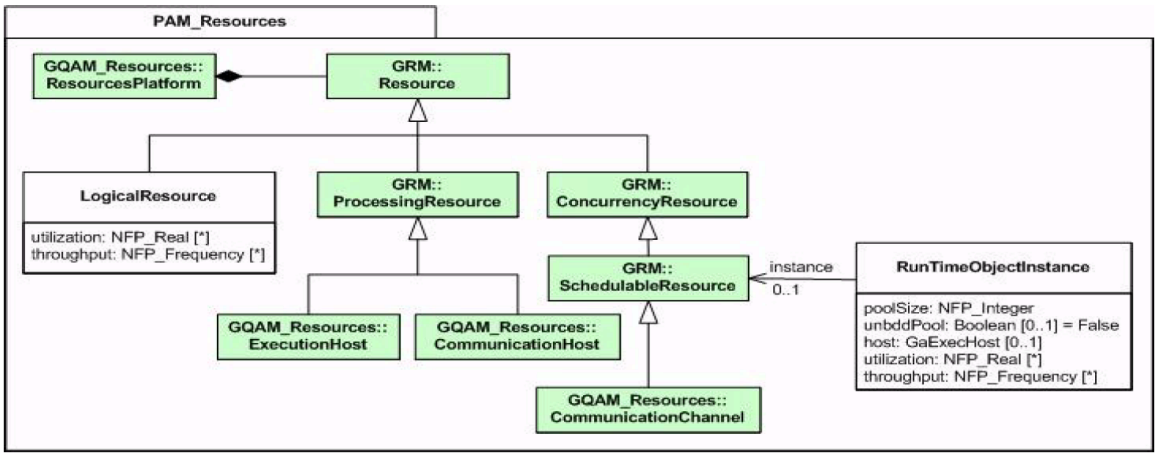


Fig 6 PAM resource model

The PAM resource domain model shown in Fig 6 extends the GQAM resource domain model with the addition of LogicalResource and RunTimeObjectInstance. Compared to the SPT resource model, PAM and GQAM also add CommunicationChannel and CommunicationHost to support the modeling of messaging.

In practice, the MARTE profile uses slightly different naming from the domain model. Table 1 lists the details of the MARTE stereotypes and attributes in the profile plug-in used for the generation of CSM from UML design models.

Table 1. MARTE stereotypes and attributes used for the generation of CSM

MARTE Stereotype	Attribute	Value
GaWorkloadEvent	pattern:ArrivalPattern	closed(population, extDelay:NFP_Duration)
		open(interArrivalTime:NFP_Duration, arrivalRate:NFP_Frequency, arrivalProcess:string)
		periodic(period)
Resource	resMult	Int
SchedulableResource	resMult	Int
PaLogicalResource	poolSize	Int
PaRunTInstance	poolSize	Int
	instance	«SchedulableResource»
	host	«GaExechost»
GaExecHost	resMult	Int
	speedFactor	Real
	commTxOvh	(real, units):NFP_Duration
	commRcvOvh	(real, units):NFP_Duration
GaCommHost	resMult	Int
	speedFactor	Real
	capacity	(datarate, units):NFP_DataTxRate
	blockT	(real, units):NFP_Duration
PaStep	hostDemand	(real, units):NFP_Duration
	blockT	(real, units):NFP_Duration
	rep	Real
	prob	Real
	servDemand	«GaRequestedService»
	servCount	Real
	behavior	«GaScenario»
	noSync	TRUE | FALSE
	behavDemand	«GaScenario»
	behavCount	Real
	extOpDemand	string name of operation
	extOpCount	Real
	respT	((value, units),req):NFP_Duration
GaCommStep	hostDemand	(real, units):NFP_Duration
	blockT	(real, units):NFP_Duration
	rep	Real
	prob	Real
	servDemand	«GaRequestedService»
	servCount	Real
	nosync	TRUE | FALSE
	behavDemand	«GaScenario»

MARTE Stereotype	Attribute	Value
	behavcount	real
	extOpDemand	string name of operation
	extOpCount	real
	msgSize	(real, units):NFP_Datasize
GaAcqStep	resource	«Resource»
	resUnits	Integer
GaRelStep	resource	«Resource»
	resUnits	Integer
PaResPassStep	resource	«Resource»
	resUnits	Integer
GaScenario	root	«PaStep»

2.4. LAYERED QUEUEING NETWORKS (LQN)

Layered Queueing Networks (LQN) model contention for both software and hardware resources, based on requests for services. Clients make service requests and queue at the server. In ordinary queueing networks there is one layer of servers; in LQN, servers may make requests to other servers, with any number of layers [105]. An LQN can thus model the performance impact of the software structure and interactions, and be used to detect software bottlenecks as well as hardware performance bottlenecks [81].

In an LQN the software resources are called tasks, (representing a software process with its own thread of execution, or some other resource such as a buffer) and the hardware resources are called processors or devices (typical devices are CPUs and disks). Tasks can have different priority levels on their CPU. The workload of an LQN is driven by open arrival streams of external requests, or by closed workloads modeled by reference tasks with set multiplicities which cycle and make requests.

An LQN can be represented by a graph with nodes for entries, tasks and devices, and arcs for service requests (labeled by the mean number of messages sent). There are two types

of arc to represent asynchronous messages (with no reply) and synchronous messages which block the sender until there is a reply. Tasks receive either type of message at designated interface points called entries. A Task has a different entry for every kind of service it provides; an entry also represents a class of service. A special type of Task called a Semaphore Task is used to model access control to logical resources. A Semaphore Task has only to paired Entries; an Entry for acquisition/locking which must be called synchronously and a paired Entry for release/unlocking which must be called asynchronously.

Internally an Entry has service demands defined by sequences of smaller computational blocks called Activities, which are related in sequence, loop, parallel (AND fork/joins) and alternative (OR fork/joins) configurations. Activities have processor service demands and generate calls to entries in other tasks.

A third type of interaction called forwarding is a combination of synchronous and asynchronous behaviour. The sending client task makes a synchronous call and blocks until it receives a reply. The receiving task partially processes the call and then forwards it to another server, which then becomes responsible for sending a reply to the blocked client task; it can be forwarded with a probability, and any number of times. The intermediate server task can begin a new operation after forwarding the call.

LQNs are saved in either a string-based text format [147] or in an XML format [67]. Models are solved either by simulation using LQSim or by analytic approximations using LQNS. LQNS is based on Rendezvous Networks [129] and the Method of Layers [105], with a number of additional approximations [37][39][40]. The approximations have

limitations in dealing with priorities (poor accuracy) and with AND-joins that do not have an AND-fork in the same task, so simulation is often useful.

Fig 7 shows the LQN visual notation used in this thesis. The LQN entities are:

- task: parallelogram
- entry: small shaded parallelogram inside a task and near the task's top boundary
- activity: small square inside a task
- reply activity: small shaded square inside a task
- asynchronous call: an open arrowhead, shown here with one side only
- synchronous call: a solid arrowhead
- forwarding call: a dashed line with a solid arrowhead

Fig 7 also shows the LQN notations for AND and OR forks and joins between activities.

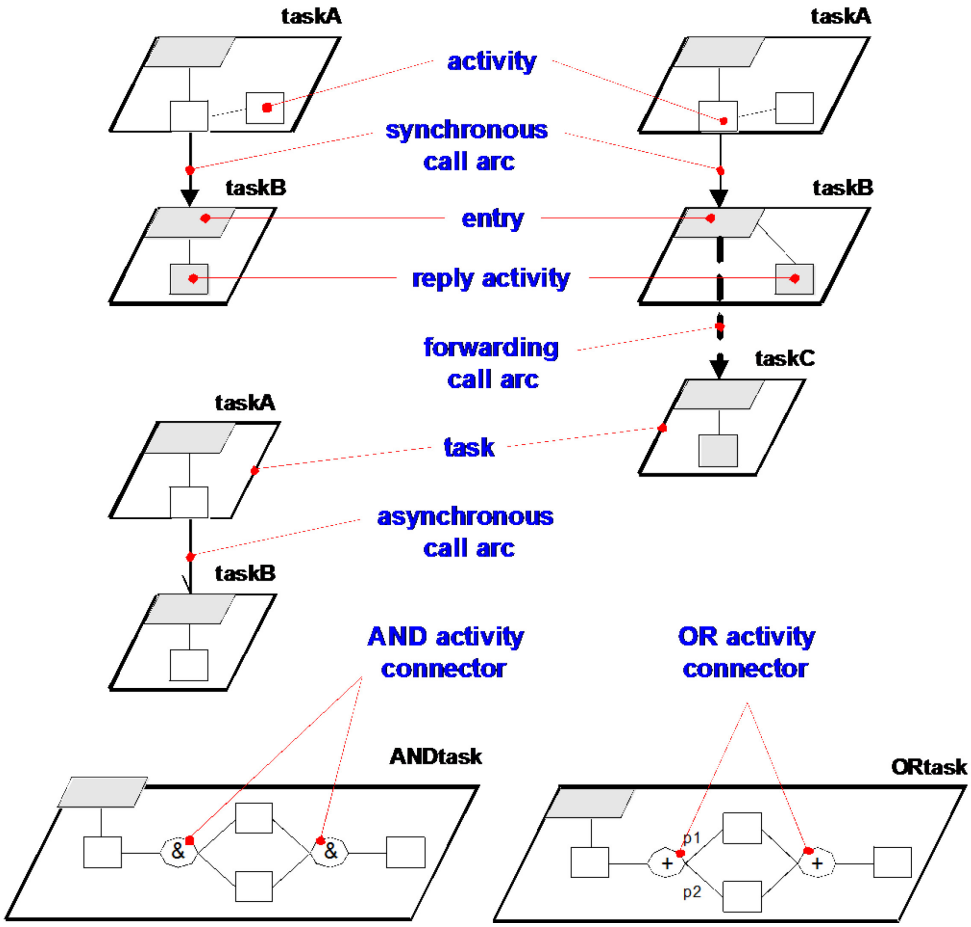


Fig 7 LQN visual notation

An alternate LQN visual notation only has tasks and entries and does not show activity detail. With that notation, a task is represented as a rectangle with entries being listed to the left of the task name. The call arcs remain the same as above.

To tools used in this research are ModelMerge and Simplify [48], described briefly here for completeness.

LQN Model Merge

Sometimes separate models are created for different uses of the same system. Typically they have mostly the same tasks, but include different entries. They can be combined into a single LQN model by a tool called ModelMerge. The algorithm for ModelMerge is based on name matching. Tasks with the same name are merged into a single task and all the different entries are added to the merged task. Some of these entries may be duplicates; they are candidates for simplification using LQN Simplify.

LQN Simplify

LQN Simplify identifies and eliminates duplicate entries on a task-by-task basis. Simplify makes multiple passes through the LQN until no additional duplicate entries can be found at any layer.

LQN Simplify identifies duplicate entries based on their service demands and the calls they make. At the bottommost layer entries only have service demands and the simplification begins there. Entries within a given task with the same service demands, or with similar enough service demands (within a given band), are identified as duplicates and replaced by a single entry with the identified service demands; the call arcs from higher layers are routed to the single entry. Once all the duplicate entries in the bottom layer have been processed, LQN Simplify moves on to the next higher layer. At higher

layers entries must have the same service demands, or the same approximate service demands, and must make the same number of calls, or approximate number of calls within a given tolerance band, to the same entries in lower layers in order to be identified as duplicates. Duplicate entries are then replaced by a single entry with those demands and calls, and the call arcs from higher layers are then re-routed to the single entry. Simplify processes the LQN layers from the bottom up. Once all the layers have been processed, then the processing restarts with a new pass. Simplify stops when it has made a complete pass without simplifying anything at any layer.

The basic framework for both ModelMerge and Simplify was developed by Israr in [48].

3. CORE SCENARIO MODEL (CSM)

The Core Scenario Model (CSM) is the foundation of the PUMA architecture, and was developed by the author with input from the larger PUMA project team. The purpose of CSM is to express all of the available design information relevant to performance evaluation in a single model in order to facilitate the automation of the SPE process and alleviate the transformation complexity problem described in section 2.1. By providing an intermediate model between the software model and the performance model, it unifies the creation of performance models (any software model, any performance model) and overcomes the M×N problem raised by multiple types of models in both domains [137]. CSM is a *projection* from the UML behaviour into the performance domain. CSM filters out a mass of design information irrelevant to performance, adapts to different UML diagram types and different tools, and makes it easy to verify the performance information, as well as to supply default interpretations and values if necessary. For other software specification formalisms it has a similar role.

Used with UML, CSM captures the essence of performance specification and estimation as expressed in the MARTE Profile, and strips away the design detail that is irrelevant to that analysis. It is suited to the production of performance models of several kinds, as demonstrated in this thesis by the generation of LQNs and QNs. It is well-suited to expressing the performance-relevant information from different UML sub-models as it is derived directly from the General Resource Model (GRM) and Performance Analysis Modeling (PAM) used in MARTE. CSM can also be used with non-UML software specifications such as Use Case Maps [3] [79] [132]. CSM was proposed as a standard

for intermediate performance-friendly models [93] [94]. Other related scenario models with the same purpose include Execution Graphs and KLAPER [39].

3.1. PERFORMANCE THROUGH UNIFIED MODEL ANALYSIS (PUMA)

This research is part of the Performance through Unified Model Analysis (PUMA) project [125]. PUMA attempts to unify the use of arbitrary combinations of design diagrams and of performance models in a general tool architecture that is applicable to other extra-functional analyses based on scenarios. The PUMA architecture is shown in Fig 8 below.

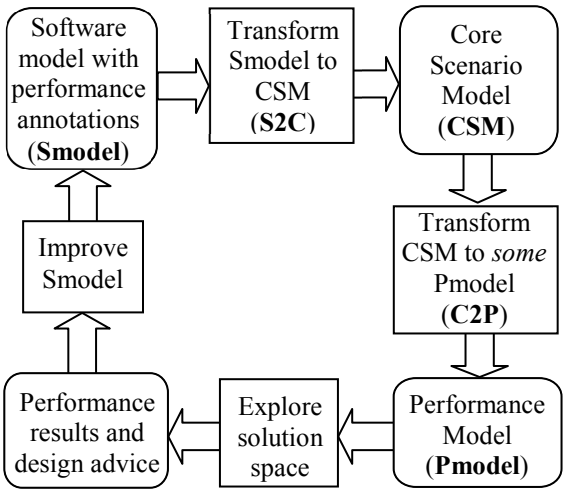


Fig 8 The PUMA architecture

The UML to performance model translations mentioned in section 2.1 are specific to one kind of UML behaviour diagram and one kind of performance model. This approach results in an $M \times N$ problem when it comes to the definition of transformations from an M UML diagram types to N performance models since each combination of UML diagram to performance model pairing requires a separate transformation. $M \times N$ problems are best addressed by a common intermediate format, such as the Core Scenario Model (CSM)

described in chapter 3. The use of CSM as an intermediate model turns the $M \times N$ problem for defining transformations into an $M+N$ problem where specific transformations need only be defined from the different UML diagrams to CSM and then from CSM to the particular performance model. In addition to reducing the overall number of transformations needed, the PUMA architecture also reduces their complexity since CSM is conceptually between the UML and performance modeling paradigms.

The strength of the PUMA approach is evident when the addition of a new performance formalism is considered, as it is much simpler to translate from CSM to a performance model, than from a UML tool. The information in CSM is filtered and verified; only the performance-related definitions are included.

Fig 9 illustrates the chain of model transformations envisaged in PUMA. The relevant information in the UML design *U* can be scattered in different behaviour and deployment sub-models, and possibly in other sub-models as well. Some of it is expressed in the stereotypes and tagged values of the SPT Profile, and some (e.g. the sequence of actions) is implicit in the UML. The CSM intermediate model collects and organizes all this into a form that is convenient for generating the performance model *P*, and allows us to check for consistency and completeness of this information from the viewpoint of *P*. The CSM metamodel supports a two-step process as shown in Fig 9, in which the *U2C* transformation extracts the scenario model and the *C2P* transformation derives a performance model. Additional *C2P* transformations may support different performance formalisms for *P*, thus mitigating the $M \times N$ complexity problem explained above. CSM is intended to support all kinds of performance models.

CSM also provides traceability links back to the UML in order to facilitate the *feedback* of the performance results into the context of the design model *U*.

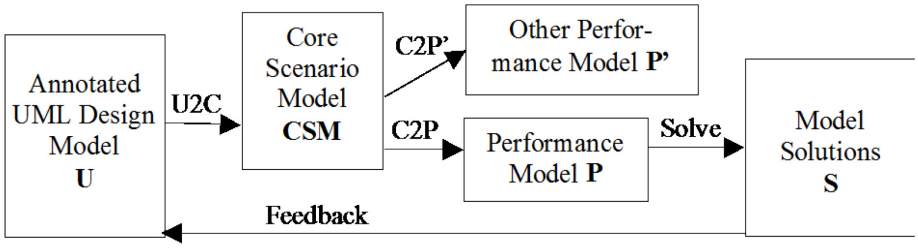


Fig 9 Two-step transformation supporting consistency-checking and a variety of performance formalisms

3.2. PURPOSE OF CSM

The purpose of CSM is to express all of the available design information relevant to performance evaluation in a single model in order to facilitate the automation of the SPE process and alleviate the transformation complexity problem described in section 2.1. CSM provides the basis for the performance model generation algorithms that will be described in chapter 6 and it also provides a platform for the manipulation of software designs and architectures in order to evaluate the performance impact of different alternatives. The CSM to CSM transformations described in chapter 5 include algorithms to clean up CSMs with minor syntactic flaws, algorithms to simplify CSMs in order to facilitate the generation of performance models, and algorithms to discover calling interactions in CSM scenarios.

The change of perspective from a functional specification model in UML to a performance model is profound and requires a re-orientation of the model information. Resources, which are peripheral in functional specifications, are central for performance. Performance is determined by the way in which operations use resources (which resources, for how long, and in what order). The CSM metamodel expresses resource-

centric models, which may be derived from UML via MARTE, in a precise way that supports the generation of different kinds of performance models.

The CSM metamodel captures the essential entities in the MARTE performance modelling domain which are required for building performance models, and it makes explicit some facts which have to be inferred from UML and MARTE data.

3.3. CSM DEFINITION

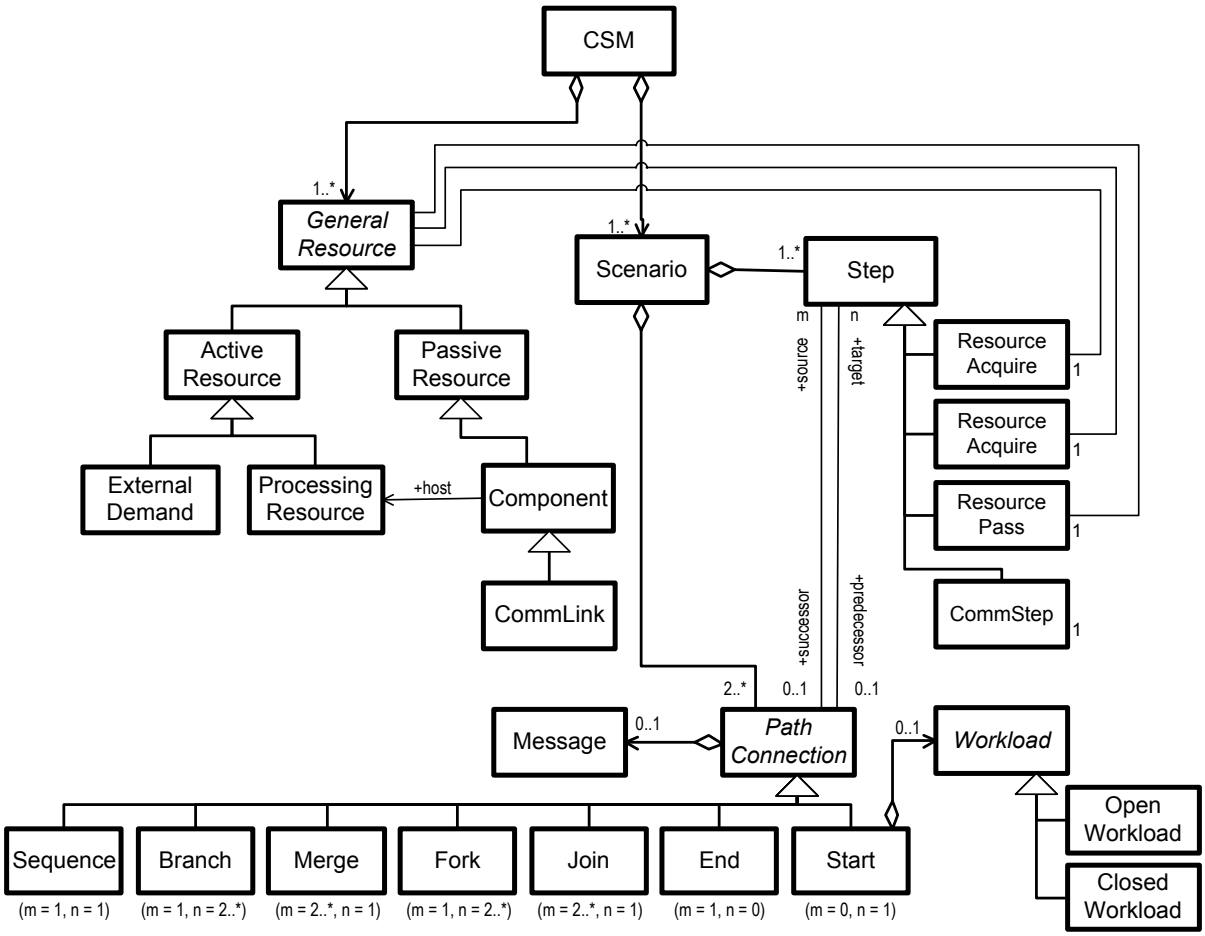


Fig 10 Classes in the CSM metamodel introduced in [93] [94]

The CSM metamodel introduced in [93] [94], is shown in Fig 10. CSM was originally designed for compatibility with the performance annotations defined in the SPT profile. After the MARTE profile replaced SPT, the CSM definition was revised in order to ensure continued compatibility with UML2 and MARTE.

Thanks to the flexibility of the original CSM specification, the revisions required to ensure compatibility with MARTE were minor. The changes from CSM v1 [89] to CSM v2 [90] are summarized as follows:

- *CommStep* (subtype of *Step*) replaced *Message* (a standalone type, associated with *PathConnection* in CSM v1)
    - *CommStep* reuses the message kind and message size from *Message*
- *CommLink* (standalone type) was added
- *ResourcePass* (subtype of *Step*) was added
- *ExternalDemand* (standalone type) was added to replace *ExternalOperation*
- *PassiveResource* was renamed to *LogicalResource*

A class diagram of the XML schema implementation of CSM v2 is shown in Fig 12 with an expanded version provided in the appendix.

Table 2 lists the attribute, association and containment details for CSM v2. The rows for new classes or changed classes compared to CSM v1 are shaded.

ID attributes are generated automatically in order to uniquely identify CSM objects. All objects in a CSM must have an ID attribute. Associations are implemented as attributes of IDRef type. IDRefs are constrained to only refer to IDs that are defined in the model. The optional traceability link attribute is intended to maintain traceability to the corresponding objects in the original design model.

Table 2. Details of attributes and associations of the CSM v2 metaclasses. Shaded rows show classes that are different from CSM v1. Attributes, associations, and contained elements in bold are required and have validity constraints.

Metaclass	Attributes/Associations	Containment	Constraints
CSM	name (optional) description (opt.) author (opt.) created (opt.) version (opt.) traceabilityLink (opt.)	**scenarioList** componentList logicalResourceList processingResourceList commLinkList perfMeasureList variableList	**scenarioList** contains at least one Scenario
CSMElement (abstract class)	**id** name (opt.) description (opt.) traceabilityLink (opt.)		**id** is unique
Scenario	*same as CSMElement +* probability (opt.) parentRefinementList (opt.)	**stepList** **startList** **endList** commStepList resourceAcquireList resourceReleaseList resourcePassList sequenceList branchList mergeList forkList joinList	**stepList** contains at least one Step **startList** contains a single Start **endList** contains at least one End
ScenarioElement (abstract class)	*same as CSMElement +* durationPerfMeasureList (opt.) triggerPerfMeasureList (opt.) stopPerfMeasureList (opt.)		
Step	*same as ScenarioElement +* **predecessor** **successor** hostDemand (opt.) component (opt.) probability (opt.) repetitionCount (opt.) blockT (opt.) respT (opt.) nosync (opt.)	externalDemand refinement	**predecessor** contains a single *IDRef* to a *PathConnection* **successor** contains a single *IDRef* to a *PathConnection* hostDemand, externalDemand and refinement cannot be defined concurrently

Metaclass	Attributes/Associations	Containment	Constraints
CommStep	*same as Step +* msgKind (opt.) msgSize (opt.) txComp (opt.) rcvComp (opt.) commLink (opt.) predCommStep (opt.) succCommStep (opt.)	*same as Step*	*same as Step*
ResourceAcquire	*same as Step +* **acquire** rUnits (opt.) priority (opt.)	*same as Step*	*same as Step +* **acquire** contains a single *IDRef* to a *GeneralResource*
ResourceRelease	*same as Step +* **release** rUnits (opt.)	*same as Step*	*same as Step +* **release** contains a single *IDRef* to a *GeneralResource*
ResourcePass	*same as Step +* **pass** rUnits (opt.)	*same as Step*	*same as Step +* **pass** contains a single *IDRef* to a *GeneralResource*
Refinement	**parentStep** **subScenario**		**parentStep** contains a single *IDRef* for a Step **subScenario** contains a single *IDRef* for a Scenario
PathConnection (abstract class)	*same as ScenarioElement +* **source** **target**		
Sequence	*same as PathConnection*	*same as PathConnection*	**source** contains a single *IDRef* for a Step **target** contains a single *IDRef* for a Step
Branch	*same as PathConnection*	*same as PathConnection*	**source** contains a single *IDRef* for a Step **target** contains at least two *IDRefs* for Steps
Merge	*same as PathConnection*	*same as PathConnection*	**source** contains at least two *IDRefs* for Steps **target** contains a single *IDRef* for a Step
Fork	*same as PathConnection*	*same as PathConnection*	**source** contains a single *IDRef* for a Step **target** contains at least two *IDRefs* for Steps
Join	*same as PathConnection*	*same as PathConnection*	**source** contains at least two *IDRefs* for Steps **target** contains a single *IDRef* for a Step

Metaclass	Attributes/Associations	Containment	Constraints
Start	*same as PathConnection*	*same as PathConnection* + closedWorkload openWorkload	**source** is undefined or empty **target** contains a single *IDRef* for a Step closedWorkload and openWorkload cannot be defined concurrently
End	*same as PathConnection*	*same as PathConnection*	**source** contains a single *IDRef* for a Step **target** is undefined or empty
Workload (abstract class)	*same as ScenarioElement* + arrivalProcess arrivalParam1 (opt.) arrivalParam2 (opt.) externalDelay (opt.) responseTimeList (opt.)		
OpenWorkload	*same as Workload*		
ClosedWorkload	*same as Workload* + **population**		**population** contains a value
GeneralResource (abstract class)	*same as CSMElement* + multiplicity (opt.) schedPolicy (opt.) perfMeasureList (opt.)		
ActiveResource (abstract class)	*same as GeneralResource,* + opTime (opt.)		
ProcessingResource	*same as ActiveResource* + speedFactor (opt.) commRcvOvh (opt.) commTxOvh (opt.) commLinkList (opt.)		
LogicalResource	*same as GeneralResource* + nestedOnly (opt.)		
Component	*same as LogicalResource* + host isActiveProcess (opt.) parentComponent (opt.) subComponentList (opt.)		
CommLink	*same as Component* + blockTime (opt.) capacity (opt.) resMult (opt.) processingHostList (opt.) speedFactor (opt.)		

3.3.1. CSM Notation and Constraints

A shorthand notation for CSM is described as follows:

```
CSM = ( {E}, {A} )
```

```
      E = CSM Entity                {E} = set of CSM Entities
      A = CSM Association           {A} = set of CSM Associations
      A = E x E
```

A CSM is a set of CSM entities and associations between those entities. At the top level, the CSM entities are Scenarios, Steps, or Resources:

```
E = Sc | St | R
      Sc = Scenario
      St = Step
      R = Resource
```

The term "CSM element" is applied to both CSM entities and associations. A CSM element must be one of ScenarioType (Sc), StepType (St), ResourceType (R), or AssociationType (A).

The associations between CSM entities comprise both containment and logical associations. The specific associations are:

- Scenario contains Steps
- Step has a logical *refinement* association with a sub-Scenario (optional)
- Step has a logical *component* association with a Resource (optional)
- Resources have containment associations with other Resources (optional)
- Resources have logical *host* associations with other Resources (optional)

The Steps in a Scenario are partitioned among the Scenarios so every Step belongs to exactly one Scenario. The specific definition of a Scenario is as follows:

```
Sc = ( S, {Pc}, {Pa}, W )
      S = {St}subset
      Pc = Path Connection    {Pc} = set of Path Connections
      Pa = Path Association   {Pa} = set of Path Associations
      W = Workload
```

<u>PathConnections</u> – PathConnections represent the connections between the Steps in a Scenario. Path Connections have the following types:

```
Pc = Start | End | Seq | Fork | Join | Branch | Merge
```

The particular associations between Steps and PathConnections are given by the following *path associations*:

```
Pa = (St x Pc) ∪ (Pc x St)
```

Path associations are specified from a Step to a *predecessor* <u>and</u> *successor* PathConnection and from a PathConnection to a *source* <u>and/or</u> *target* Step. Path associations have the following restrictions:

- Start must have one association to a target Step
  - there can be no association from a source Step to a Start
- End must have one association from a source Step
  - there can be no association going from an End to a target Step
- Seq must have one association from a source Step and one association to a different target Step
- Fork must have one association from a source Step and two or more associations to different target Steps
- Join must have two or more associations from different source Steps and one association to a different target Step
- Branch must have one association from a source Step and two or more associations to different target Steps
- Merge must have two or more associations from different source Steps and one association to a different target Step

<u>Workloads</u> – Scenarios have optional workload characteristics. A Workload can be either a ClosedWorkload with a constant user population or an OpenWorkload with a given user arrival interval. Workloads are defined below:

```
W = Wc | Wo | null
      Wc = Closed Workload
      Wo = Open Workload
```

The interpretation of workloads is that they are attached to start points and describe the triggering of scenarios. Scenarios without workloads can only be executed as sub-scenarios. The interpretation of a closed workload is that a workload generator provides a token that triggers the execution of the scenario. The token is cycled back to the workload generator after the scenario ends and is reused/reissued after some delay. Multiple tokens can trigger multiple concurrent executions of the same scenario, however the number of concurrent executions can never exceed the overall token population. An open workload has a workload generator that creates tokens at a given rate. Each token only triggers a single execution of the scenario and it does not cycle back to the generator after the scenario completes. Since the tokens do not cycle through the system and there is no set token population there is no limit on the number of tokens executing the scenario concurrently.

Step Types – There are different kinds of Steps as follows:

```
St = Se | SeR | Sc | Sra | Srr | Srp
      Se = Execution Step
      SeR = Execution Step with Refinement
      Sc = Communication Step
      Sra = Resource Acquire Step
      Srr = Resource Release Step
      Srp = Resource Pass Step
```

Execution Steps execute operations and make service demands. Execution Steps are standalone steps that specify their own operation and service demand.

Execution Steps with Refinement use sub-scenarios to specify operations and service demands with more granularity. End points in a sub-scenario refinement are assumed to connect back to the successor of the parent SeR, unless they are specifically tagged as being *noSync* . End points with *noSync* terminate inside the sub-scenario.

Communication Steps add messaging-related functionality to execution steps.

Resource Acquire Steps denote the acquisition of resources along the Scenario and specify the number of resource units that are being acquired if the given resource has a multiplicity greater than 1.

Similarly, Resource Release Steps denote the release of resource along the Scenario.

Resource Pass Steps specify which resources are passed exclusively to a specific parallel path after a Fork. Resource Pass Steps must be used directly following a Fork. The default interpretation of resource holding after a Fork is that resources held before a Fork are shared across all the parallel branches following the Fork. If a Resource Pass Step is used right after a Fork then the interpretation is that the following scenario path is granted exclusive access to the resource units specified in the Resource Pass Step.

Resources – The Resource types are defined as:

```
R = Rp | Rc | Rpassive | Re
      Rp = Processing Resource
      Rc = Component Resource
      Rl = Logical Resource
      Re = External Resource
```

- Processing Resources represent processors for the execution nodes of the system.
- Component Resources are the manifestations of artifacts deployed on the nodes.
  - Components most commonly represent deployed instances of software modules that are not re-entrant, have access control, and have limited concurrency (such as tasks and/or processes), but they can also represent re-entrant modules without access control or with unlimited concurrency (such as objects).
- Logical Resources represent resources with no access control that are not specified as Components.
- External Resources represent resources otherwise defined outside of the CSM.

3.3.2. Performance Annotations

CSM provides a performance sub-model that allows for the specification of performance requirements, assumptions, predictions and measurements. This performance annotation sub-model can be combined with the traceabilityLink attribute present in all CSM objects to record performance needs and results.

The sub-model for performance related annotations is shown in Fig 11. *PerfMeasure* can be one of the five types of measures listed in *PerfAttribute*. *PerfMeasure* can have multiple values defined by *PerfValue* which can be used for expected results as well as calculated results. In order to report results, a new *PerfValue* for each result being reported can be added to a given *PerfMeasure*. The *kind* attribute of PerfMeasure defines the type of the value and the *source* attribute defines where the value comes from. When reporting performance results the *source* would be " measured".

The usage and interpretation of a *PerfMeasure* is defined by its associations. All *PerfMeasures* are contained in the CSM and can be associated with any of a *Scenario*, a *Workload*, one or more *Steps* (through separate associations), or a *GeneralResource*. The proposed convention is that *PerfMeasures* associated with a *Scenario*, a *Workload*, or a resource should not be associated with any other elements. A *PerfMeasure* associated with a *Scenario* or a *Workload* can be a *delay* for denoting an end-to-end response time or a *throughput*. A *PerfMeasure* associated with a resource can be a *throughput* or *utilization*.

PerfMeasures associated with *Steps* are either associated to a single *Step* using the +*step* association or to a pair of *Steps* using the +*trigger* and +*end* associations. The +*trigger* and +*end* association pair is used to denote duration and should start before the execution of the +*trigger Step* and finish after the execution of the +*end Step*. It is possible to

associate the +*trigger* and +*end* of a *PerfMeasure* to the same *Step*, in which case the duration is that of the single *Step*.

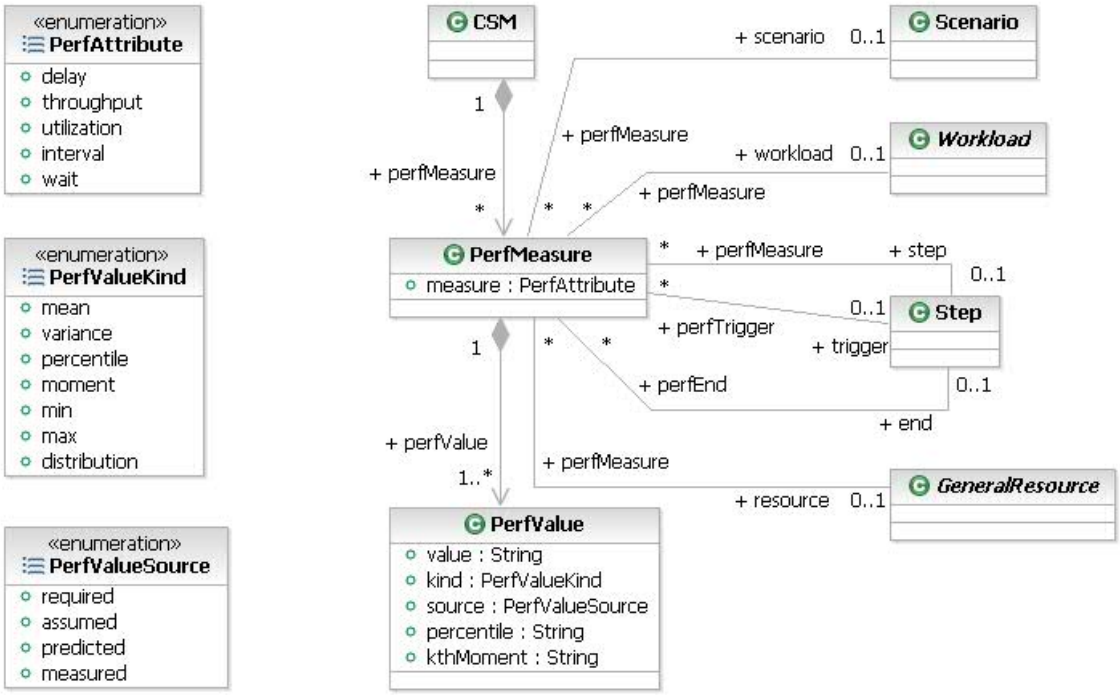


Fig 11 CSM class diagram for performance-related annotations

The CSM performance sub-model can be used in the PUMA architecture in order to provide feedback of performance analysis results in the original design model. The user defines the sought performance measures in the design model. These performance measures can be either *required* to indicate a performance requirement, *assumed* to indicate a performance assumption that the developer is making, *predicted* to indicate that they were obtained from a previous performance analysis, or *measured* to indicate that they were obtained from past measurements. The performance measures can be carried to the corresponding CSM, along with traceability links to identify the specific elements in the design model with which they are associated. Additional traceability links from the CSM to the generated performance model are required to feed results from the

performance model back into the CSM and then back into the design model. This in principle completes the feedback loop presented in the PUMA tool architecture shown in Fig 8. Results feedback is however not implemented in this thesis research.

3.3.3. Parameters in CSM

All attributes of CSM elements are defined as strings, meaning that their values can be given as variables. Therefore all CSM attributes and associations can be parameterized. It is up to the user to define her preferred convention for defining parameters and the rules for assigning values to them.

3.3.4. XML Schema for CSM

For use with tools, the CSM metamodel is defined as an XML schema. The schema specifies the CSM metaclasses and attributes in a widely accepted format that can be read by many programs, including an integrated development environment (IDE) such as Eclipse for Java development.

The attributes of the CSM entities correspond to tagged values in MARTE and are described in Table 2. The ID attributes are used in an XML schema for CSM [147] to uniquely identify CSM elements, particularly if they are not named. These IDs are also referenced by other elements in the CSM XML documents in order to capture the association relationships listed in Table 2. The class diagram for the CSM XML schema generated by Eclipse is shown in Fig 12 below. A larger version of the class diagram is also included in the appendix for improved readability.

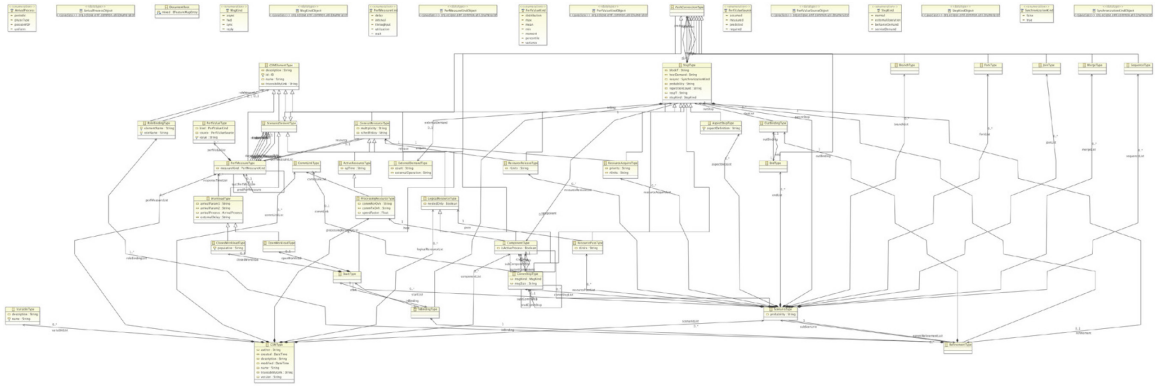


Fig 12 Class diagram for the CSM v2 XML schema [90].

3.4. CSM BEHAVIOUR MODEL

CSM scenarios denote behaviour in the context of the use of resources, which is key for performance analysis. CSM thus models the emergent behaviour of the system and that behaviour's associated resource context.

Sections 3.4.1 and 3.4.2 describe how behaviour interacts with resources in CSM in order to discover the system's emergent resource architecture. The requirements for well-formed CSM scenarios are also presented in section 3.4.3.

3.4.1. Emergent Resource Architecture

CSM explicitly identifies the use of resources with *ResourceAcquire*, *ResourceRelease*, and *ResourcePass* steps. These explicit indications ensure correct calculation of resource holding times; they have attributes for the resource ID and the number of units of the resource that are acquired, released, or passed. The system's resource architecture is dependent both on the deployment of the resources as well on how those resources are used. The complete picture of resource architecture can thus only emerge as CSM Scenarios are executed.

These resource relationships have been defined by Woodside as a *resource architecture* [116]. A particular aspect of resource architecture is the nesting of resource usages, which arises naturally in systems with remote procedure calls (RPC). Layered queueing models represent this nested use and its effect on resource holding times in a direct way. UML shows some nested resource usages directly with its representation of calls and returns, but non-nested usages are also important as a common performance optimization, which is represented in UML by asynchronous interchanges. Non-nested resources and resource contexts are explored in more detail in chapter 4.

3.4.2. Resource Handling during Scenario Execution

The CSM resource model shown in Fig 13 shows the relationships between the CSM resources types. *ActiveResource* and *PassiveResource* represent system resources. Active resources encompass *ProcessingResource* devices and placeholders for external services called *ExternalOperation*. Passive resources include operating system processes (or threads) identified as *Components*, which are hosted by *ProcessingResources*. All resources have an optional multiplicity.

In this way a standalone *Step* (but not a *Step* with a sub-scenario refinement) has a host resource through its associated component. Unprotected resources have been combined with protected resources, based on a multiplicity parameter which defines the number of units of the resource, such as a number of buffers, or of threads. An exclusively-held resource has a multiplicity of 1, while an unprotected resource is indicated by an infinite multiplicity. This avoids the need for separate classes and is consistent with resource notation in queueing models (e.g. [48]).

In addition to referencing resources, the resource operation steps - *ResourceAcquire*, *ResourceRelease*, and *ResourcePass* - also optionally reference the number of units of a multiple resource (the default number of resource units is 1). Resource units are defined as integers with the interpretations described in Table 3.

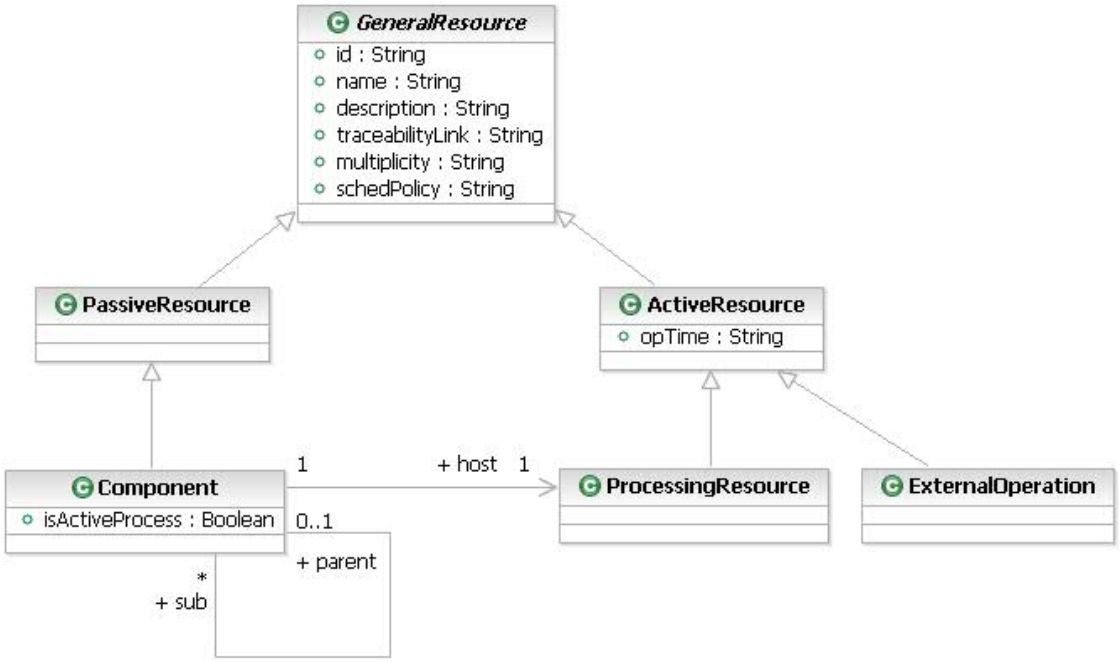


Fig 13 CSM v1 resource model

Table 3. Resource units and interpretations for resource operations.

Resource Operation Step	Resource Units	Interpretation
ResourceAcquire	not specified	1 unit (default)
ResourceAcquire	0	all units
ResourceRelease	not specified	1 unit (default)
ResourceRelease	0	all units
ResourcePass	not specified	1 unit (default)
ResourcePass	0	all units passed

The *ResourceAcquire*, *ResourceRelease*, and *ResourcePass* elements encountered as a scenario is executed create a *resource context* for the traversal. The resource context

encompasses all the resources that are held at a given point in a CSM traversal as well as the order in which the resources have been acquired.

ResourcePass – The resource context interpretation can be explicitly over-ridden by using *ResourcePass*. *ResourcePass* (resourceIdentifier, resourceUnits *[optional]*) specifies a resource and an optional number of resource units to be passed to a specific parallel or alternate subpath. If the number of resource units being passed is not specified, then the default assumption is that only 1 resource unit is passed.

ResourcePass indicates that only those resources specifically identified by *ResourcePass* are available to the particular parallel or alternate branch. When passing multiple resources, a separate *ResourcePass* must be used for each resource. The order of the resulting resource context after passing multiple resources follows the sequential order of the *ResourcePass* elements in the CSM. For well-formed CSMs, the use of *ResourcePass* is constrained so it must directly follow a *Fork* or another *ResourcePass*.

For parallel branches, the interpretation of *ResourcePass* is that the resources being passed are removed from the resource context of the other parallel branches. Thus the parallel branches that are not explicitly passed any resources share only the remaining resources. The resources that are passed to a parallel branch become protected from resource operations along any of the other parallel branches. As a result, it becomes necessary to reconcile the different resource contexts at a subsequent *Join*. The simplest reconciliation strategy is to aggregate all the resource from the different incoming parallel branches into a common resource context at the *Join*. Since all of the incoming parallel branches are always all traversed before proceeding past the *Join*, there is no problem with inconsistencies when aggregating the resource context at the *Join*. The remaining

research question when aggregating the resources is to determine the order in which the resources should be held in the aggregated resource context after the *Join*.

3.4.3. Well-Formed Scenarios

The following constraints are used to define a class of "well-formed" CSM Scenarios that can be used for further transformations and for the generation of performance models. Scenarios that do not conform to these constraints may be created and may have meaning, but they may not be a suitable source for further transformations. CSMs that are not "well-formed" are not acceptable input for the automated tools presented in later chapters.

"Well-formed" CSM Scenarios must satisfy the following constraints:

- Scenarios must begin with only a single *Start* point
- Scenarios must terminate with at least one *End* point
- The *source* and *target Steps* for a *PathConnection* must be different
- The *predecessor* and *successor PathConnections* for a *Step* must be different
- An "alternate branch" is defined as the sequential path fragment following a *Branch*
    - The first *Step* of an alternate branch specifies the probability of that alternate branch. The probabilities are treated as weights, as follows:
        - If the first *Step* of an alternate branch does not have a probability specified, then it is assumed to have a weight of 1
    - The weights of all the alternate branches following a Branch are normalized to sum to 1
    - Alternate branches must end with an *End* point or a *Merge*
- A "parallel branch" is defined as the sequential path fragment following a *Fork*
    - Parallel branches must end with an *End* point or a *Join*

In addition to the constraints listed above, the following convention is used to determine how CSM Scenarios are interpreted for performance modeling:

- Scenarios that begin with a *Start* point that has a *Workload* specified are executed independently
  - Scenarios that start with a *ClosedWorkload* cycle, in multiple concurrent executions, with one execution instance for each member of the population
  - Scenarios starting with an *OpenWorkload* do not cycle, but execute once for every arrival and then terminate
  - Scenarios starting with a *ClosedWorkload* must have one *End* point that terminates in the same component as the *Start* point
  - The *Workload* on independent *Scenarios* is ignored when those *Scenarios* are used as nested sub-scenarios
- *Scenarios* that begin with a *Start* point that does not have a *Workload* specified are not executed independently; their execution depends on the workload of other scenarios and they are executed only if they are included as nested sub-scenarios
- *End* points that have a *noSync* attribute set to *true* are assumed to stop the execution of the scenario at that *End* point and do not connect back to the outer scenario
  - If the *End* point is in an independent scenario that started with a *ClosedWorkload* then there is no cycling after that *End* point
  - If the *End* point is in a sub-scenario, then there is no return to the higher level scenario after that *End* point

Table 4 lists a set of validation checks that are performed automatically in order to identify CSMs that are not well-formed for generating performance models.

Table 4. Validation checks for well-formed CSMs

CSM Element	Validation Check	Success Condition	Failure Condition
Scenario	Scenario has a single Start point	Scenario contains exactly 1 Start element	Scenario contains 0 Start elements or 2+ Start elements
	Scenario has at least one End point	Scenario contains 1+ End elements	Scenario contains 0 End elements
Start	Start is not preceded by any other CSM element	The Start.*source* attribute is either undefined or empy	The Start.*source* attribute is defined and contains a value
	Start is followed by a single StepType element	The Start.*target* attribute contains the IDRef of exactly	The Start.*target* attribute contains the IDRef of 0 or 2+

CSM Element	Validation Check	Success Condition	Failure Condition
		1 StepType element	StepType elements
End	End is preceded by a single StepType element	The End.*source* attribute contains the IDRef of exactly 1 StepType element	The End.*source* attribute contains the IDRef of 0 or 2+ StepType elements
	End is not followed by any other CSM element	The End.*target* attribute is either undefined or empy	The Start.*target* attribute is defined and contains a value
Sequence	Sequence is preceded by a single StepType element	The Sequence.*source* attribute contains the IDRef of exactly 1 StepType elements	The Sequence.*source* attribute contains the IDRef of 0 or 2+ StepType elements
	Sequence is followed by a single StepType element	The Sequence.*target* attribute contains the IDRef of exactly 1 StepType element	The Sequence.*target* attribute contains the IDRef of 0 or 2+ StepType elements
Fork	Fork is preceded by a single StepType element	The Fork.*source* attribute contains the IDRef of exactly 1 StepType elements	The Fork.*source* attribute contains the IDRefs of 0 or 2+ StepType elements
	Fork is followed by 2 or more StepType elements	The Fork.*target* attribute contains the IDRef of 2+ StepType elements	The Fork.*target* attribute contains the IDRef of 0 or 1 StepType elements
Join	Join is preceded by a single StepType element	The Join.*source* attribute contains the IDRefs of 2+ StepType elements	The Join.*source* attribute contains the IDRef of 0 or 1 StepType elements
	Join is followed by 2 or more StepType elements	The Join.*target* attribute contains the IDRef of exactly 1 StepType element	The Join.*target* attribute contains the IDRefs of 0 or 2+ StepType elements
Branch	Branch is preceded by a single StepType element	The Branch.*source* attribute contains the IDRef of exactly 1 StepType elements	The Branch.*source* attribute contains the IDRefs of 0 or 2+ StepType elements
	Branch is followed by 2 or more StepType elements	The Branch.*target* attribute contains the IDRefs of 2+ StepType elements	The Branch.*target* attribute contains the IDRef of 0 or 1 StepType elements
Merge	Merge is preceded by a single StepType element	The Merge.*source* attribute contains the IDRefs of 2+ StepType elements	The Merge.*source* attribute contains the IDRef of 0 or 1 StepType elements
	Merge is followed by 2 or more StepType elements	The Merge.*target* attribute contains the IDRef of exactly 1 StepType element	The Merge.*target* attribute contains the IDRefs of 0 or 2+ StepType elements
StepType	A StepType element is preceded by a single PathConnection element	The StepType.*predecessor* attribute contains the IDRef of exactly 1 PathConnection element	The StepType.*predecessor* attribute contains the IDRef of 0 or 2+ PathConnection elements
	A StepType element is followed by a single PathConnection element	The StepType.*successor* attribute contains the IDRef of exactly 1 PathConnection element	The StepType.*successor* attribute contains the IDRef of 0 or 2+ PathConnection elements

3.5. DERIVATION OF CSM FROM UML

The generation of CSM from UML1.4 and UML2 Sequence Diagrams and Activity Diagrams was done by Tauseef Israr and is described in more detail in [48]. The contributions of the author to this work consisted of the definition of the basic correspondences between UML and CSM constructs and the formulation of the basic rules regarding the traversal of UML models and the generation of CSM objects. The author was also consulted heavily by Israr. For completeness, the underlying rules for the transformation are outlined in this section.

The basic rules for deriving CSMs from UML are as follows:

- Deployment Diagrams
  - nodes are CSM ProcessingResources
  - artifact manifestations are CSM Components
  - host associations between CSM Components and ProcessingResources are derived from the deployment associations
- Sequence Diagrams
  - lifelines are CSM Components
  - execution occurrences are CSM Steps
  - sending a synchronous call message results in a CSM ResourceAcquire for the component representing the lifeline receiving the message
  - sending a synchronous reply message results in a CSM ResourceRelease for the component representing the lifeline that sends the message
  - sending an asynchronous message results in a CSM ResourceRelease for the component representing the lifeline that sends the message and a ResourceAcquire for the component representing the lifeline receiving the message
  - spontaneous execution occurrences imply a CSM ResourceAcquire for the component corresponding to the lifeline on which the execution occurrence appears

    - all the CSM components are released when the scenario ends
- Activity Diagrams
    - swimlanes represent the behaviour of CSM Components
    - activities are CSM Steps
    - when the activity sequence changes swimlanes it results in a CSM ResourceRelease for the component representing the swimlane that is being left and a ResourceAcquire for the component representing the swimlane that is being entered
    - all the CSM components are released when the activity graph ends

In a Sequence Diagram a step is represented by a focus of control (the rectangles on the lifelines in A Sequence Diagram and a corresponding CSM for a synchronous interaction). One focus of control can carry several interactions; in this case the assumption is that the CSM has a *Step* for the processing first, and then that Step is followed by the paths for the interactions. If there are nested foci of control, they generate additional sequential CSM *Steps* and those may be mixed in sequence with interactions. If a nested focus of control also shows interactions then, as before, the CSM places the Step before the interaction behaviour.

Fig 14 shows a simple example of a UML Sequence Diagram with `<<PAresource>>` stereotypes on the lifelines associated with components, and `<<PAstep>>` stereotypes on messages that invoke Steps, together with the CSM that corresponds to it. Because the call to *doB* is synchronous and thus is assumed to block *CompA*, and therefore *CompA* is not released when *CompB* is acquired. (The diagram uses the SPT stereotypes PAresource and PAstep which in MARTE would be PaRunTInstance and PaStep).

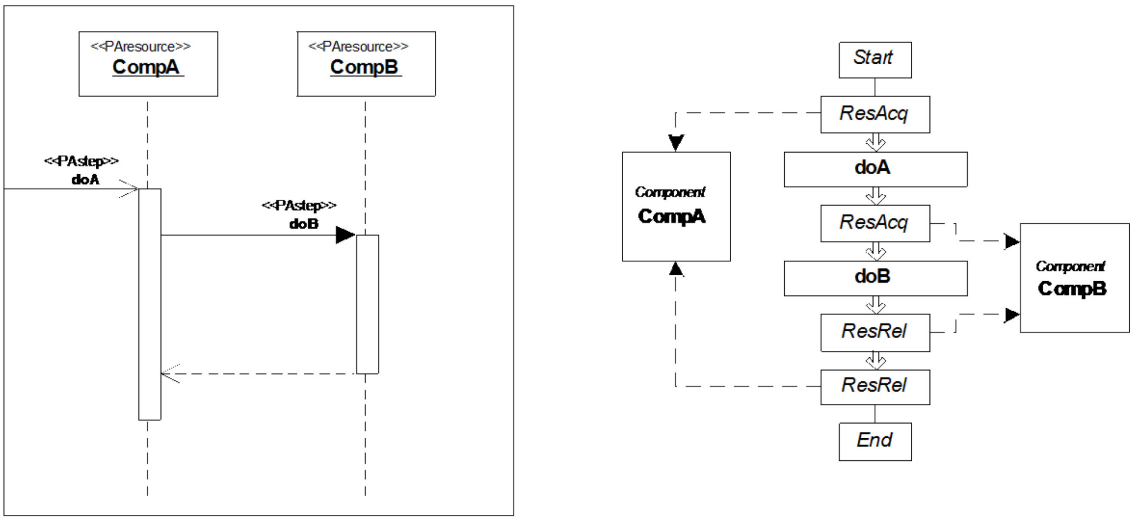


Fig 14 A Sequence Diagram and a corresponding CSM for a synchronous interaction

A sub-scenario within a Step may have multiple end-points due to forking of flow. In an Activity Diagram this may arise within a SubActivity state (an Activity with refinement), and in a UML2 Interaction Diagrams it may arise within a par combined fragment. The connection to the behaviour following the high-level Step may be made in two ways:

- the paths within the sub-diagram synchronize and then exit together,
- or some paths do not synchronize and survive the end of the sub-behaviour, to continue in parallel.

The synchronized exit is the default, and the <<`noSync`>> attribute on the first Step following the fork marks a path that should not synchronize.

More detail on MARTE elements is given in section 2.3.2 in relation to the corresponding CSM elements introduced later. UML State Machine Diagrams can also be annotated (e.g.), but are not described here.

Annotations define resource capabilities and demands, parameters of workloads and behaviour, and values of performance. Precise and consistent definitions are basic to

PUMA. The essential *capabilities of a resource* are its rate of execution of operations and its multiplicity. As described earlier, PUMA identifies four kinds of resources:

- *ProcessingResource*, a CPU or other device which hosts and executes operations, possibly a multiprocessor. Its operation time is its cycle time.
- *ExternalResource*, providing a service defined externally to the design (e.g. a file operation). Its attributes are known to the modeling environment.
- *Component*, an operating system process and its thread pool,
- *LogicalResource*, anything else, (e.g. mutexes, memory or buffer pools, and locks).

The last two are *logical* resources whose operation time depends on the behaviour which is executed while holding them. Resource demands include CPU time (host demand) for Steps, which may be in units of machine cycles, but commonly states time values for the target CPU. External operation demands give the number of requests.

The CSM interpretation of UML entities and constructs summarizes the CSM interpretation of UML entities.Please note that CSM Components are based on *instances* in the UML. CSMs can be derived from any combination of UML Activity Diagrams, Sequence Diagrams, and Deployment Diagrams that have the requisite information as to the system structure and behaviour.

Table 5. The CSM interpretation of UML entities and constructs

Concept	UML –SD	UML -AD	CSM	Main Attributes
scenario	Sequence Diagram	Activity Diagram	Scenario	
start	inserted before the first Step of a Scenario	InitialNode/as UML-ID	Start	association to a workload entity
operation	ExecutionInstance or message	Action	Step	probability, repetitions, demands
operation with refinement	Combined Fragment (CF)	Structured-ActivityNode	Step with Refinement	probability, repetitions, sub-scenario association

Concept	UML –SD	UML -AD	CSM	Main Attributes
refinement sub-scenario	CF operand	Activity	Scenario	
sequence	causal sequence (see Sec. 3.3)	ActivityEdge	Sequence	
OR-fork	alt or opt CF	DecisionNode	Branch	(probability is on first Step on the branch)
OR-join	end of alt CF	MergeNode	Merge	
AND-fork	par CF, or sending an asynchronous message.	ForkNode	Fork	
AND-join	end of par CF, or receiving an asynchronous message.	JoinNode	Join	
acquire resource (process)	implicit at first Step in a new Lifeline.	implicit after an ActivityEdge to a new ActivityPartition	ResourceAcquire	Component
release resource (process)	implicit at reply, or end of ExecutionInstance	implicit before an ActivityEdge to a new ActivityPartition	ResourceRelease	Component
acquire (explicit)	message with «GaAcqStep»	node with «GaAcqStep»	ResourceAcquire	
release (explicit)	message with «GaRelStep»	node with «GaRelStep»	ResourceRelease	
end	after entity with no successor	FinalNode (if more than one, interpreted as FlowFinal)	End	
workload	GaWorkloadEvent	as UML-ID	ClosedWorkload or OpenWorkload, depending on attributes	
component association	implicit in Lifeline “represents” association	implicit in ActivityPartition	Component	multiplicity, discipline
external resource	extOp attribute of a Step	as UML-ID	ExternalOperation	number of operations
Concept	**Deployment Diagram**		**CSM**	**Main Attributes**
processor	Node which is stereotyped «GaExecHost»		ProcessingResource	multiplicity, cycle time, discipline
logical resource	«LogicalResource»		LogicalResource	multiplicity, discipline

3.5.1. TPC-W Example

The Transaction Processing Performance Council's transactional web benchmark (TPC-W) [120] models an on-line bookstore.

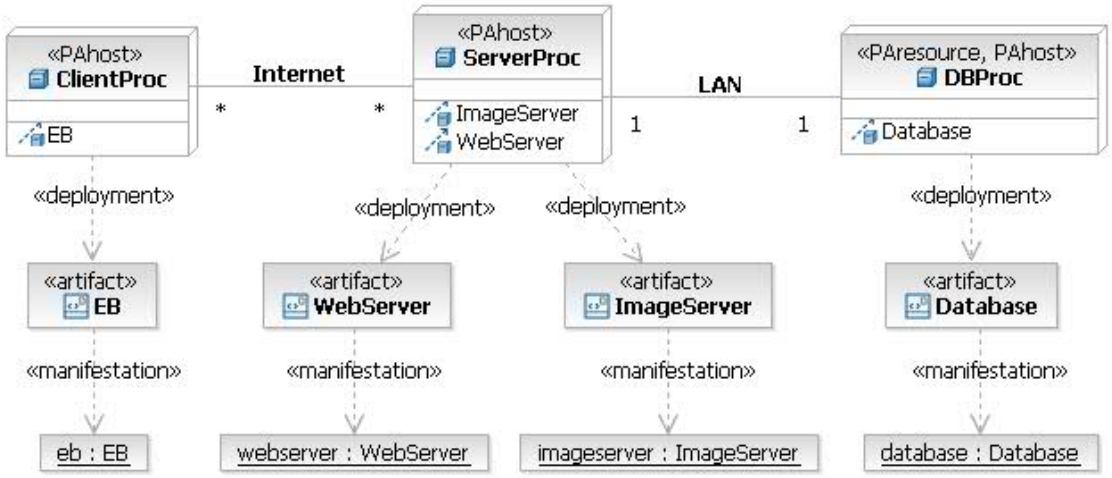


Fig 15 Deployment diagram for TPC-W

The components of TPC-W are logically divided into three tiers: a) set of emulated web browsers (EB), b) web tier including Web Servers, Image Servers and Web Caches and c) persistent storage. TPC-W simulates customers browsing and buying products from a website.

The TPC-W specification describes 14 different web pages that correspond to typical operations performed by a customer of an e-commerce website. The first page to be visited by a user is the "Home" page; it includes the company logo, promotional items and navigation options to the top best-selling books, a list of new books, search pages, shopping cart, and order status pages. At every page, the user is offered a selection of pages that can be visited next; the user is assumed to make a random choice. The user may browse pages containing product information, perform searches with different keys and put items in the cart, or may decide to place an order. When ordering, a new

customer has to fill out a customer registration page; for returning customers, the personal information is retrieved from the database and filled in automatically. Before ordering, the user may also update the shopping cart content. After deciding to buy, the user enters the credit card information and submits the order.

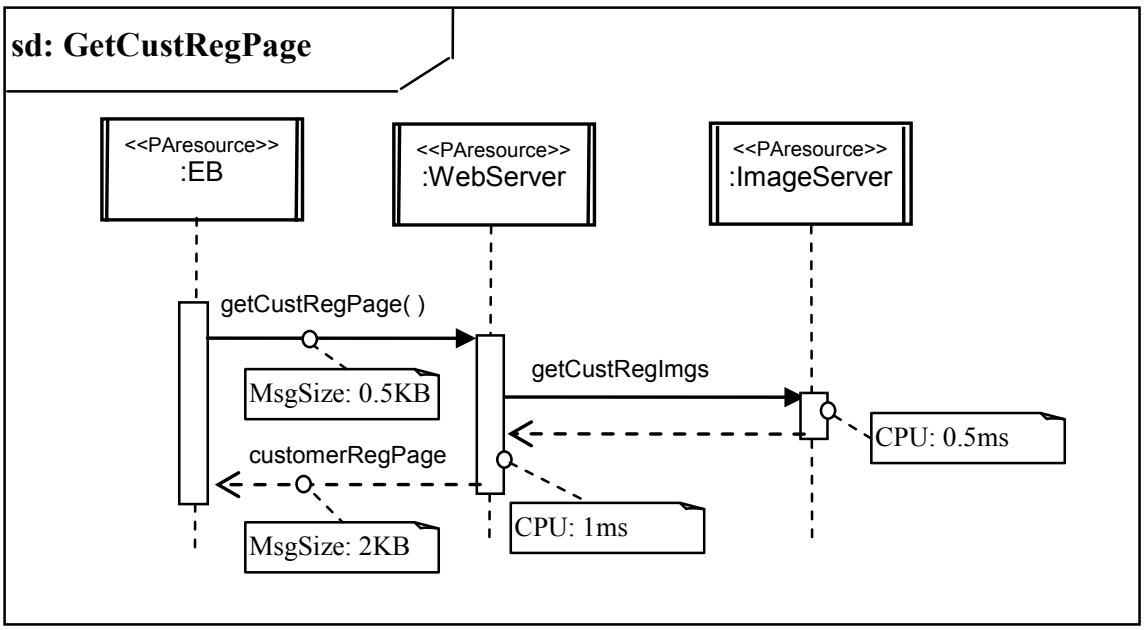


Fig 16 Sequence Diagram for the TPC-W GetCustRegPage scenario, with informal SPT annotations

The system obtains credit card authorization from a Payment Gateway Emulator (PGE) and presents the user with an order confirmation page. The user can also view the status of previous orders. Two additional web pages are provided for the system administrator. The navigation options provided on every page, lead to an access distribution referred to as the "Web Interaction Mix". Thus 80% of the web page accesses are to the Home, New Products, Best Sellers and Search pages while the remaining 20% of the accesses are to the Shopping Cart, Order, Buy and Admin web pages.

To illustrate how CSM is derived from UML, two TPC-W scenarios were chosen: Fig 16 shows a simple scenario that returns the customer registration page and Fig 17 shows a more complicated scenario where the user places an order.

The UML model contains the structural and behavioural views required for performance evaluation [94]:

- Deployment of high-level software components to hardware devices (Fig 15)
- One or more key performance scenarios annotated with SPT performance information modeled as interaction diagrams (Fig 16 and Fig 17).

The Deployment Diagram shows the software components, their corresponding artifacts and the deployment of artifacts on processing nodes. The DBProc node is stereotyped with both PAhost and PAresource since it has a multiplicity of 5. The PAhost stereotype identifies the node as a host, while the PAresource stereotype is needed to specify the multiplicity. If the multiplicity is not specified, it is assumed to be 1.

The Sequence Diagram for the *GetCustRegPage*, shown in Fig 16, returns the registration web page to the Emulated Browser (EB) that models the user. This scenario is interesting because it starts with a non-secure request for a registration/login page from the EB to the WebServer which ends with a reply returning the registration/login page which should really be encrypted. The user will use the returned page to register as either an existing or a new customer in another interaction (not shown here). The following operations are performed:

- EB issues a request for the customer registration page;
- WebServer gets the necessary images (company logo, button images, etc.) from ImageServer;
- WebServer constructs the HTML customer registration page and returns it to EB.

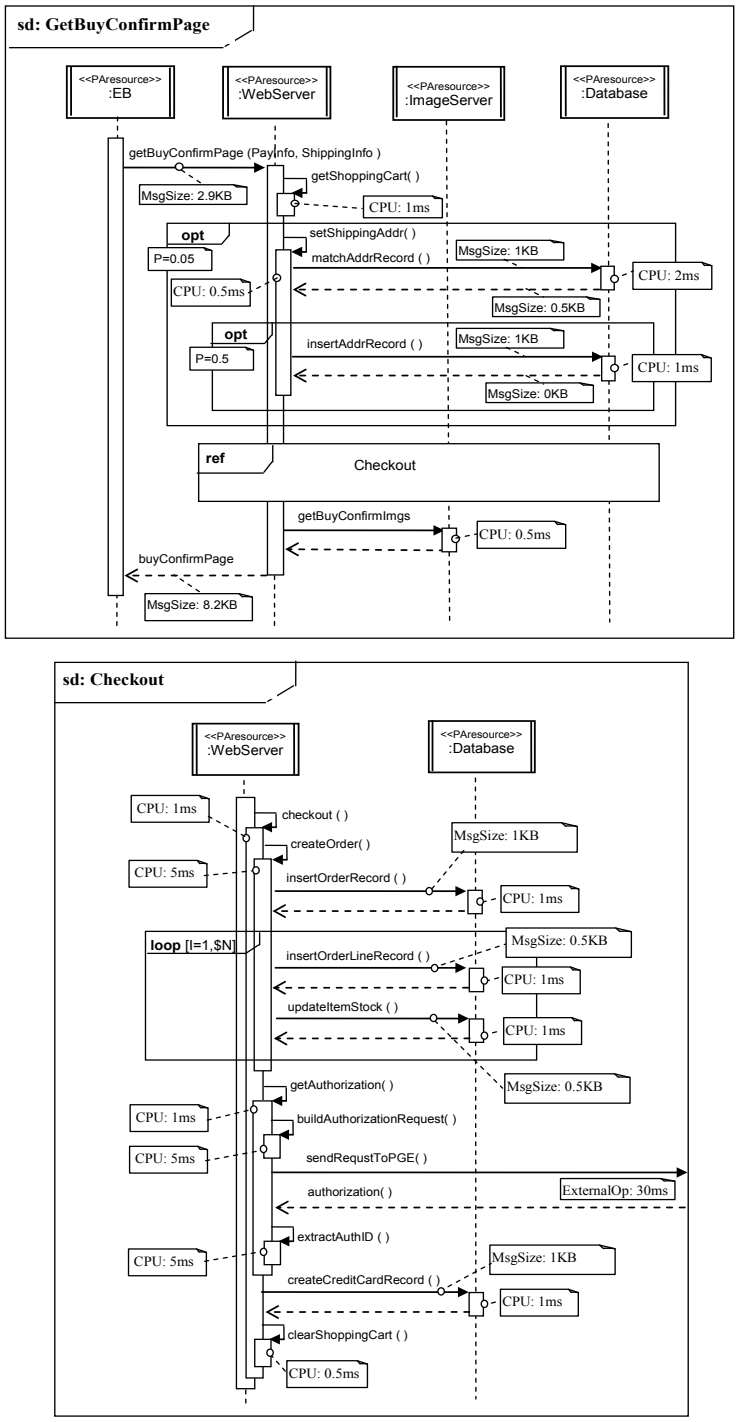


Fig 17 Sequence Diagrams for the TPC-W GetBuyConfirmPage and Checkout scenarios, with informal SPT annotations

The *GetBuyConfirmPage* scenario is described in the two interaction diagrams shown in Fig 17. The top interaction transfers the shopping cart content into a newly created order

for the registered customer and executes a full payment authorization, then returns a web page containing the details of the newly created order to the EB.

The following operations are performed:

- EB issues a request to WebServer for “buy confirm page”;
- WebServer gets the corresponding shopping cart object;
- With 5% probability, a shipping address is passed from EB.
- WebServer tries to match the shipping address in the corresponding table in the database
    - If no address record is found, it inserts a new address record
- The Checkout sub-scenario is invoked (shown as a ref fragment)
- WebServer gets necessary images from ImageServer
- WebServer constructs the HTML code for the buy confirm page and returns it to EB.

The Checkout sub-scenario is shown in the bottom Sequence Diagram in Fig 17. It creates a new order in the database, with all the items in the cart turned into order lines. Then an authorization is obtained from the PGE which is actually an external system. Finally the credit card is registered in the database and the cart is cleared.

The performance annotations used in this model are:

- CPU host demand in milliseconds for operations (applicable to Execution Occurrences and Messages in the Sequence Diagram)
- the probability for *alt* and *opt* Interaction Operands
- the repetition count for *loop* Interaction Operands
- the processor speed for host devices (applicable to nodes stereotyped as <<PAhost>>)
- device multiplicity
- the PGE is represented as an “external operation” (a tagged value of the stereotype PAstep that indicates the name of the external operation and the number of visits [86]). It is represented in the performance model as a new task.

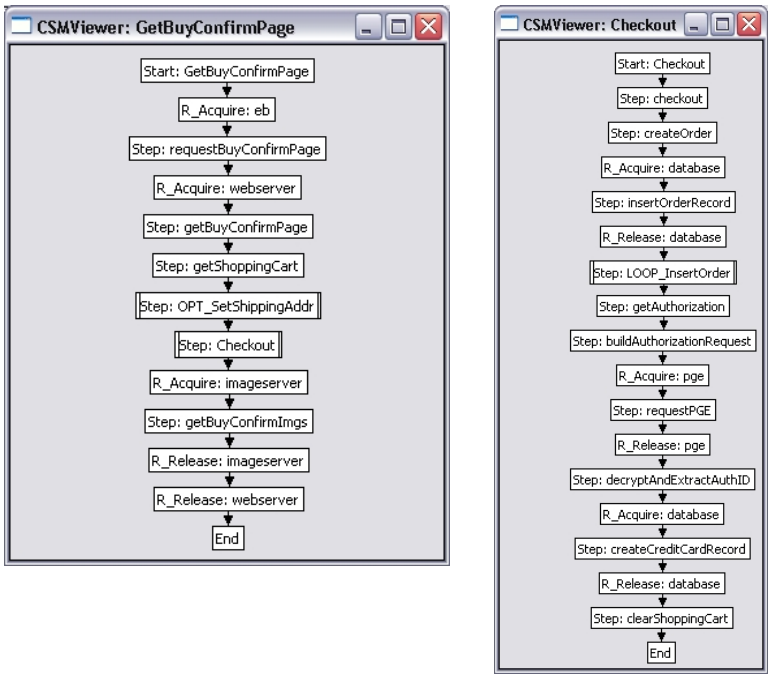


Fig 18 CSM for the TPC-W GetBuyConfirmPage and Checkout scenarios

The figures use informal performance annotations, in order to limit the clutter. However, the complete stereotypes and corresponding tagged values were defined in the Rational Software Architect (RSA) UML tool used to generate the CSM models. This work used an extended SPT profile.

Fig 18 shows the corresponding CSM scenarios for *GetBuyConfirmPage* and *Checkout* as displayed by the CSM Viewer. The CSM Viewer is an Eclipse and plug-in that generates a visual representation of the scenarios in a CSM file. Scenario elements are shown as rectangles with the type specified on the left-hand side (*R_Acquire* and *R_Release* are shorthand for *ResourceAcquire* and *ResourceRelease* respectively). Sequential path relationships are shown as arrows. Steps with sub-scenario Refinements are shown as rectangles with double bars on the side. A Step with a Refinement is clickable in the CSM Viewer and the corresponding sub-scenario for the refinement is displayed in a new window when it is clicked. In this case, “Checkout” is a Step with a Refinement in the

GetBuyConfirmPage scenario and the corresponding Checkout sub-scenario is also shown in Fig 18.

3.6. ASPECT-WEAVING IN CSM

Aspect-Oriented Modeling (AOM) decouples secondary or supporting functions from the main functionality. AOM allows for separate functionalities to be described independently. These separate functionalities can then be *woven* into different system configurations as needed.

The description of the main functionality is called the *primary model* while the secondary functions are described as *aspect models*. A *join point* is the point in the primary model where the aspect model should be added in. Aspect *weaving* is the process of plugging in the aspect model into the primary model at an appropriate join point.

Aspects are behaviour and associated resources. As such, CSM is well suited both for describing and for weaving aspect-oriented models. Using CSM for AOM leverages the ability to parameterize CSMs introduced in section 3.3.3. Parameterized scenarios and steps can be used to describe the aspect behaviour and parameterized resources can describe the resource roles required by the aspect behaviour. CSM for AOM uses a simple parameter naming convention whereby parameterized variables start with a '$' character while role placeholders start with a '|' character.

Features that address pervasive or "cross-cutting" concerns have been described separately as aspect models (e.g., for security, [36] [98]). Aspect models have been composed into a primary design model via expression as UML class diagrams [29], interaction diagrams [36] [57], statecharts [44] [57], and activity diagrams in [10] [98] [110]. However, using aspects defined with different diagram types will introduce

accidental complexity in the methods, and require additional checking that the results of composition are consistent. Using CSM for aspect composition eliminates this risk and difficulty. Additionally, aspect weaving has performance implications due to changes in overall system behaviour and/or resource architecture (i.e. aspects for messaging can change the system's protocol stack and its associated behaviour; aspects for distributed locking and mutual exclusion involve changes to behaviour around resource use; security aspects change behaviour and resourcing with respect to credential validation and authentication, as well as messaging behaviour).

The advantages of using CSM for AOM aspect weaving are thus threefold:

- The UML models can include any of the UML behaviour formalisms, and can combine a mixture of interaction diagrams, activity diagrams and statecharts with class and deployment diagrams into a single analysis. These UML diagrams must be annotated with performance properties using MARTE [87]. A single aspect composition algorithm addresses all these combinations
- The extracted CSMs are smaller than the original UML. Therefore the composition process is both simpler and more robust, as it does not raise consistency issues between composition operations carried out within different UML diagrams.
- The CSM form combines behaviour, resources, component instances and the MARTE performance attributes (such as CPU consumption: the aspect behaviour may have different CPU consumption when composed in different locations). Thus it has all the necessary information, in a form suitable for the composition, which must address all these factors.

This approach of composing aspects in a scenario-based model is conceptually similar to that used in Aspect-Oriented Use Case Maps (AoUCM) which was developed in separate research by G. Mussbacher, D. Amyot, and M. Weiss and which is described in [77] [78].

The AoUCM approach focuses on capturing system requirements and was further refined into Aspect-Oriented User Requirements Notation (AoURN) by Mussbacher in [79].

WEAVING SECURITY ASPECTS

NFRs such as security and performance may easily be in conflict in a complex distributed system with sensitive data and many users. An example is the re-design of the Secure Sockets Layer (SSL) protocol proposed in [56], which was motivated by the goal to reduce the performance cost on the server in a SSL connection. While the performance goal was achieved, the main security goal of the SSL protocol, namely to set up a secure connection, was violated in the re-designed version (see [5] for more details). If such impacts are analyzed early in the software development, then drastic and expensive changes may be avoided.

The author led the effort to design a practical aspect weaving method for a feasibility study of aspect-oriented CSM, reported in [99]. The study involved the addition of SSL security to a web server. This first version of the work on aspect-oriented CSM used an ad-hoc composition of security aspect behaviour into the primary scenario, at locations specified directly by the designer. The method was subsequently refined by collaborators into a formal framework for general aspect composition of scenarios given as CSMs [100], useful for all kinds of aspects. Properties of the CSM entities are expressed in a form suitable for operating on the model, and operations that are useful for AOM are defined. Very general predicate-logic conditions govern the location and style of composition, and the composition of quantitative performance attributes is specified separately.

The process of using CSM for AOM is as follows:

- Describe the primary model and the aspect models in separate CSMs
  - Use parameters for variables (`$variable`) and role names for resources (`|role`) when elements in the aspect models are to be instantiated as elements in the primary model after weaving
- Identify the join points in the primary model where aspects are to be woven in
- Create *specific aspect model* instances corresponding to each instance to be woven
  - Where appropriate, map aspect variables and roles to existing elements in the primary model
- Weave resources into the primary model by adding resources from the aspect models into the primary model CSM where they do not already exist
- Add the aspect behaviour as sub-scenarios in the primary model at the designated join points
- Weave the aspect behaviour into the primary model by flattening the sub-scenarios into the primary model (use the flattening algorithm described in section 5.3.5)

An example of using CSM for AOM to weave security aspects into a distributed transaction processing system is described in the rest of this section. The example was presented at the WOSP 2007 conference [100]and was later published as a journal paper in [127].

The goal of the study was to determine the performance properties of alternative security features presented as UML design model aspects. Previous work has first composed the security aspects with the primary design model in the UML domain [96] and then carried out the performance analysis. In particular, Petriu et al. in [98] [118] employed the PUMA approach to evaluate performance after composing security aspects in UML. Performance concerns require changes to the usual aspect-oriented modeling and model composition, including special attention to resource usage, and to composing the

performance parameters, and the use of concurrency and deployment model views which are essential for performance analysis.

3.6.1. Example – Primary Model

The primary model for the web server is based on TPC-W [120]. The primary model represents the basic functionality of an on-line bookstore without any security mechanisms. SSL secure communication is later added to the primary model through aspect composition.

The primary model does not implement SSL encryption for account pages and for obtaining credit card authorization from the PGE. The navigation options provided on every page, lead to an access distribution referred to as the "Web Interaction Mix". Thus 80% of the web page accesses are to the Home, New Products, Best Sellers and Search pages while the remaining 20% of the accesses are to the Shopping Cart, Order, Buy and Admin web pages. Of those ordering interaction, a quarter or 5% of the accesses are to secure web pages requiring SSL encryption.

To illustrate how security aspects can be composed at the CSM level, the entire functionality of TPC-W is not used for the primary model. Instead two scenarios that access secure pages were chosen: GetCustRegPage which is shown in the Sequence Diagram in Fig 16 and the more complicated GetBuyConfirmPage and Checkout scenarios shown in Fig 17. The initial deployment for TPC-W is shown in Fig 15.

The chosen TPC-W scenarios are explained in more detail in section 3.5.1 and the derived CSMs are shown in Fig 16 and Fig 17.

3.6.2. Example – Generic Aspect Model

A generic aspect model describes the solution proposed by the aspect in a general way, not related to the specific primary model in which will be eventually inserted. According to [36] a generic aspect model can be instantiated multiple times to produce multiple *context-specific aspect models* based on different *binding rules*.

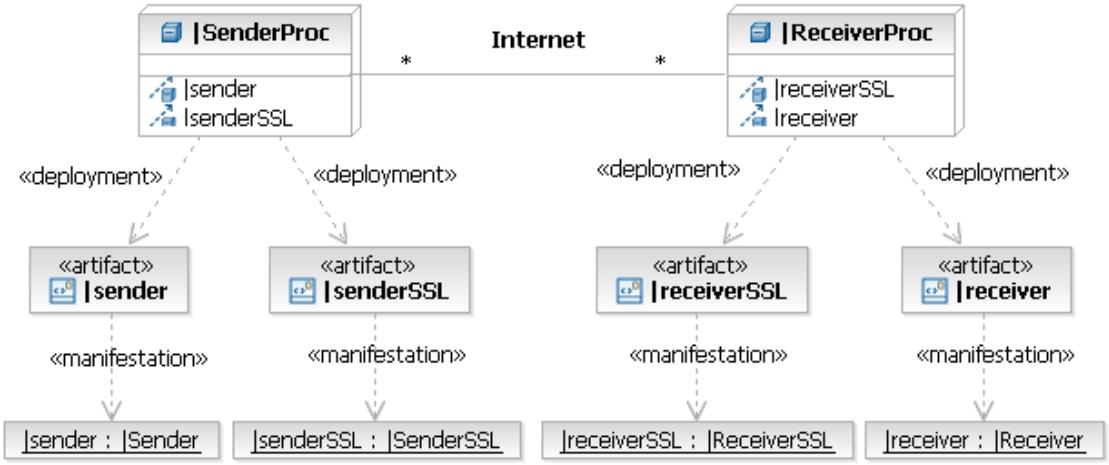


Fig 19 Deployment diagram for SSL data transfer aspect

The SSL data transfer is modeled as a generic aspect. Fig 19 shows a Deployment Diagram describing constraints on the structure, as well as an interaction diagram describing the behaviour. In this case, the structural constraint is that the SSL proxies must be located on the same node as the processes they are associated with. All the nodes are generic; they will be bound to specific nodes later in the process of instantiating the context-specific aspects.

The SSL data transfer interaction diagram in Fig 20 involves four generic roles: `|sender` (the data source), `|senderSSL` (data source SSL proxy), `|receiver` (data target) and `|receiverSSL` (data target SSL proxy). The convention used is that generic role names

start with a '|', similar to [36]. These roles are to be bound to application components when the generic aspect is instantiated to a specific context.

A message from `|sender` is first broken into fragments by `|senderSSL`. The number of fragments depends on the length of the data to be transferred. For each fragment, the source counter is incremented; this is the unique counter for the data source. Both the target and source counters are appended to the fragment and a digest is created across this string, using a secret digest string. The digest is appended onto the fragment and then encrypted using the symmetric key exchanged during the handshake phase (resulting in a payload). A header is pre-pended to this information, which contains the type of the message, the length of the fragment and digest, and the SSL version number used by the data source. This entire entity is the record that is sent to the target; `|receiverSSL` increments the source counter, extracts the header, decrypts the payload using the symmetric key, extracts the fragment and digest, and validates the digest using the secret digest string. If either the decryption or the digest validation fails, the receiving target sends an alert to the data source that indicates the failure type.

Depending on the overall application protocol (which is independent of the SSL protocol), the data source may attempt to re-send the record, or terminate.

Fig 21 shows the automatically generated CSM scenarios for SSL data transfer.

It is important to mention that the performance annotations in the generic aspects use variable placeholders instead of concrete values for the tagged values. These variables will be assigned concrete values only after the instantiation of the generic aspect to a specific context.

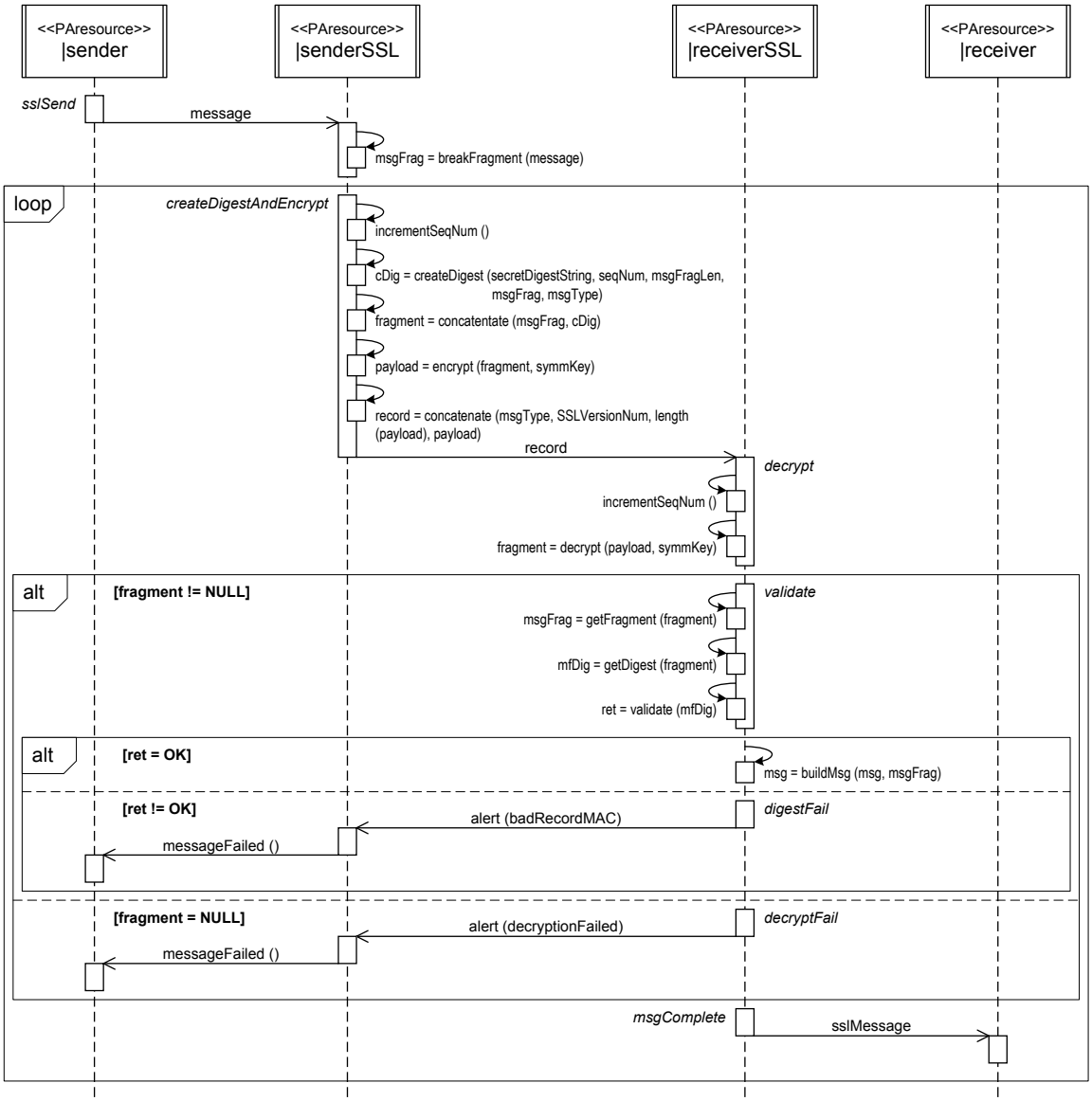


Fig 20 Sequence Diagram for the SSL Transfer scenario

The description of the SSL data transfer aspect model raises a general issue: what level of detail is appropriate for the UML model when trying to integrate the analysis of multiple non-functional properties - in this case, security and performance. The interaction diagram in Fig 20 gives a detailed functional description that is necessary for the logical verification of the security mechanisms by using a first-order logic model, as in [38]. However, for performance analysis a coarser granularity level would be more

appropriate. For instance, many of the small sequential steps could be aggregated into larger steps, which would need fewer performance annotations. The aggregation could be done automatically, under the user's guidance. The user needs to be involved if he/she would have to enter performance annotations only for the coarser-granularity steps obtained by aggregation. Fig 21 gives the aspect CSMs.

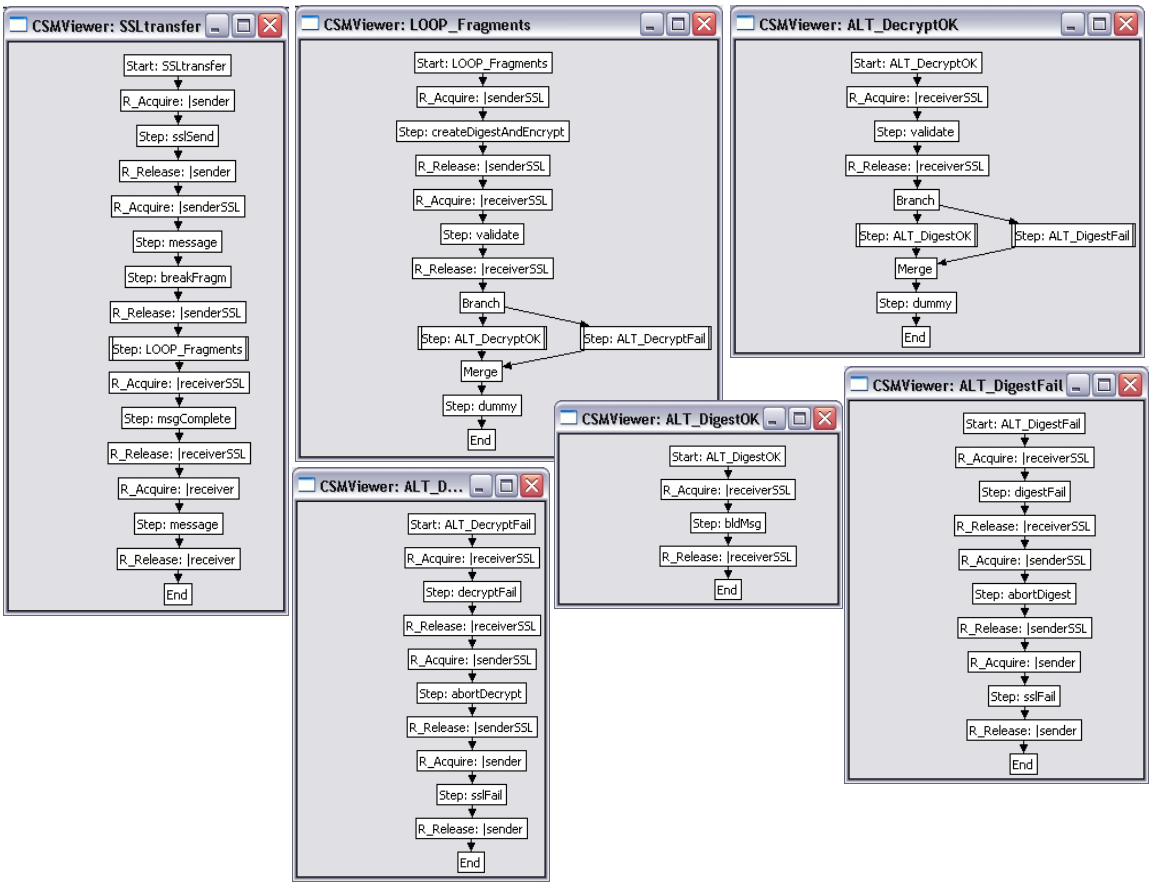


Fig 21 CSM for the SSL Transfer scenario and related sub-scenarios

3.6.3. Example – Specific Aspect Model

Before composing the aspect with the primary model, the *generic aspect model* for a given application context must be instantiated by binding the roles to application-specific values. As already mentioned, a generic aspect model can be instantiated multiple times to produce multiple *context-specific aspect models* based on different *binding rules*.

Because SPT or MARTE annotations are necessary for performance analysis, the binding rules have two parts: one for binding generic roles to components/nodes from the primary model, and the other for giving concrete values to the performance parameters.

The composition of the aspect models with a primary model can be performed at different levels: UML, CSM or LQN. There are many papers in the literature focused on the composition at the UML level. For instance, in [36] [107], the generic aspect models are defined by using the concept of *UML templates*. The context-specific models are obtained from generic models by binding the parameter templates to values from the primary model context. The advantage of composing at the UML level is that the resulting model is also in UML; therefore, it may be further visualized, developed or analyzed with tools that operate directly on UML models. A disadvantage is that the UML metamodel is very complex; this has a direct impact on any composition algorithms.

Table 6. Context-specific aspect resource bindings.

Generic Aspect	*Context-Specific Aspect*	
SSLtransfer	**SSLcall**	**SSLreply**
|sender	Eb	webserver
|senderSSL	ebSendSSL (new)	webSendSSL (new)
|receiverSSL	webRcvSSL (new)	ebRcvSSL (new)
|receiver	Webserver	eb
|senderProc	ClientProc	ServerProc
|receiverProc	ServerProc	ClientProc

When performing the aspect composition at the CSM level, a distinct advantage is that CSM was defined to be a unique target for transformation from many UML versions and diagrams, and a source to many performance models. For instance, aspect-oriented CSM is able to compose aspects with primary models even if they are originally defined in

different UML behaviour diagrams (i.e., any mix of activity, sequence, communication, and interaction overview diagrams can be handled at the CSM level). Another advantage is that the CSM metamodel is much simpler than the UML metamodel, and therefore the composition algorithms are easier to design and implement. An obvious disadvantage is that CSM, which is designed to model scenarios, is much more restricted in scope and usage than UML.

The first step in creating aspect models from generic CSM models is to make the aspect models into sub-scenarios and apply a CSM Resource Acquire/Release Clean-Up for sub-scenarios transformation.

Generic aspect models are transformed into context-specific aspect models by binding the resource roles to actual resources and then assigning context-specific performance values to step processing demands, branching probabilities, optional probabilities, and loop repetition counts.

The first step in transforming the generic aspect model into a context-specific aspect model involves binding the generic resource roles `GR` to context-specific resources `SR`. These context-specific resources can be either existing resources `PR` from the primary model, or new resources required by the aspect.

The resource binding algorithm is:

```
for all GRi:
  if GRi has a corresponding PRj then
    SRi = PRj
  else
    SRj = instantiate( GRi )
```

In the TPC-W `GetBuyConfirmPage` example, the generic *SSLtransfer* aspect is bound to two different context-specific aspects; a context-specific *SSLcall* aspect for the `EB` calling

the `WebServer`, and a different context-specific *SSLreply* aspect for the `WebServer` replying to the `EB`. The resource bindings for the context-specific aspects are given in Table 6.

The next step in getting a context-specific aspect model is to assign values to the performance annotations: execution demands, branching probabilities, etc. The concrete annotations can be either values or expression (for instance, the number of loop repetitions depends on the message size $MSG_SIZE, which depends on the join point into the primary model).

Table 7. Performance values for the top-level scenario in both SSLcall and SSLreply context-specific aspect models

Step	Service Demand
sslSend	0.1
Message	0.1
msgComplete	0.1
sslMessage	0.1
Loop	**Repetition Count**
LOOP_Fragments	ceiling($MSG_SIZE / 512)

The values for the performance annotations used in the *SSLcall* and *SSLreply* top-level scenarios are given in Table 7. The service demands have literal values, while the repetition count for *LOOP_Fragments* is an expression indicating that the repetition count is equal to the message size divided by the fragment size (512 bytes) and rounded up to the nearest integer.

The performance values are the same for both the *SSLcall* and *SSLreply* context-specific aspects because the two aspects are symmetrical in this example. This is not always the case, and it may be possible to have different context-specific aspects based on the same

generic aspect that have different performance values (e.g., different service demands for encryption and decryption due to using different encryption algorithms).

It is worth mentioning that the communication between the `WebServer` and `PGE` taking place in the `Checkout` sub-scenario must be secure, as well. This would require additional work to instantiate the SSL generic aspects to another context. The current example assumes instead that the latency of the external operation that accesses `PGE` includes the overhead for SSL transfer.

3.6.4. Weaving Aspect Models

The CSM composed model is generated by weaving the context-specific aspect models into the primary model. The weaving involves identifying the appropriate join points in the primary model behaviour and inserting the context-specific aspects at those join points. As part of the weaving, an aspect's resource context must also be reconciled with the primary model resource context at the join point. Finally, the woven aspects are inspected for any remaining performance annotations that can be further resolved.

Fig 22 shows the composed CSM model for *GetBuyConfirmPage* with SSL data transfer between `EB` and `WebServer`. For this example, the join point for the *SSLcall* aspect is the *call* step, while the join point for the *SSLreply* aspect is the *reply* step. The weaving is done by replacing the join point steps with complex steps – *call* and *reply* are replaced with *SSLcall* and *SSLreply* – and using the context-specific aspects as refinements for those complex steps.

As part of the resource context reconciliation during weaving, the *SSLcall* aspect loses the *ResourceAcquire:eb* element at the beginning since `EB` is already acquired in the primary model before *SSLcall* is invoked as well as the *ResourceRelease: webserver* at

the end, since it is already released in the primary model after the aspect completes. Similarly, the *SSLreply* aspect loses the first *ResourceAcquire: webserver* and the last *Resource Release: eb*.

The aspect composition is straightforward in this case because the SSL aspect was used to replace simple non-secure messages in the primary model. This allowed us to substitute simple CSM steps with one input and one output with composed steps with one input and one output. In the general case, an aspect may require more than a single input and/or output, which leads to a more complex composition. An example is an aspect model containing alternative or parallel behaviours, where the respective branches need to be "attached" to the primary model in different input or output points.

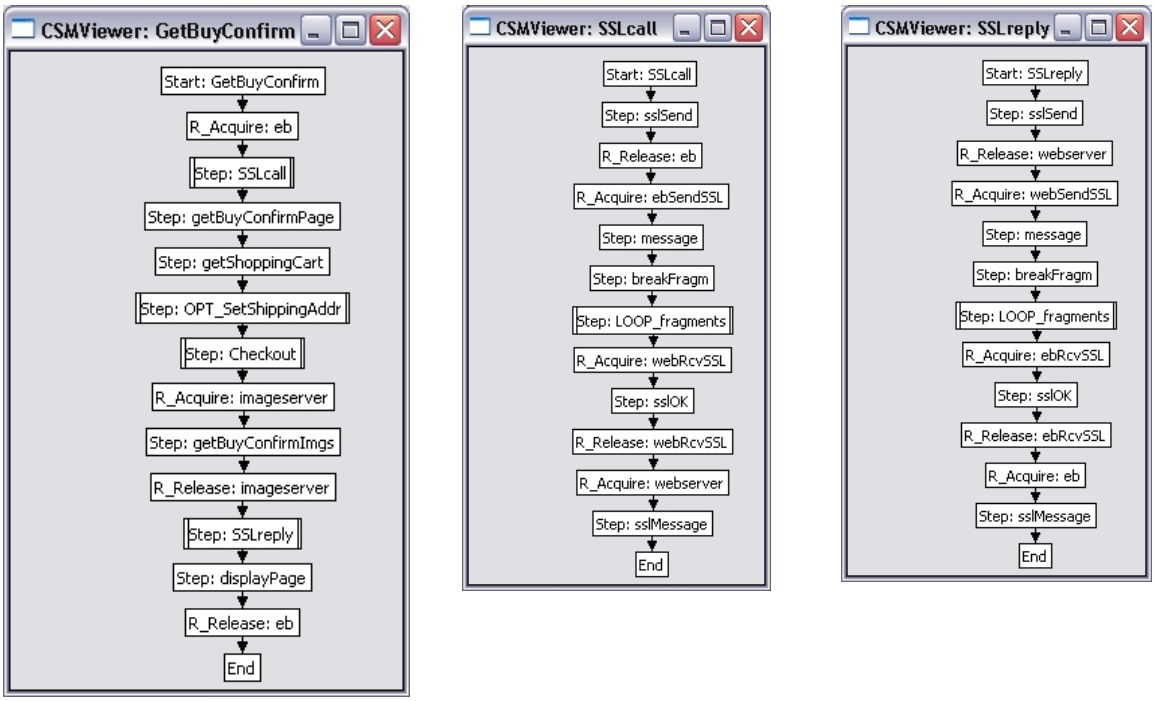


Fig 22 CSM for the composed model for GetBuyConfirmPage with SSL

A more general composition approach involves defining aspect join contexts instead of just join points. Instead of being simple steps, join contexts are either CSM path fragments (i.e. sequences of steps with single beginnings and single ends) or

combinations of CSM path fragments. Those path fragments are then used to generate sub-scenarios and are replaced by complex steps using those sub-scenarios as refinements. The aspects can be composed into the model either as sub-scenario replacements or in combination with the existing sub-scenarios. More research is necessary for developing algorithms for more complex compositions cases such as these.

4. RESOURCE CONTEXTS

The performance of a system emerges from the interaction between behaviour and resources. The performance of a given behaviour or operation can depend greatly on whether all of the needed resources are available at the time of execution. The *resource context* of an operation describes both the resources that are held as well as the order in which they have been acquired. This chapter explores the concept of resource contexts as well as how they relate to layered performance models.

4.1. NESTING OF RESOURCE CONTEXTS

A CSM Step or PathConnection has a *resource context* which is the ordered set of resources that are being held at that point in the scenario traversal. A resource is deemed to be *held* at a given point in a scenario if it has been acquired during a traversal and it has not yet been released. The set order is the order in which resources have been acquired. Resources are added to the resource context as they are acquired during a traversal and they are removed from the resource context as they are released. The resource context is thus an artifact of the scenario traversal and cannot be defined statically. The resource context is propagated between scenarios and sub-scenarios as the traversal progresses. A CSM with alternate or parallel branches which later join or merge should have the same resource context at the corresponding join or merge. If the resource context on incoming branches is not the same at a join, then that CSM does not have a "nested" resource context.

The resource context for a traversal can be stored as a stack of resource holding records specifying the resource and the number of resource units being held. In simple terms, whenever a *ResourceAcquire* is encountered during the CSM traversal, a resource

holding record (with the "`resourceName, resourceUnits`" format) is created and pushed on the resource context stack. Similarly, whenever a *ResourceRelease* is encountered the corresponding resource holding record for the resource is popped off the resource context stack. If the *ResourceAcquire* or *ResourceRelease* do not specify the resource units, then the default value for resource units is assumed to be 1. If additional units of a resource already being held are acquired, then a new resource holding record is created.

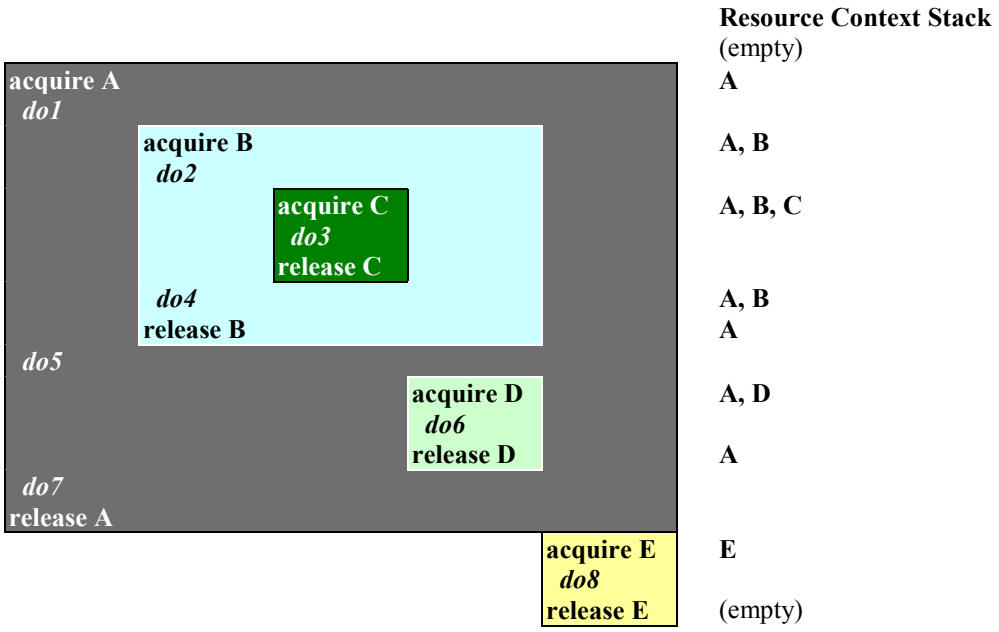


Fig 23 Nested resource context

The addition and removal of resource records in the resource context stack is straightforward when resources are held in a nested manner, meaning that resources are released in the reverse order to that in which they were acquired. Thus the resources being held prior to the acquisition of a given resource are still held after the release of that particular resource. For example, resources A, B, and C are held in a nested manner if resource A is acquired, then resource B is acquired, then resource C is acquired, then

resource C is released, then resource B is released, and finally resource A is released. Resources A, B, C are also held in an nested manner if resource A is acquired, then resource B is acquired, then resource B is released, then resource C is acquired, then resource C is released, and finally resource A is released. Fig 23 shows a nested resource context. Time proceeds downwards and each resource holding period is visualized as a shaded rectangle.

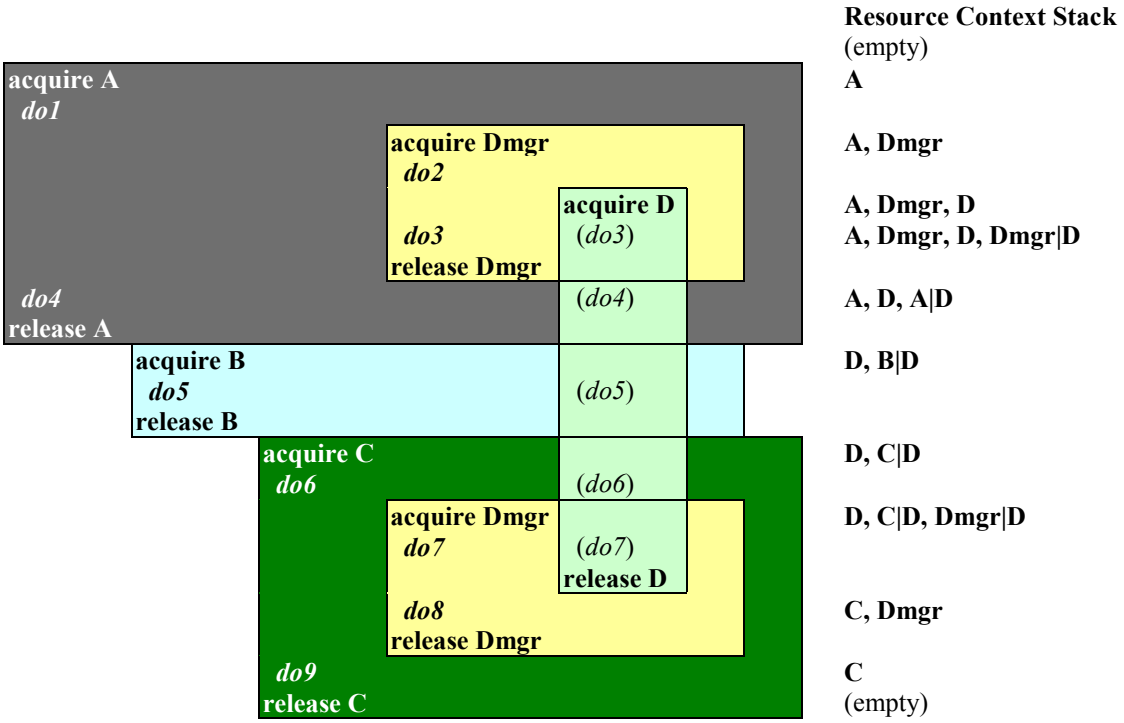


Fig 24 Resource context with non-nested resources

However, if resources are not acquired and released in a nested fashion then the holding of a given resource can occur in the context of different holdings of other resources. Non-nested resource holding contexts require a special naming notation for the resource context. For non-nested resource contexts the '|' operator is used to show resources that are held in the partial context of other resources. The '|' operator is used when an operation is performed by a component resource that is not the latest resource to have

been acquired. The component resource performing the operation is indicated to the left of the '|' and the resources acquired after that resource was acquired are listed to the right of the '|'. For example, X | Y indicates that the component resource X is operating within the context of resource Y.

Fig 24 shows a non-nested resource context where the resource holdings overlap as shown by the overlapping of the rectangles representing the resource holdings. Non-nested resource contexts require a different treatment in generating the LQN.

4.2. RESOURCE CONTEXT PROPAGATION

The propagation of resource contexts during a traversal uses the following basic semantics:

- traversals begin with an empty resource context
- *ResourceAcquire* steps add resources to the resource context
- *ResourceRelease* steps remove resources from the resource context
- *ResourcePass* steps modify the resource context depending on the interpretations described further down in this section
- Steps are associated with and executed by the Component resource that was last acquired; Steps cannot be executed by logical resources
- a sub-Scenario begins with the resource context of its parent Step
- a sub-Scenario ends with its own resource context which is propagated to the next element following its parent Step

The propagation of resource contexts along a linear scenario is straightforward: a traversal begins with an empty resource context; *ResourceAcquire*, *ResourceRelease*, and *ResourcePass* elements operate on the resource context; and each successive element inherits the resource context of the element before it. However, CSMs with sub-scenarios

or scenarios with forks/joins or branches/merges have more complicated resource propagation semantics.

The resource propagation interpretation for complex Steps with sub-scenarios is that the sub-scenario inherits the resource context of the containing parent Step. Resource operations in the sub-scenario operate on the inherited resource context and the modified resource context is propagated back to the element following the parent Step at the end of the sub-scenario. The parent Step thus has an entry resource context before the sub-scenario starts and propagates a different exit resource context after the sub-scenario is over.

The resource propagation interpretation at a Fork is that each parallel path segment shares the resource context of the Fork. For clarity about the resource context, this work assumes that resources along any particular forked path should be protected from operations on resources along another parallel path. Conditions on resource operations along a parallel path segment are thus defined in order to maintain a well-formed resource context at a subsequent Join.

The basic resource operation constraints for parallel path segments are that new resources used on a given parallel path segment need to be specifically acquired on that particular segment and that a previously held resource that is used on a given parallel path segment cannot be released on any different parallel path segment.

Using this interpretation of shared resource contexts along all parallel path segments means that the resource context at a Join holds the union of all the resources held at the preceding Fork as well as all the new resources acquired along any of the incoming parallel branches. A resolution strategy is needed to determine the order in which the new

resources are added to the resource context at the Join. Furthermore, there is also a need for resource context reconciliation after a Join if any resources have been released along any incoming parallel path segment.

The Join resource resolution strategy used in the CSM2LQN tool described in chapter 6 is to stop and warn the user if a resource is released along a parallel path segment without being explicitly identified with a ResourcePass after the Fork. If new resources are acquired along the parallel path segments before a Join then the default strategy for CSM2LQN is to add the new resources in the order in which the parallel path segments are traversed before the Join.

The resource context propagation interpretation at a Branch is that each alternate path inherits its own copy of the resource context of the Branch. Since alternate branches are mutually exclusive at execution time, the resource context is not shared among the different alternate branches and resource operations along any one branch are localized to that particular branch. There are no constraints on resource operations so there is a need to reconcile the resource context of different incoming branches whenever a Merge is encountered. Since for each Merge the incoming branches are mutually exclusive, the reconciliation process is non-trivial. Three possible automatic resource context reconciliation strategies at a Merge are:

- mark which incoming alternate branch is the main/default branch and propagate that branch's resource context after the Merge
- identify the set of resources common to all the incoming alternate branches and keep that as the resource context after the Merge
- aggregate all the resources from all the incoming alternate branches and keep that as the resource context after the Merge

All of these strategies have advantages and drawbacks. The default branch strategy has the advantage of reducing operations on the resource context since only the resource context for the default branch is kept and resource contexts for other branches are discarded. However, that means that the resource context after the Merge can be inconsistent with respect to the other incoming branches. The common resources strategy has the advantage of ensuring that all the resources available after the Join are definitely present regardless of which incoming branch was used. However, resources that are not held/acquired along all incoming branches are discarded. The aggregation of resources strategies has the advantage of not discarding any resource acquired along any of the incoming branches, but there is the danger of keeping resources that have not been acquired or have been released along some of the incoming branches.

Another possible strategy whenever the resource contexts at a Merge are not the same for each incoming alternate branch is to delay the merging and proceed as though each alternate branch extends through the remaining part of the scenario (i.e. the scenario path segments following the Merge is duplicated for each alternate branch until either the resource contexts along each branch become the same at a later Merge point or the scenario ends).

Additionally, there is also the strategy of detecting inconsistent resource contexts at *Joins* and *Merges* and warning the designer. The designer would then be required to specify the resource context that is to be propagated after the *Join* or *Merge* instead of relying on any automatic reconciliation strategy.

The Merge resource resolution strategy used in the CSM2LQN tool described in chapter 6 is to delay the merging and duplicate the path segments following the Join for each alternate branch.

The concept of *resource neutrality* is proposed to describe situations where resource context reconciliation is not needed. Resource neutrality refers to sub-scenarios, nested *Fork/Join* pairs, or nested *Branch/Merge* pairs where the resource context is the same before and after. A resource neutral sub-scenario has $ResourceContext_{(\text{before Start})}$ equal to $ResourceContext_{(\text{after End})}$. Similarly, a resource neutral *Fork/Join* pair has $ResourceContext_{(\text{at Fork})}$ equal to $ResourceContext_{(\text{at Join})}$. and a resource neutral *Branch/Merge* pair has $ResourceContext_{(\text{at Branch})}$ equal to $ResourceContext_{(\text{at Merge})}$. Resource context reconciliation is a proposed topic for further research with the goal of devising automated resource context reconciliation strategies that can transform badly formed CSMs into well-formed ones.

5. CSM PRE-PROCESSING

This chapter describes the algorithms for the series of transformation used to pre-process CSM input models in order to ensure they are well-formed and ready for LQN generation.

The CSM pre-processing consists of a series of CSM-to-CSM transformations that verify the composition of the input CSM and clean up CSM elements and in order to create an input CSM that can readily be transformed into an LQN. Some of the pre-processing transformations can be accomplished by inspection while others require a traversal of the CSM scenarios. The scenario traversal algorithm is explained first in section 5.1.

The first pre-processing step consists of a validation of the CSM syntax and semantics rules explained in chapter 3. The following CSM pre-processing steps employ CSM-to-CSM transformations to further clean up the CSM scenarios. Some of these CSM-to-CSM transformations require scenario traversals while others can be accomplished by direct examination of CSM elements. The algorithm for CSM validation is explained first, followed by the algorithm for scenario traversal, and finally the algorithms for the pre-processing transformations. The algorithms are presented as follows:

- CSM validation (section 5.1)
- CSM scenario traversal (section 5.2)
    - next scenario element (section 5.2.1)
    - previous scenario element (section 5.2.2)
    - scenario path traversal (section 5.2.3)
- CSM normalization (section 5.3)
    - create unique instances of all sub-scenarios (section 5.3.1)
    - assign components to all Steps (section 5.3.2)
    - check for non-nested component contexts (section 5.3.3)

    - remove duplicate ResourceAcquire and ResourceRelease elements in sub-scenarios (section 5.3.4)
    - flatten sub-scenarios (section 5.3.5)
    - assign path segments to sequential scenario fragments (section 5.3.6)
    - clean up forks/joins/branches/merges with empty path segments (section 5.3.7)
    - clean up CommSteps (section 5.3.8)
    - clean up resource acquisition and release (section 5.3.9)
- Interaction discovery (section 5.4)

Note: For the purpose of explaining algorithms, "StepType" refers to CSM elements of type Step, CommStep, ResourceAcquire, ResourceRelease, or ResourcePass. Similarly, "PathConnectionType" refers to CSM elements of type Start, End, Sequence, Fork, Join, Branch, or Merge.

5.1. CSM VALIDATION

The XML schema for CSM references the unique XML identifiers (XML data type 'IDRef') of CSM elements in order to store the associations between elements – as described in section 3.3.1 and shown in the class diagram in Fig 10. However not all XML parsers can verify the type of elements being referenced through IDRef identifiers and it is thus possible to create CSM files that can be parsed but are not meaningful. Therefore, the first CSM-to-CSM pre-processing step is a semantic check that validates the scenario elements.

PURPOSE

CSM Validation checks that the IDRef attributes of all CSM scenario elements conform to the multiplicity constraints in the CSM definition (syntax check) and to the scenario

constraints for well-formed CSMs (semantic check). The algorithm performs the validation checks listed in Table 4.

Success: This algorithm succeeds if all the syntax and semantic checks from Table 4 succeed.

Failure: This algorithm fails with an error message if any of the syntax or semantic checks fail. In case of failure, the entire transformation fails.

REQUIREMENTS

CSM Validation requires a CSM instance as input.

PROCESS

CSM Validation directly examines the attributes of all scenario elements in the input CSM. This algorithm does not use a scenario traversal.

Table 8. Algorithm for CSMValidation

Algorithm	**CSMValidation(CSM)**
Input	**CSM (CSMType)**
Output	**success (boolean)**
CSMValidation returns *true* if all the checks listed in Table 4 are successful. Returns *false* otherwise.	
Pseudocode	

```
//iterate through all the CSMElements in the input CSM
for all CSMElement[i] in CSM do
   if type of CSMElement[i] == Scenario then
      for all ScenarioElement[j] in CSMElement[i] do
         if type of ScenarioElement[j] == Start then
            if Scenario does not have a single Start point then
               return false
            if ScenarioElement[j].source != ( null | empty ) then
               return false
            if ScenarioElement[j].target != single IDRef of StepType then
               return false
         else if type of ScenarioElement[j] == End then
            if Scenario does not have at least 1 End point then
               return false
            if ScenarioElement[j].source != single IDRef of StepType then
               return false
            if ScenarioElement[j].target != ( null | empty ) then
               return false
```

Algorithm	**CSMValidation(CSM)**

```
        else if type of ScenarioElement[j] == Sequence then
           if ScenarioElement[j].source != single IDRef of StepType then
              return false
           if ScenarioElement[j].target != single IDRef of StepType then
              return false
        else if type of ScenarioElement[j] == ( Fork | Branch ) then
           if ScenarioElement[j].source != single IDRef of StepType then
              return false
           if ScenarioElement[j].target >= 2 IDRefs of StepType then
              return false
        else if type of ScenarioElement[j] == ( Join | Merge ) then
           if ScenarioElement[j].source >= 2 IDRefs of StepType then
              return false
           if ScenarioElement[j].target != single IDRef of StepType then
              return false
        else if type of ScenarioElement[j] == StepType then
           if ScenarioElement[j].source != single IDRef of PathConnection then
              return false
           if ScenarioElement[j].target != single IDRef of PathConnection then
              return false
        //endif - done checking the type of the ScenarioElement
     //endfor - done iterating through the ScenarioElements
   //endif - done checking the type of the CSMElement
//endfor - done iterating through CSMElements
return true
```

5.2. CSM SCENARIO TRAVERSAL

5.2.1. Next Scenario Element

PURPOSE

The traversal of CSM scenarios is done on an element-by-element basis for a given scenario. The NextScenarioElement algorithm is used to return the set of scenario elements which follow a given input scenario element.

Success: Returns the set of PathConnectionType elements that come after an input StepType element OR returns the set of StepType elements that come after an input PathConnectionType element.

Failure: There is no failure condition. Returns *null* if there are no following elements.

REQUIREMENTS

NextScenarioElement requires a StepType or PathConnectionType element as input.

PROCESS

The NextScenarioElement algorithm is described below.

Table 9. Algorithm for NextScenarioElement

Algorithm	**NextScenarioElement(CurrentElement)**
Input	**CurrentElement (StepType or PathConnectionType)**
Output	**NextElement (PathConnectionType or StepType)**
NextScenarioElement returns the collection of direct successors to the CurrentElement that is passed in. If the CurrentElement does not have a successor the algorithm returns *null*.	
Pseudocode	

```
//check the type of the current element
if type of CurrentElement == StepType then
   NextElement[] = CurrentElement.successor
else if type of CurrentElement == Start | Sequence | Join | Merge then
   NextElement[] = CurrentElement.target
else if type of CurrentElement == Fork | Branch then
   for all CurrentElement.target[i] do
   NextElement[i] = CurrentElement.target[i]
else if type of CurrentElement == End then
   NextElement[] = null
//endif - done checking the type of the current element
return NextElement[]
```

5.2.2. Previous Scenario Element

PURPOSE

The traversal of CSM scenarios is done on an element-by-element basis for a given scenario. The PreviousScenarioElement algorithm is used to return the set of scenario elements (as defined in 3.3.3) that come before a given input scenario element.

Success: Returns the set of PathConnectionType elements that come before an input StepType element OR returns the set of StepType elements that come before an input PathConnectionType element.

Failure: Returns *null* if there are no previous elements.

REQUIREMENTS

PrevScenarioElement requires a StepType or PathConnectionType element as input.

PROCESS

The PrevScenarioElement algorithm is described below.

Table 10. Algorithm for PrevScenarioElement

Algorithm	**PrevScenarioElement(*CurrentElement*)**
Input	**CurrentElement (*StepType* or *PathConnectionType*)**
Output	**PreviousElement (*PathConnectionType* or *StepType*)**
PreviousScenarioElement returns the collection of direct predecessors to the CurrentElement that is passed in. If the CurrentElement does not have a predecessor the algorithm returns *null*.	
Pseudocode	

```
//check the type of the current element
if type of CurrentElement == StepType then
   PreviousElement[] = CurrentElement.predecessor
else if type of CurrentElement == End | Sequence | Fork | Branch then
   PreviousElement[] = CurrentElement.source
else if type of CurrentElement == Join | Merge then
   for all CurrentElement.source[i] do
      PreviousElement[i] = CurrentElement.source[i]
else if type of CurrentElement == Start then
   PreviousElement[] = null
//endif - done checking the type of the current element
return PreviousElement
```

5.2.3. Scenario Path Traversal

PURPOSE

ScenarioPathTraversal uses the NextScenarioElement algorithm described above in order to get the next element along the scenario path. ScenarioPathTraversal is a recursive algorithm that calls itself until it reaches an End or a Join/Merge that has not yet been reached through all its incoming paths.

Success: Keeps recursively calling itself to advance to the next element along the scenario until it comes to an End element or a Join/Merge element that has not yet been reached through all its incoming paths.

Failure: There is no failure condition.

REQUIREMENTS

ScenarioPathTraversal requires a StepType or PathConnectionType element as input.

PROCESS

The ScenarioPathTraversal algorithm is described below.

Table 11. Algorithm for ScenarioPathTraversal

Algorithm	**ScenarioPathTraversal(CurrentElement)**
Input	**CurrentElement (StepType or PathConnectionType)**
Output	**n/a**
This recursive algorithm should be called with a Start element. It uses the NextScenarioElement algorithm to find each successive element along the scenario path and recursively calls itself until it reaches either an End element. It will only recursively call itself at a Join or Merge element if all of the Join or Merge's incoming paths have been traversed.	
Pseudocode	

```
//check if the current element has already been traversed
if CurrentElement→CurrentElementVisited == true then
   warning message that CurrentElment has already been traversed
   //stop traversal
```

Algorithm	**ScenarioPathTraversal(CurrentElement)**

```
if type of CurrentElement == StepType then
   //continue traversal
   NextElement = NextScenarioElement(CurrentElement)
   flag CurrentElementVisited = true
   ScenarioPathTraversal(NextElement)
else if type of CurrentElement == Start | Sequence then
   //continue traversal
   NextElement = NextScenarioElement(CurrentElement)
   flag CurrentElementVisited = true
   ScenarioPathTraversal(NextElement)
else if type of CurrentElement == Fork | Branch then
   //continue traversal for each next element in turn
   NextElement[] = NextScenarioElement(CurrentElement)
   for all NextElement[i] do
      flag CurrentElementVisited = true
      ScenarioPathTraversal(NextElement[i])
else if type of CurrentElement == Join | Merge then
   //continue traversal only AFTER all incoming paths have been traversed
   PreviousElement[] = PreviousScenarioElement(CurrentElement)
   for all PreviousElement[i] do
      check if PreviousElement was visited
   if all previous elements were visited then
      flag CurrentElementVisited = true
      NextElement[] = NextScenarioElement(CurrentElement)
else if type of CurrentElement == End then
   //stop traversal
//endif - done checking the type of the current element
```

5.3. CSM NORMALIZATION

CSMs can be generated from multiple sources and the resulting models can use different styles, conventions, and/or modeling patterns and may require additional transformations at the CSM level in order to normalize the models and provide the required input format for the generation of performance models. These transformations are based on the concept of equivalence between CSM constructs and include the cleaning up of CSMs so they are well-formed for generating performance models and the simplification of CSMs with empty elements or with long sequences of Steps.

The following equivalence rules for CSM scenarios are proposed as a framework for enabling CSM-to-CSM transformations. Some of them are used in the transformations of this chapter. There are four types of equivalence:

- **basic equivalence**: any two CSM scenarios are equivalent as long as they have the same workloads and they generate the same service demands on the underlying resources
- **order equivalence**: the scenarios have basic equivalence and also generate their service demands in the same order (more stringent than basic equivalence)
- **resource acquisition order equivalence**: two scenarios acquire resources in the same order
- **resource release order equivalence**: the two scenarios release the resources in the same order
- **resource context equivalence**: the scenarios both acquire and release the same resources in the same order.

The equivalence types apply to scenario fragments as well.

In addition, for fragments:

- **resource neutrality equivalence**: requires that the beginning and ending resource contexts for the fragments are the same
- **branch number equivalence**: applies to Branches or Forks with the same number of outgoing alternate branches or parallel branches
- **nested fragment equivalence**: makes a completely nested Fork-Join fragment or Branch-Merge fragment to be equivalent to a sequential fragment as long as they generate the same service demands

There are additional equivalence rules for Steps as well:

- **name equivalence**: allows Steps to be equivalent if they have the same name.
- **operational equivalence**: allows Steps to be substituted in they have the same resource demands
- **parametric equivalence**: Steps are substitutable if they have the same parameters

- **scenario fragment equivalence**: substitution rule for Steps which allows for the substitution of a Step for a scenario fragment without resource operations; the Step has the total resource demands of the Steps in the fragment

The basic equivalence rule for resources is that any two sets of resources are equivalent if they provide the same services.

5.3.1. Create Unique Instances of All Sub-Scenarios

PURPOSE

This algorithm creates copies of scenarios which are referenced in multiple sub-scenarios so that each sub-scenario references a unique scenario.

Success: All sub-scenarios are unique instances with unique elements IDs.

Failure: There is no failure condition.

REQUIREMENTS

This algorithm requires a CSM object as input.

PROCESS

The algorithm to create unique instances of all sub-scenarios directly examines the attributes of all Scenario elements in the input CSM. This algorithm does not use a scenario traversal.

Table 12. Algorithm for CreateUniqueSubScenarios

Algorithm	**CreateUniqueSubScenarios(** CSM **)**
Input	**CSM (***CSMType***)**
Output	**n/a**
This algorithm creates deep copies with unique element IDs for all scenarios referenced in multiple refinements.	
Pseudocode	

Algorithm	**CreateUniqueSubScenarios(** CSM **)**

```
//iterate through all the CSMElements in the input CSM
for all CSMElement[i] in CSM do
   if type of CSMElement[i] == Scenario then
      for all ScenarioElement[j] in CSMElement[i] do
         //check parent references to see if used in multiple sub-scenarios
         if (Scenario.parent != null) AND (Scenario.parent contains n>1 IDs)
         then
            for each sub-scenario refinement r do
               create a deep copy of the Scenario element and all contained
               element
               create unique IDs (append the ID of the parent Step to the IDs of
               each new scenario element)
               make sure that all references to the new scenario elements use the
               new IDs
               substitute the sub-scenario reference to the original Scenario
               with a reference to the new Scenario element
```

5.3.2. Assign Components to All StepType Elements

PURPOSE

This algorithm assigns components to all StepType scenario elements. This requires a scenario traversal in order to keep track of the current component and ensure that it is assigned to all Step, CommStep, ResourceAcquire, ResourceRelease and ResourcePass elements.

This algorithm is used three times in the CSM2LQN generation process:

- *before* flattening sub-scenarios (section 5.3.5) to make sure that all StepType elements have the proper components assigned
- *after* cleaning up resource acquisition and release based on messaging semantics (section 5.3.9) and *before* interaction discovery (section 5.4) to make sure that the clean-up process has not changed the component context

Success: The component attribute of every StepType element has a reference to the ID of the Component element in which it executes.

Failure: Generates an error message and returns *false* if the component attribute is already set and references a different component element than the one in which the StepType element executes.

REQUIREMENTS

This algorithm requires a Scenario object as input.

PROCESS

The algorithm uses the ScenarioPathTraversal algorithm (section 5.2.3) to trace the execution of the input Scenario element and to track the component context of that execution. It then updates the component attribute of the StepType elements as they are being traversed based on that component context. The algorithm is described below.

Table 13. Algorithm for AssignComponentsToSteps

Algorithm	**AssignComponentsToSteps(CurrentScenario)**
Input	**CurrentScenario (ScenarioType)**
Output	**n/a**

This algorithm sets the component attribute of StepType elements based on scenario traversal to discover of the component context for the execution of the StepType elements.

Pseudocode

```
//local variables to track component context
componentStack = NULL
acqComponent = NULL
relComponent = NULL
currentStep = NULL
```

```
//begin with the Start point of the input Scenario
currentElement = CurrentScenario→StartList[0]
```

```
//check the type of the current scenario element
if type of currentElement == ResourceAcquire then
   if type of resource being acquired == Component then
      acqComponent = currentElement.acquire->resource
      currentElement.component = acqComponent
      componentStack.Push( acqComponent )
```

```
else if type of currentElement == ResourcePass then
   if type of resource being passed == Component then
      currentElement.component = currentElement.pass->resource
```

Algorithm	**AssignComponentsToSteps(CurrentScenario)**

```
else if type of currentElement == ResourceRelease then
   if type of resource being released == Component then
      relComponent = currentElement.release->resource
      currentElement.component = relComponent
      componentStack.Pop()
else if type of currentElement == Step then
   if type of resource being acquired == Component then
      acqComponent = currentElement.acquire->resource
      currentElement.component = acqComponent
else if type of currentElement == CommStep then
   if type of resource being acquired == Component then
      acqComponent = currentElement.acquire->resource
      currentElement.component = acqComponent
//continue ScenarioPathTraversal
ScenarioPathTraversal( NextScenarioElement ( currentElement ) )
```

5.3.3. Check for Non-nested Component Contexts

PURPOSE

This algorithm checks for non-nested component contexts as described in section 4.1 and sets a flag if non-nestedeness is discovered.

This algorithm is used twice in the CSM2LQN generation process:

- *before* flattening sub-scenarios (section 5.3.5) to make sure that all Steps have the proper components assigned
- *after* cleaning up resource acquisition and release based on messaging semantics (section 5.3.9) and *before* interaction discovery (section 5.4) to make sure that the clean-up process has not changed the component context

Success: Sets a global flag denoting non-nested component context if non-nestedness is discovered.

Failure: There is no failure condition.

REQUIREMENTS

This algorithm requires a Scenario object as input.

PROCESS

The algorithm uses the ScenarioPathTraversal algorithm (section 5.2.3) to trace the execution of the input Scenario element and to track the component context of that execution. It then updates the component attribute of the StepType elements as they are being traversed based on that component context. The algorithm is described below.

Table 14. CheckNonNestedComponent

Algorithm	**CheckNonNestedComponent(CurrentScenario)**
Input	**CurrentScenario (ScenarioType)**
Output	**n/a**
This algorithm checks for non-nested component contexts and sets the NonNestedComponentFlag if non-nestedness is found.	
Pseudocode	

```
//global flag to denote non-nested component context
NonNestedComponentFlag = FALSE
//local variables to track component context
componentStack = NULL
acqComponent = NULL
relComponent = NULL
currentComponent = NULL
currentStep = NULL
```

```
//begin with the Start point of the input Scenario
currentElement = CurrentScenario→StartList[0]
```

```
//check the type of the current scenario element
if type of currentElement == ResourceAcquire then
   if type of resource being acquired == Component then
      acqComponent = currentElement.acquire->resource
      componentStack.Push( acqComponent )
```

Algorithm	**CheckNonNestedComponent(CurrentScenario)**

```
else if type of currentElement == ResourceRelease then
   if type of resource being released == Component then
      relComponent = currentElement.release->resource
      //check if component being released is on top of the component stack
      if relComponent == componentStack.Top() then
         //component is well-nested
         componentStack.Pop()
      else
         //component is not nested
         componentStack.RemoveComponent( relComponent )
         relComponent.nestedOnly = FALSE
         NonNestedComponentFlag = TRUE
//continue ScenarioPathTraversal
ScenarioPathTraversal( currentElement )
```

5.3.4. Remove Duplicate ResourceAcquire and ResourceRelease Elements in Sub-Scenarios

PURPOSE

The default resource context interpretation (that sub-scenarios inherit the resource context of their parent step) means that it is unnecessary for sub-scenarios to acquire the same component as the parent step at the beginning or release it at the end. In fact, an initial *ResourceAcquire* for the same component as that associated with the parent step is ambiguous since there is no way to know whether additional resource units for that component must be acquired or not. Similarly, a final *ResourceRelease* for the same component as the one associated with the parent step is ambiguous since there is no way to know if the component should really be released or not. The purpose of this algorithm is to remove those ambiguous *ResourceAcquire* and *ResourceRelease* elements in sub-scenarios. Future work could extend this algorithm to allow the user to control the removal of individual ResourceAcquire or ResourceRelease elements.

Success: If an initial ResourceAcquire for the same component as the component of the parent Step is found at the beginning of a sub-scenario and/or a final ResourceRelease for

the same component as the component of the parent Step is found at the end of a sub-scenario, then they are removed.

Failure: There is no failure condition.

REQUIREMENTS

This algorithm requires a Refinement object as input.

PROCESS

The algorithm uses NextScenarioElement (section 5.2.1) and PreviousScenarioElement (section 5.2.2) to check whether the initial ResourceAcquire and the final ResourceRelease in a sub-scenario act on the same component as the parent Step for the sub-scenario. If so, the ResourceAcquire and ResourceRelease are removed from the sub-scenario. The algorithm is described below.

NOTE

This algorithm should be used *before* flattening sub-scenarios in section 5.3.5.

Table 15. Algorithm for SubRemoveDuplicateResAcqResRel

Algorithm	**SubRemoveDuplicateResAcqResRel(Refinement)**
Input	**Refinement (RefinementType)**
Output	**n/a**
This algorithm finds and removes initial ResourceAcquire and final ResourceRelease elements from sub-scenarios if they acquire/release the same component as that of the parent Step for the sub-scenario.	
Pseudocode	

```
parentStep = Refinement->parentStep
subScenario = Refinement->subScenario
subStart = subScenario->StartList[0]
nextStep = NextScenarioElement( subStart )
nextPathConn = NextScenarioElement( nextStep )
subEnd = subScenario->EndList[0]
prevStep = PrevScenarioElement( subEnd )
prevPathConn = PrevScenarioElement( prevStep )
```

Algorithm	**SubRemoveDuplicateResAcqResRel(Refinement)**

```
//check the type of the next sub-scenario Step
if type of nextStep == ResourceAcquire then
   if type of nextStep->acquire == Component then
      if the nextStep.acquire == parentStep.component then
         //remove nextStep and nextPathConn from the sub-scenario
         subStart.target = nextPathConn.target
         NextScenarioElement( nextPathConn ).source = subStart.id
         delete nextStep
         delete nextPathConn
```

```
//check the type of the previous scenario element
if type of prevStep == ResourceRelease then
   if type of prevStep->release == Component then
      if prevElement.release == parentStep.component then
         //remove prevStep and prevPathConn from the sub-scenario
         subEnd.source = prevPathConn.source
         PrevScenarioElement( prevPathConn ).target = subEnd.id
         delete prevStep
         delete prevPathConn
```

5.3.5. Flatten Sub-Scenarios

PURPOSE

Going into and out of sub-scenarios complicates traversal and the generation of performance models. To avoid this, this algorithm "flattens" a sub-scenario by moving the sub-scenario's elements into the higher level scenario containing the sub-scenario's parent Step and connecting them in place of the parent Step.

Success: The sub-scenario elements after the sub-scenario Start point and before the sub-scenario End point are connected in the higher level scenario replacing the sub-scenario's parent Step. There are no sub-scenarios in the CSM.

Failure: There is no failure condition.

REQUIREMENTS

This algorithm requires a Step with a sub-scenario refinement as input. Assign Components to All StepType Elements (section 5.3.2) and Remove Duplicate

ResourceAcquire and ResourceRelease Elements in Sub-Scenarios (section 5.3.4) should be done before this algorithm.

PROCESS

The algorithm inspects all Steps with sub-scenario refinements and connects the target of the PathConnection leading to the parent Step to the first StepType element after the sub-scenario Start and connects the source of the PathConnection leading from the parent Step to the last StepType element before the sub-scenario End. The sub-scenario is thus "flattened" and takes the place of its containing parent Step.

In cases where there are multiple levels of sub-scenarios, this algorithm is invoked multiple times until all sub-scenarios are flattened into the top-level scenario. The algorithm is described below.

Table 16. Algorithm for SubFlatten

Algorithm	**SubFlatten(CSM)**
Input	**CSM (CSMType)**
Output	**n/a**
This algorithm connects sub-scenarios in place of their parent Steps.	
Pseudocode	

```
for all Scenario[i] in CSM.ScenarioList do
   for all Step[j] in Scenario.StepList do
      if Step[j] contains Refinement then
         //get parent step and associated path connections
         parentScenario = Scenario[i]
         parentStep = Step[j]
         parentPredecessor = parentStep.predecessor
         parentSuccessor = parentStep.successor
         //get sub-scenario and first and last steps
         subScenario = parentStep->Refinement->subScenario
         subFirstStep = subScenario->StartList[0]->target
         subLastStep = subScenario->EndList[0]->source
         //sub-scenario Start and End are no longer needed
         delete subScenario->Start
         delete subScenario->End
         //copy sub-scenario element into parent scenario
         for all ScenarioElements in subScenario do
            parentScenario.AddElement( ScenarioElement )
```

Algorithm	**SubFlatten(CSM)**

	`//connect sub-scenario elements in place of the parent step`
	`parentPredecessor.target = subFirstStep`
	`subFirstStep.predecessor = parentPredecessor`
	`parentSuccessor.source = subLastStep`
	`subLastStep.successor = parentSuccessor`

	`//parent step and sub-scenario objects are no longer needed`
	`delete parentStep`
	`delete subScenario`

5.3.6. Assign Path Segments to Sequential Scenario Fragments

PURPOSE

The ability to identify sequential path fragments in scenarios is required for detecting whether forks and branches have outputs without workload and to disentangle looping path structures. This algorithm adds a path segmentID attribute to all steps and path connections.

Success: All sequential path segments in a CSM have unique segmentIDs.

Failure: There is no failure condition.

REQUIREMENTS

The input scenario has been flattened (i.e. there are no sub-scenarios).

PROCESS

The input scenario is traversed using the ScenarioPathTraversal algorithm and all scenario elements are given a segmentID attribute. The segmentID is incremented following forks, joins, branches, and merges. The algorithm is described below.

Table 17. Algorithm for AssignPathSegmentIDs

Algorithm	**AssignPathSegmentIDs(CSM)**
Input	**CSM (CSMType)**
Output	**n/a**
This algorithm assigns a unique path segmentID to all ScenarioElements.	

Algorithm	**AssignPathSegmentIDs(CSM)**

Pseudocode

```
initialize segmentID variable

for all Scenario[i] in CSM.ScenarioList do

   //begin with the Start point of the current Scenario
   currentElement = CurrentScenario→StartList[0]

   if type of currentElement is Fork OR Branch then
      //set segment ID for Fork or Branch
      currentElement.SetAttributeWithValue( "segmentID", segmentID)
      //continue traversal for each outgoing target of the Fork or Branch
      for all currentElement.target[j] do
         increment segmentID
         ScenarioPathTraversal( currentElement.target[j] )

   else if type of currentElement is Join OR Merge then
      //only continue after traversing all incoming sources of the Join or Merge
      if all currentElement.source[j] have been traversed then
         increment segmentID
         //set segment ID for Join or Merge
         currentElement.SetAttributeWithValue( "segmentID", segmentID)
         ScenarioPathTraversal( NextScenarioElement( currentElement ) )

   else if type of currentElement is END then
      //set segment ID for End; traversal stops
      currentElement.SetAttributeWithValue( "segmentID", segmentID)

   else
      //currentElement is any other scenario element types
      //set segment ID and continue traversal
      currentElement.SetAttributeWithValue( "segmentID", segmentID)'
      ScenarioPathTraversal( NextScenarioElement( currentElement ) )
```

5.3.7. Clean Up Forks with Empty Path Segments

PURPOSE

Performance modelling involves the use of resources to do some work, therefore path segments without any resource interactions or associated workload are not meaningful for performance. This algorithm simplifies forks or branches that include path segments without any workload by removing those "empty" segments.

Success: Path segments without any workload that follow forks or branches are removed.

Failure: There is no failure condition.

REQUIREMENTS

The SubFlatten and AddPathSegmentIDs algorithms have been executed.

PROCESS

All path segments are traversed to see if they have any resource interactions (resource acquire/release/pass) or demands (steps with specified demands). If a path segment following a Fork does not have any resource interactions or demands and it ends with an End point then it is removed from the scenario. The algorithm is described below.

Table 18. Algorithm for CleanUpEmptyPathSegments

Algorithm	**CleanUpEmptyPathSegments(CSM)**
Input	**CSM (CSMType)**
Output	**n/a**
This algorithm identifies path segments without any demand following a Fork and removes them from the scenario.	
Pseudocode	

```
for all Scenario[i] in CSM.ScenarioList do
   for all Fork[j] in Scenario[i].ForkList do
   //check each outgoing path segment
   for all Fork[j].target[k] do
      segID = Fork[j]->target[k].segmentID
      segHasDemand = FALSE
      segHasResourceOp = FALSE
      //check that path segment finishes with an End point
      if path segment finishes with End element AND path segment does not have
         ( ResourceAcquire | ResourceRelease | ResourcePass ) then
         //check for path segment workload
         segmentHasWorkload = FALSE
         for all Steps in path segment do
            if Step.hostDemand != ( 0 OR "" ) then
               segmentHasWorkload = TRUE
            if Step has ExternalDemand then
               segmentHasWorkload = TRUE
         //remove segment if it does not have any workload
         if segmentHasWorkload == FALSE then
            delete currentFork.target[i]
            delete path segment
```

Algorithm	**CleanUpEmptyPathSegments(CSM)**

```
//if Fork has only one outgoing path segment left replace with Sequence
if currentFork.target[].size() == 1 then
   create new Sequence to replace Fork
   Sequence.source = currentFork.source
   Seqounce.target = currentFork.target
   delete currentFork
```

5.3.8. Clean Up CommSteps

PURPOSE

CommSteps have txComp and rcvComp attributes to denote the sending and receiving components for the message. This algorithm checks those attributes and ensures that they reference the correct components.

Success: The txComp and rcvComp attributes of all CommSteps reference the correct sending and receiving components. Warnings are generated if inconsistencies are found.

Failure: There is no failure condition.

REQUIREMENTS

Component have been assigned to all StepTypes in the Scenario.

PROCESS

Examine all CommSteps in the scenario and verify that the txComp attribute is the same as the component of the previous StepType and the rcvComp attribute is the same as the component of the next StepType. The algorithm is described below.

Table 19. Algorithm for CleanUpCommSteps

Algorithm	**CleanUpCommSteps(CSM)**
Input	**CSM (CSMType)**
Output	**warning message if inconsistent txComp or rcvComp attributes are found**
This algorithm verifies that txComp and rcvComp attributes of all CommSteps reference the correct sending and receiving Components.	
Pseudocode	

Algorithm	**CleanUpCommSteps(CSM)**

```
for all Scenario[i] in CSM.ScenarioList do
   for all CommStep[j] in Scenario[i].CommStepList do
      prevPathConn = PrevScenarioElement( CommStep[j] )
      prevStep = PrevScenarioElement( prevPathConn )
      nextPathConn = NextScenarioElement( CommStep[j] )
      nextStep = NextScenarioElement( nextPathConn )
      if CommStep[j].txComp == ( NULL || empty ) then
         CommStep[j].txComp = prevStep.component
      else if CommStep[j].txComp != prevStep.component then
         show warning message
         CommStep[j].txComp = prevStep.component
      if CommStep[j].rcvComp == ( NULL || empty ) then
         CommStep[j].rcvComp = nextStep.component
      else if CommStep[j].rcvComp != nextStep.component then
         show warning message
         CommStep[j].rcvComp = nextStep.component
```

5.3.9. Clean Up Resource Acquisition and Release

PURPOSE

It is possible to specify extraneous ResourceAcquire and ResourceRelease elements when scenarios have synchronous/blocking interactions. A synchronous call means that the calling component is held while waiting for the reply. Therefore it is incorrect to release a calling component after a synchronous message, or to reacquire it before the reply is received.

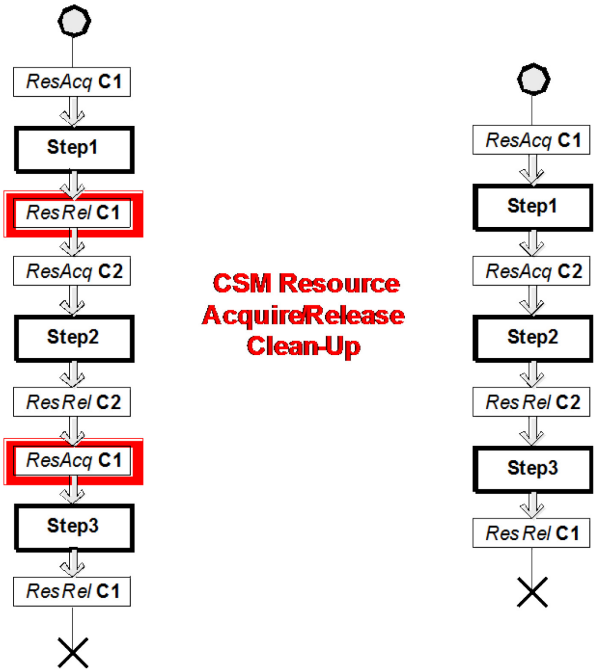


Fig 25 CSM with extraneous ResourceAcquire and ResourceRelease on the left and cleaned-up CSM on the right

This algorithm removes extraneous ResourceReleases of the calling component when making synchronous calls as well as extraneous ResourceAcquires of the calling component when receiving the replies to those calls. Fig 25 shows an example of what this algorithm does where a synchronous call is made from C1 to C2 but C1 is incorrectly released when making the call and then reacquired when receiving the reply. This is incorrect and potentially misleading because C1 is actually blocked for the duration of the call to C2 and thus is not really released and re-acquired. Future work could extend this algorithm to allow the user to control the removal of individual ResourceAcquire or ResourceRelease elements.

Success: Unnecessary ResourceAcquires and ResourceReleases when making synchronous calls are removed from the scenario.

Failure: There is no failure condition.

REQUIREMENTS

CleanUpCommSteps has been executed.

PROCESS

The algorithm examines all the CommSteps in a Scenario. It removes ResourceReleases of the calling component before CommSteps making synchronous calls and it removes ResourceAcquires of the receiving component after CommSteps for replies to synchronous calls. The algorithm is described below.

Table 20. Algorithm for CleanUpCommResAcqResRel

Algorithm	**CleanUpCommResAcqResRel(CSM)**
Input	**CSM (CSMType)**
Output	**n/a**
This algorithm removes unnecessary ResourceReleases of the calling Component before CommSteps making synchronous calls and removes unnecessary ResourceAcquires of the receiving Component after CommSteps making replies to synchronous calls	
Pseudocode	

```
for all Scenario[i] in CSM.ScenarioList do
   for all CommStep[j] in Scenario[i].CommStepList do
      if CommStep[j].msgKind == sync do
         prevPathConn = PrevScenarioElement( CommStep[j] )
         prevStep = PrevScenarioElement( prevPathConn )
         if type of prevStep == ResourceRelease then
            if prevStep.release == CommStep[j].txComp then
               //disconnect ResourceRelease from scenario path and delete
               CommStep[j].predecessor = prevStep.predecessor
               prevPathConn2 = PrevScenarioElement( prevStep )
               prevPathConn2.target = prevPathConn.target
               delete prevPathConn
               delete prevStep
      else if CommStep[j].msgKind == reply do
         nextPathConn = NextScenarioElement( CommStep[j] )
         nextStep = NextScenarioElement( nextPathConn )
         if type of nextStep == ResourceAcquire then
            //check if releasing the same component making the synchronous call
            if nextStep.acquire == CommStep[j].rcvComp then
               //disconnect ResourceAcquire from scenario path and delete
               CommStep[j].successor = nextStep.successor
               nextPathConn2 = NextScenarioElement( nextStepType )
               nextPathConn2.source = nextPathConn.source
               delete nextPathConn
               delete nextStep
```

5.4. INTERACTION DISCOVERY

PURPOSE

Synchronous interactions reduce performance by reducing concurrency and LQNs detect this effect if it is present. Therefore there is a benefit in detecting synchronous interactions even if they are only implied in a scenario, if only to bring them to the attention of the designer. Some other scenario analyses (such as those used in [112]) only determine the total device demands by different classes of requests and cannot describe the interactions of software components. Other models which do retain the software component context of demand, such as Kahkipuro's AQN [58], require that blocking interactions be explicitly identified.

The detection of blocking interactions is important because they are not always obvious in a system but they have an important performance impact. The performance model generating algorithms described here take a conservative approach to performance modeling and assume that whenever a scenario path returns to a previously visited component, that component was waiting for the return and was blocked during the wait. This maximizes the interpretation of blocking interactions and yields models that capture all the possible blocking points in a system.

The scenario analysis described here maximizes the synchronous interpretation of calls - wherever asynchronous message exchanges indicate that a task may be waiting for a reply, the resulting interaction is interpreted as a blocking synchronous interaction when generating the performance model. This approach reveals synchronous calling patterns to the designer even if they are disguised as sequences of asynchronous messages. This aspect of the research leading to the present work was introduced in [92]. Additional

blocking may also be introduced by the environment, such as platform-dependent artifact like an ORB, in behaviour that is not described in the CSM. To properly handle this additional blocking, more information is needed, for example from completions of the model such as those proposed in [124].

Success: Emerging synchronous interactions that use asynchronous messages are detected and the msgKind for the CommSteps involved are adjusted to reflect the interaction.

Failure: There is no failure condition.

REQUIREMENTS / PRECONDITIONS

The following are required preconditions for InteractionDiscovery:

- the CSM has been validated – CSMValidation has been executed and returned TRUE (section 5.1)
- there are unique instances of all sub-Scenarios – CreateUniqueSubScenarios has been executed (section 5.3.1)
- the component attributes are set for all Steps – AssignComponentsToSteps has been executed (section 5.3.2)
- the Component resource context is nested – CheckNonNestedComponent has been executed and the NonNestedComponentFlag is set to FALSE (section 5.3.3)
- removed duplicate ResourceAcquire and ResourceRelease in sub-scenarios – SubRemoveDuplicateResAcqResRel has been executed (section 5.3.4)
- all sub-Scenarios have been flattened – SubFlatten has been executed (section 5.3.5)
- the txComp and rcvComp attributes are set for all CommSteps – CleanUpCommSteps has been executed (section 5.3.8)

The following are *not* required preconditions before executing InteractionDiscovery:

- path segmentIDs have all been set – AssignPathSegmentIDs has been executed (section 5.3.6)

- Forks with empty path segments have been cleaned up – CleanUpEmptyPathSegments has been executed (section 5.3.7)
- unnecessary ResourceAcquires and ResourceReleases have been cleaned up – CleanUpCommResAcqResRel has been executed (section 5.3.9)

PROCESS

The interaction discovery uses the ScenarioPathTraversal algorithm to traverse the scenario and build a component interaction stack (similar to the component stack used for discovering component context) and detect whether there are synchronous interactions using asynchronous messages. If such interactions are discovered then the message types are changed from async to sync, reply, or forwarding as discovered.

An additional clean-up of resource acquisition and release (section 5.3.9) is optional but can be useful after interaction discovery in order to account for previously unknown blocking interactions. The algorithm for interaction discovery is described below.

Table 21. Algorithm for InteractionDiscovery

Algorithm	**InteractionDiscovery(CurrentScenario)**
Input	**CurrentScenario (ScenarioType)**
Output	**success (Boolean)**
This algorithm traverses CSM scenarios and discovers synchronous interactions based on the order in which messages are exchanged between components. It returns TRUE if the discovery did not encounter any problems and returns FALSE otherwise.	
Pseudocode	

```
while( traverse CSM Scenario ) do
   currentComponent = currentStep->Component
   nextComponent = nextStep->Component
   if( currentComponent != NULL && nextComponent != NULL) then
      if( currentComponent == nextComponent) then
         //there is no interaction
      else
         //currentStep and nextStep have different Components
```

Algorithm	**InteractionDiscovery(CurrentScenario)**

```
        if( nextComponent is NOT on interactionStack ) then
           //request being sent to nextComponent
           //push currentComponent on interactionStack for initiating
           // interaction
           interactionStack.Push( currentComponent)
           continue traverse CSM Scenario Steps
        else
           //nextComponent is on interactionStack
           //get component from top of interactionStack
           topComponent = interactionStack->Top
           if( topComponent == nextComponent ) then
              //currentComponent sent a reply to currentComponent for a previous
              // request
              mark interaction from currentComponent to nextComponent as REPLY
              //remove nextComponent from interactionStack
              interactionStack.Pop()
              //nextComponent sent a synchronous request to currentComponent
              mark pending interaction from nextComponent to currentComponent as
              SYNC
              continue traverse CSM Scenario Steps
           else
              //nextComponent is NOT the top of the interactionStack
              //currentComponent sent a forwarded reply to nextComponent for a
              // previous request
              mark interaction from currentComponent to nextComponent as REPLY
              //pop component from top of interactionStack
              previousComponent = topComponent
              topComponent = interactionStack->Pop()
              while( topComponent != nextComponent ) do
                 //previousComponent sent a forwarding request to topComponent
                 mark pending interaction from previousComponent to topComponent
                 as FWD
                 //remember current topComponent
                 previousComponent = topComponent
                 //pop component from top of interactionStack
                 topComponent = interactionStack->Pop()
              //nextComponent sent a synchronous request to previousComponent
              mark pending interaction from nextComponent to previousComponent
              as SYNC
              continue traverse CSM Scenario Steps
//traversal is finished
if( interactionStack->Size != 0 ) then
   //there are asynchronous pending interactions
   lastComponent = lastStep->Component
   topComponent = interactionStack->Top
   while( interactionStack->Size != 0 ) do
      //topComponent sent an asynchronous request to lastComponent
      mark pending interaction from topComponent to lastComponent as ASYNC
      //remember current topComponent
      lastComponent = topComponent
      //pop component from top of interactionStack
      topComponent = interactionStack->Pop()
```

6. GENERATING PERFORMANCE MODELS FROM CSM

The generation of performance models involves a mapping of the CSM resources and operations to the corresponding performance model elements together with the relationships and interactions between the operations and resources.

This chapter describes the interpretation of service requests used in the generation of performance models, the correspondences between CSM elements and performance model elements, the generation of QN and LQN models from CSM, and the algorithm for returning the next path element when traversing a CSM scenario.

It is assumed that only well-formed CSMs, as described in section 3.4.3 and chapter 5 are used as input for the generation of performance models.

6.1. ALGORITHM OVERVIEW FOR GENERATING PERFORMANCE MODELS FROM CSM

The following sections describe the algorithms used to generate Layered Queueing Network (LQN) and Queueing Network (QN) performance models from CSM. The process is also described as part of the S2P algorithm described in [93] [94] [127] [139].

- Generate the target model processors and tasks corresponding to the resources in the CSM.
- Generate the target model activities and entries corresponding to the behaviour in the CSM.
- Determine the connections between the target model behaviour elements, using the CSM scenario traversal uses the algorithms presented in section 5.2.

The high-level algorithm to generate LQN performance models from CSM is detailed in Table 22 below. The high-level algorithm to generate QN performance models from CSM follows in Table 23.

Table 22. High-level algorithm for GenerateLQNPerfModel

Algorithm	**GenerateLQNPerfModel(CSM)**
Input	**CSM (*CSMType*)**
Output	**LQN (*LQNModel*)**
This algorithm creates a LQN performance model from the input CSM.	
Pseudocode	

```
//validate and normalize CSM
CSMValidation( CSM )
CreateUniqueSubScenarios( CSM )
```

```
for all Scenario[i] in CSM.ScenarioList do
   AssignComponentsToSteps( Scenario[i] )
```

```
flagNonNestedComponentContext = FALSE
for all Scenario[i] in CSM.ScenarioList do
   flagNonNestedComponentContext = CheckNonNestedComponent( Scenario[i] )
```

```
for all Scenario[i] in CSM.ScenarioList do
   for all Step[j] in Scenario[i].StepList do
      if Step[j] contains Refinement then
         SubRemoveDuplicateResAcqResRel( Step[j]->Refinement )
```

```
SubFlatten( CSM )
```

```
//optional CSM normalization
if option is enabled then
   AssignPathSegmentIDs( CSM )
   CleanUpEmptyPathSegments( CSM )
```

```
CleanUpCommSteps( CSM )
```

```
//optional CSM normalization
if option is enabled then
   CleanUpCommResAcqResRel( CSM )
```

```
for all Scenario[i] in CSM.ScenarioList do
   InteractionDiscovery( Scenario[i] )
```

```
//generate LQN
lqn = GenerateLQN( CSM )
return lqn
```

Table 23. High-level algorithm for GenerateQNPerfModel

Algorithm	**GenerateQNPerfModel(CSM)**
Input	**CSM (*CSMType*)**
Output	**QN parameters**
This algorithm generates the parameters of a QN performance model based on the input CSM.	
Pseudocode	

```
CSMValidation( CSM )
```

```
qnParams = GenerateQN( CSM )
return qnParams
```

6.2. CSM AND PERFORMANCE MODEL CORRESPONDENCES

There are direct correspondences between some CSM elements and LQN and QN elements in the target performance models. Table 24 lists those CSM elements and the corresponding LQN and QN elements.

Table 24. Correspondences between CSM elements and LQN and QN performance target model elements

CSM Element	LQN Element	QN Element
ProcessingResource	Processor	Processor
Component	Task	-
Non-nested Component	Task with semaphore type	-
PassiveResource	Task with semaphore type	-
ExternalOperation	Task with dedicated Processor	Processor
Start	Reference Task with dedicated Processor	(workload class)
Step	Activity	(service demand)
CommStep with asynchronous message	Activity to make call Entry to receive call	-
CommStep with synchronous message	Activity to make call Entry to receive call	-
CommStep with reply message	Activity to send reply	-
CommStep with forwarding message	Activity to send reply Entry to forward call Entry to receive call	-
Fork	AND-Fork activity connection	-
Join	AND-Join activity connection	-
Branch	OR-Fork activity connection	(change in visit ratio)
Merge	OR-Join activity connection	(change in visit ratio)

A CSM Component may represent an operating system process, an object or a module. Given that any synchronous call, whether to a process, object or module block the calling task, it is equivalent to a call to a task in the performance model. As such, CSM Component corresponds to a LQN task. There is no QN representation for a Component, but the host of a component is used to identify the QN Processor that satisfies the service

demands of the operations executing within the context of that component. A LQN Reference Task can serve either as an open workload generator inserting asynchronous requests, or a closed workload generator, in which case it has a multiplicity equal to the population, and each task makes synchronous requests (and waits for the response).

CSM Steps correspond to LQN Activities and generate the service demands that are aggregated by processor in the QN.

CSM CommSteps which make a call (asynchronous, synchronous, or forwarding) map to LQN Activities to make the call and LQN Entries to receive the call. The calls are thus represented by these activity-entry pairs. CommSteps which send a reply correspond to reply activities in the LQN. Fig 26 illustrates the correspondences between different types of calls in CSM and LQN:

- Fig 26a shows a CSM scenario which acquires and releases a component A and then acquires and releases a component B. This corresponds to a LQN where the task corresponding to A makes an asynchronous call to the task corresponding to B.
- Fig 26b shows a CSM scenario which acquires a component A, then acquires and releases a component B, and then A. This corresponds to a LQN where the task corresponding to A makes a synchronous call to the task corresponding to B.
- Fig 26c shows a CSM scenario which acquires a component A, then acquires and releases a component B, then acquires and releases a component C, and then releases A. This corresponds to a LQN where the task corresponding to A makes a synchronous call to the task corresponding to B. The task corresponding to B then makes a forwarding call to the task corresponding to C (as described in Section 2.4). The task corresponding to C is then the one replying to A.
- Fig 26d shows a CSM scenario which acquires a component A, then acquires and releases a component B, then returns back to component A to execute some steps, then acquires and releases a component C, and finally releases A. This corresponds to a LQN where task A synchronously calls task B and then task C in sequence.

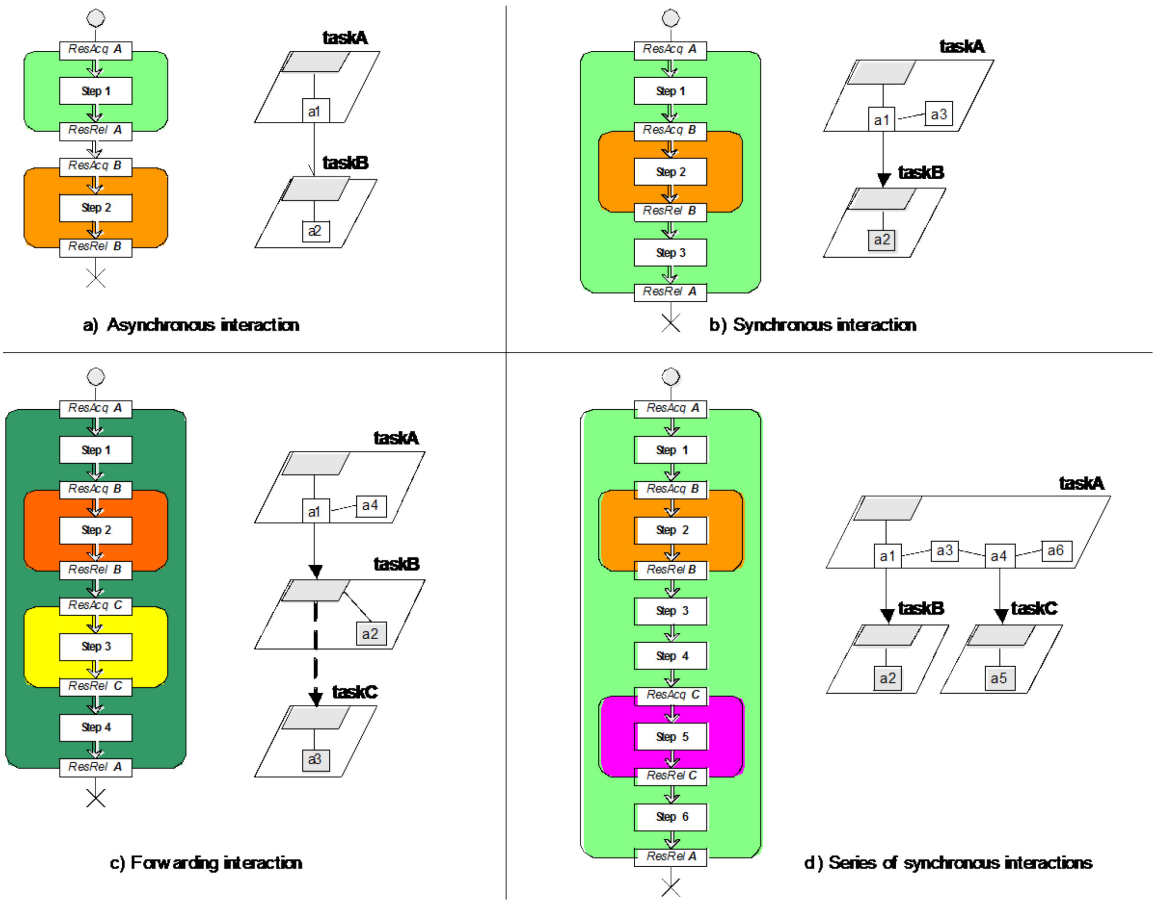


Fig 26 Interactions with different types of calls, CSM on the left and corresponding LQN on the right

CSM Forks, Joins, Branches and Merges correspond to LQN Activity connections. Branches and Merges change the visit ratios for the subsequent steps when generating aggregate service demands in the QN. CSM scenario path fragments contained within a component correspond directly to LQN activities. The LQN activity sequence notation has the usual constructs of alternate or parallel branching and joining, as well as looping. Fig 27 shows corresponding CSM and LQN scenario path notations and elements.

- Fig 27a shows a CSM scenario with parallel branches and the resulting LQN activities
  - the LQN 'AND' activity connectors are labeled with a '&'

- Fig 27b shows a CSM scenario with alternate branches and the resulting LQN activities
  - the LQN 'OR' activity connectors are labeled with a '+'

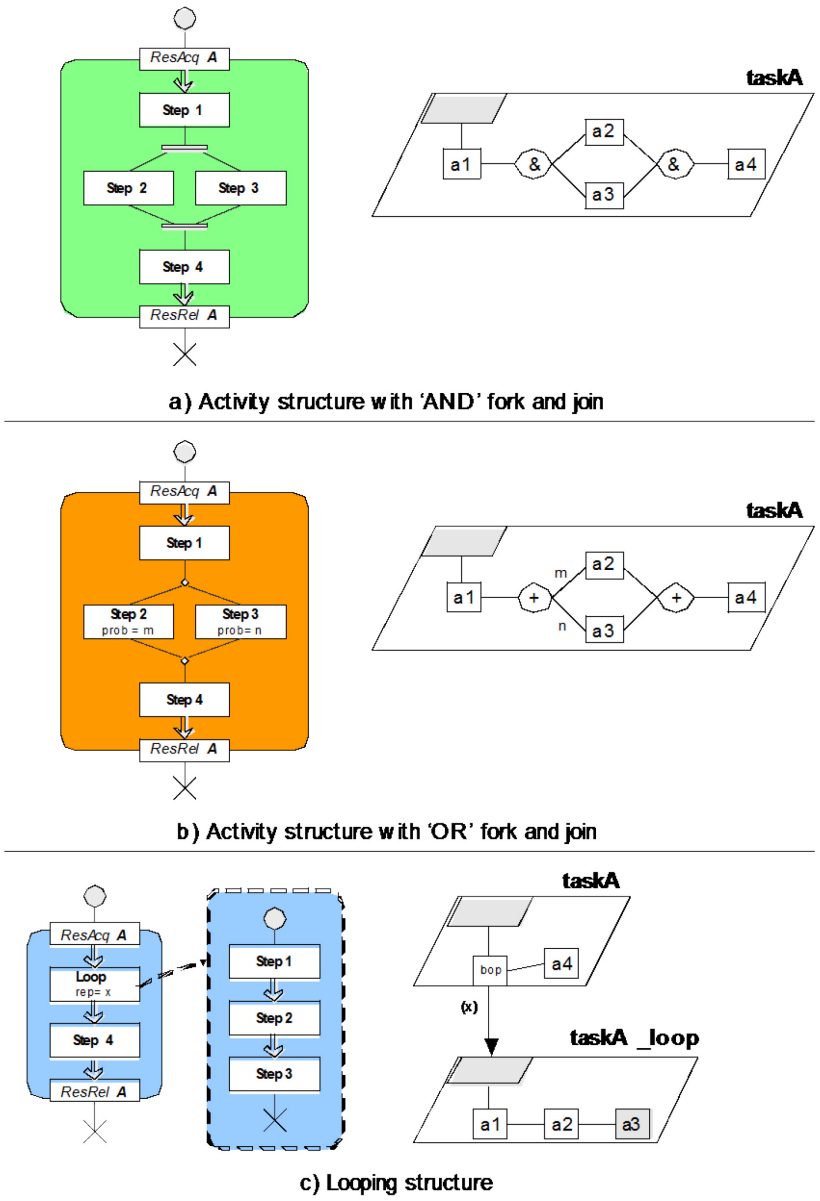


Fig 27 CSM and the corresponding LQN scenario path structures

- Fig 27c shows a CSM scenario with a repeated step containing a sub-scenario and the resulting LQN activities

- the Step repetition count is shown as a repetition multiplier on the corresponding activity in the LQN
- a more complex loop body represented in CSM as a sub-scenario inside a repeated step is represented in LQN as a pseudo-task which is called repeatedly by the loop control activity corresponding to the repeated step. The pseudo-task has infinite multiplicity (so it represents fully re-entrant behaviour, equivalent to a procedure of the calling entry) and is hosted by the same processor.
- the CSM sub-scenario is represented by the LQN activities contained in the pseudo-task

CSM ExternalOperations represent calls to a service outside the scope of the given CSM, such as a file service or a database service. Ultimately a submodel for this subsystem must be added to the LQN performance model, but as a placeholder, a task with a dedicated processor is inserted to take the calls for the service.

6.3. LQN GENERATION

PURPOSE

This algorithm creates and connects the LQN elements corresponding to the input CSM.

Success: The generate LQN is provided as an output.

Failure: There is no failure condition.

REQUIREMENTS

All the CSM normalization and interaction discovery steps described in chapter 5 have been done. In addition, all CSM elements have valid performance annotations, as mentioned in section 3.3. Otherwise, default values are assigned as shown in Table 25 which lists all the annotations used when generating LQN performance models as well as

the required default values assigned should they be left out. These values are required so that every CSM generates a solvable LQN performance model.

Table 25. CSM performance annotations and default values needed to create meaningful LQN performance models

CSM Element	Performance Annotation	Default Value for LQN
Step	hostDemand	0
	ExternalDemand (optional)	0
	repCount (optional)	1.0
	probability (optional)	1.0
ProcessingResource	opTime	1.0
	multiplicity (optional)	1
Component	host	default/infinite processor
	multiplicity (optional)	1
Branch	probability attribute of the first Step on an outgoing branch	equal probability if not specified
OpenWorkload	arrivalPattern	poissonPDF
	arrivalParam1 (denotes the rate of arrivals per second)	1.0
ClosedWorkload	population	1.0
	arrivalParam1 (denotes the think time in seconds)	1.0

PROCESS

This algorithm first creates the LQN elements corresponding to the elements in the input CSM as described in Table 24, plus a special infinite processor to act as a host for any task that does not have a defined deployment. The ScenarioPathTraversal algorithm is then used to create the activity connections for all the generated activities. The algorithm is described below.

Table 26. Algorithm for GenerateLQN

Algorithm	**GenerateLQN(CSM)**
Input	**CSM (CSMType)**
Output	**warning message if inconsistencies are found**
This algorithm generates LQN elements corresponding to the input CSM by inspection. ScenarioPathTraversal is then used to traverse the CSM scenarios and generate the LQN activity connections.	
Pseudocode	

Algorithm	**GenerateLQN(CSM)**

```
create lqnModel
create lqnInfiniteProcessor
for all ProcessingResources in CSM do
   create lqnProcessor with same name, opTime and multiplicity as
      ProcessingResource
   add lqnProcessor to lqnModel
```

```
for all Components in CSM do
   create lqnTask with same multiplicity as Component
   if Component is non-nested then
      lqnTask.type = semaphore
```

```
   if Component.host is NULL or empty then
      assign lqnTask to lqnInfiniteProcessor
   else
      assign lqnTask to lqnModel.getProcessor( Component→host.getName() )
```

```
   add lqnTask to lqnModel
```

```
for all PassiveResources in CSM do
   create lqnTask with same multiplicity as PassiveResource
   lqnTask.type = semaphore
   assign lqnTask to lqnInfiniteProcessor
   add lqnTask to lqnModel
```

```
for all Scenarios in CSM do
   for all CommSteps in Scenario do
      //check that this is not an initial CommStep which is handled with Start
      if CommStep→predecessor != Start then
         //get the LQN tasks for the sending and receiving CSM components
         lqnSendTask = lqnModel.getTask( CommStep→txComp.getName() )
         lqnRcvTask = lqnModel.getTask( CommStep→rcvComp.getName() )
```

```
         if CommStep.msgKind == async then
            //create LQN entry and first activity to receive the call
            create lqnEntry
            create lqnFirstActivity
            set lqnFirstActivity as first activity of lqnEntry
            add lqnEntry to lqnRcvTask
            add lqnFirstActivity to lqnRcvTask
            //create LQN activity to send the call
            create lqnCallActivity
            set lqnCallActivity to make async call to lqnEntry
            add lqnCallActivity to lqnSendTask
```

```
         else if CommStep.msgKind == sync then
            //create LQN entry and first activity to receive the call
            create lqnEntry
            create lqnFirstActivity
            set lqnFirstActivity as first activity of lqnEntry
            add lqnEntry to lqnRcvTask
            add lqnFirstActivity to lqnRcvTask
            //create LQN activity to send the call
            create lqnCallActivity
            set lqnCallActivity to make sync call to lqnEntry
            add lqnCallActivity to lqnSendTask
```

Algorithm	**GenerateLQN(CSM)**

```
      else if CommStep.msgKind == fwd then
        //create LQN entry and first activity to receive the call
        create lqnEntry
        create lqnFirstActivity
        set lqnFirstActivity as first activity of lqnEntry
        add lqnEntry to lqnRcvTask
        add lqnFirstActivity to lqnRcvTask
        //create LQN activity to send the call
        create lqnCallActivity
        set lqnCallActivity to make fwd call to lqnEntry
        add lqnCallActivity to lqnSendTask
      else if CommStep.msgKind == reply then
        //create LQN activity to make the reply
        create lqnReplyActivity
        set lqnReplyActivity to be a reply activity
        add lqnReplyActivity to lqnSendTask
  for all Steps in Scenario do
    //get the LQN tasks for the step's component
    lqnStepTask = lqnModel.getTask( Step→component.getName() )
    //create LQN activity for step
    create lqnStepActivity
    add lqnStepActivity to lqnStepTask
    if Step.hostDemand is not NULL or empty then
      set lqnStepActivity to have the same host demand as the Step
    else
      set lqnStepActivity to have default host demand
    if Step.probability is not NULL or empty then
      set lqnStepActivity to have the same probability as the Step
    else
      set lqnStepActivity to have default probability
    if Step.repCount is not NULL or empty then
      set lqnStepActivity to have the same repetition count as the Step
    else
      set lqnStepActivity to have default repetition count
  for all Forks in Scenario do
    //get the LQN tasks for the fork source's component
    lqnForkTask = lqnModel.getTask( Fork→source→component.getName() )
    //create LQN activities
    create lqnForkSourceActivity
    add lqnForkSourceActivity to lqnForkTask
    for all Fork.targets do
      create lqnForkTargetActivity
      add lqnForkTargetActivity to lqnForkTask
      create AndForkConnection for lqnForkSourceActivity and
        lqnForkTargetActivity
  for all Joins in Scenario do
    //get the LQN tasks for the join target's component
    lqnJoinTask = lqnModel.getTask( Join→target→component.getName() )
    //create LQN activities
    create lqnJoinTargetActivity
    add lqnJoinTargetActivity to lqnJoinTask
```

Algorithm	**GenerateLQN(CSM)**

```
      for all Join.sources do
         create lqnJoinSourceActivity
         add lqnJoinSourceActivity to lqnJoinTask
         create AndJoinConnection for lqnJoinSourceActivity and
            lqnJoinTargetActivity
   for all Branches in Scenario do
      //get the LQN tasks for the branch source's component
      lqnBranchTask = lqnModel.getTask( Branch→source→component.getName() )
      //create LQN activities
      create lqnBranchSourceActivity
      add lqnBranchSourceActivity to lqnBranchTask
      for all Branch.targets do
         create lqnBranchTargetActivity
         add lqnBranchTargetActivity to lqnBranchTask
         create OrForkConnection for lqnBranchSourceActivity and
            lqnBranchTargetActivity
   for all Merges in Scenario do
      //get the LQN tasks for the join target's component
      lqnMergeTask = lqnModel.getTask( Merge→target→component.getName() )
      //create LQN activities
      create lqnMergeTargetActivity
      add lqnMergeTargetActivity to lqnMergeTask
      for all Merge.sources do
         create lqnMergeSourceActivity
         add lqnMergeSourceActivity to lqnMergeTask
         create AndMergeConnection for lqnMergeSourceActivity and
            lqnMergeTargetActivity
   for all Start points in Scenario do
      if Start has a Workload then
         //create a reference task
         create lqnRefTask
         assign lqnRefTask to lqnInfiniteProcessor
         add lqnRefTask to lqnModel
         //create a reference entry and prepare to call the first task
         create lqnRefEntry
         add lqnRefEntry to lqnRefTask
         //get the first task after the reference task
         if Start→target is a Step then
            lqnFirstTask = lqnModel.getTask(Start→target→component.getName())
            //an activity corresponding to this step already exists
            lqnFirstActivity = activity corresponding to Step
```

Algorithm	**GenerateLQN(CSM)**

```
else
   if Start→target is a CommStep then
      lqnFirstTask = lqnModel.getTask(Start→target→rcvComp.getName())
   else if Start→target is a ResourceAcquire then
      lqnFirstTask = lqnModel.getTask(Start→target→acquire.getName())
   else
      //scenario should not begin with a ResourceRelease or ResourcePas
      display warning message
      lqnFirstTask = lqnModel.getTask(Start→target→component.getName())
   //create a new first activity to use in the first task
   create lqnFirstActivity
   add lqnFirstActivity to lqnFirstTask
```

```
//create a new entry to call from the reference entry
create lqnFirstEntry
set lqnFirstActivity as first activity of lqnFirstEntry
add lqnFirstEntry to lqnFirstTask
```

```
//check the type of workload
if ClosedWorkload then
   set lqnRefEntry population and think time from workload
   make sync call from lqnRefEntry to lqnFirstEntry
else
   set lqnRefEntry interarrival time and type from workload
   make async call from lqnRefEntry to lqnFirstEntry
```

```
//traverse the scenario past the initial step following the start
lastLqnElement = lqnFirstActivity
ScenarioPathTraversal( GetNextScenarioElement( Start→target ) )
```

```
if currentScenarioElement is of type Step then
   currentLqnElement = get activity corresponding to the step
   connect lastLqnElement in sequence with currentLqnElement
   //continue traversal
   lastLqnElement = currentLqnElement
   ScenarioPathTraversal( GetNextScenarioElement( lastLqnElement ) )
```

```
else if currentScenarioElement is of type CommStep then
   //handle sending of message
   sendActivity = get activity corresponding to the CommStep message
   send
   connect sendActivity in sequence with currentLqnElement
   lastLqnElementForTask( CommStep.txComp ) = sendActivity
   //handle receiving of message
   if CommStep.msgKind is reply then
      //get the last LQN activity from when the original call was made
      lastLqnElement = lastLqnElementForTask( CommStep.rcvComp )
      //continue traversal
      ScenarioPathTraversal( GetNextScenarioElement( lastLqnElement ) )
   else
      //making a call, get LQN entry receiving the call
      calledEntry = get entry corresponding to the CommStep message
      receive
      lastLqnElement = calledEntry.getFirstActivity
      //continue traversal
      ScenarioPathTraversal( GetNextScenarioElement( lastLqnElement ) )
```

Algorithm	**GenerateLQN(CSM)**

```
        else if currentScenarioElement is of type Fork OR Branch then
           //connect LQN fork activity
           forkActivity = get activity corresponding to the fork or branch
           connect forkActivity in sequence with currentLqnElement
           //traverse each outgoing path
           for each outgoing path do
              outActivity = get activity corresponding to the outgoing path
              lastLqnElement = outActivity
              //continue traversal
              ScenarioPathTraversal( GetNextScenarioElement( lastLqnElement ) )
        else if currentScenarioElement is of type Join OR Merge then
           //wait until all incoming paths have been traversed
           while not all incoming paths have been traversed do
              //connect incoming path
              inActivity = get activity corresponding to the incoming path
              connect currentLqnElement in sequence with inactivity
              //stop traversal
           //all incoming paths have been traversed, connect LQN join activity
           joinActivity = get activity corresponding to the join or merge
           lastLqnElement = joinActivity
           //continue traversal
           ScenarioPathTraversal( GetNextScenarioElement( lastLqnElement ) )
        else if currentScenarioElement is of type End then
           //stop traversal
```

6.3.1. CSM to LQN Testing

TEST COVERAGE BY ALGORITHM

The algorithms were implemented by J. Muttulingam and tested on cases developed by the author. Table 27 lists the test cases used during the implementation of the CSM to LQN algorithms and their purposes, which algorithm they test, and the test success and failure conditions. Each test case has multiple test models.

Table 27. Test cases for the CSM transformation algorithm

Test Case	Algorithm Checked	Success Condition	Failure Condition
T1: invalid CSM syntax check for constraints on well-formed scenarios specified in section 3.4.3	CSMValidation	warning generated for invalid CSM elements	invalid CSM elements are missed
T2: Sequential CSM scenario, single component	ScenarioPathTraversal	lists all scenario elements as they are traversed	does not list all scenario elements as they are traversed
	GetNextScenarioElement	get the next scenario element	cannot get the next scenario element

Test Case	Algorithm Checked	Success Condition	Failure Condition
T3a: scenario with a Fork and matching Join, within a single component	ScenarioPathTraversal	lists all scenario elements as they are traversed; traversal order is path segment to Fork, outgoing path segments to Join, path segment after Join	does not list all scenario elements OR traversal order is incorrect
	AssignPathSegments	path segments are identified and have segment IDs	path segments are not identified correctly
T3b: scenario with a Fork, empty non-joining path, and Join of the other paths, in a single component	ScenarioPathTraversal	lists all scenario elements as they are traversed; traversal order is path segment to Fork, outgoing path segments to Join, path segment after Join	does not list all scenario elements OR traversal order is incorrect
	CleanUp EmptyPathSegments	empty path segment is identified and removed	empty path segment is not identified and removed
T4: scenario with a Branch and matching Merge, in a single component	ScenarioPathTraversal	lists all scenario elements as they are traversed; traversal order is path segment to Branch, outgoing path segments to Merge, path segment after Merge	does not list all scenario elements OR traversal order is incorrect
	AssignPathSegments	path segments are identified and have segment IDs	path segments are not identified correctly
T5a: CSM with a Step with a sub-scenario, single component	flatten sub-scenarios	sub-scenario is flattened	sub-scenario is not flattened
T5b: CSM with two Steps with the same sub-scenarios, in a single component	create unique instances of all sub-scenarios	two sub-scenario instances are created	sub-scenario instances are not created
	flatten sub-scenarios	sub-scenario is flattened	sub-scenario is not flattened
T5c: CSM with a sub-scenario, single component, component acquired again (unnecessarily) and released in sub-scenario	remove duplicate ResourceAcquire and ResourceRelease in sub-scenarios	duplicate ResourceAcquire and ResourceRelease in sub-scenario are deleted	duplicate ResourceAcquire and ResourceRelease in sub-scenario are not deleted
	flatten sub-scenarios	sub-scenario is flattened	sub-scenario is not flattened
T5d: CSM with a sub-scenario in which a different component is acquired and released	remove duplicate ResourceAcquire and ResourceRelease in sub-scenarios	duplicate ResourceAcquire and ResourceRelease in sub-scenario not found	non-duplicate ResourceAcquire and ResourceRelease in sub-scenario are deleted
	flatten sub-scenarios	sub-scenario is flattened	sub-scenario is not flattened

Test Case	Algorithm Checked	Success Condition	Failure Condition
T6: CSM with 4 components	assign components to all StepType elements	all Steps have valid component attributes	all Steps do not have valid component attributes
T7: set of 6 CSMs with CommSteps missing or having extra ResourceAcquire and Resource Release	clean up CommSteps	missing CommStep attributes are filled in	CommSteps still miss attributes
	clean up resource acquisition and release	missing or extra ResourceAcquire and ResourceRelease after/before CommSteps are addressed	missing or extra ResourceAcquire and ResourceRelease after/before CommSteps are still there
T8a: CSM with implicit sync call expressed by async messages	Interaction discovery	sync interaction discovered and CommStep msgKind filled in correctly	sync interaction not discovered and CommStep msgKind filled in incorrectly
T8b: CSM with an implicit sync call and an async call	Interaction discovery	sync interaction discovered and CommStep msgKind filled in correctly	sync interaction not discovered and CommStep msgKind filled in incorrectly
T8c: CSM with forwarded call	Interaction discovery	forwarding interaction discovered and CommStep msgKind filled in correctly	forwarading interaction not discovered and CommStep msgKind filled in incorrectly
T8d: CSM with forwarded and async calls	Interaction discovery	forwarding interaction discovered and CommStep msgKind filled in correctly	forwarding interaction not discovered and CommStep msgKind filled in incorrectly
T8e: CSM with implicit sync and forwarded calls	Interaction discovery	sync and fwd interaction discovered and CommStep msgKind filled in correctly	sync and fwd interaction not discovered and CommStep msgKind filled in incorrectly
T8f: CSM with implicit sync, async and forwarded calls	Interaction discovery	sync and fwd interaction discovered and CommStep msgKind filled in correctly	sync and fwd interaction not discovered and CommStep msgKind filled in incorrectly
T8g: CSM with multiple async calls	Interaction discovery	no spurious sync and fwd interaction discovered and CommStep msgKind filled in correctly	spurious sync or fwd interaction are discovered and CommStep msgKind filled in incorrectly

6.4. CSM TO QN TRANSFORMATION

The CSM2QN transformation algorithm generates a Queueing Network object “qn” from a given CSM object “CSM”. The resulting QN is saved as a spreadsheet with elements and values that correspond to the input CSM.

The algorithm for the CSM to QN transformation is shown below:

Table 28. High-level algorithm for GenerateQNPerfModel

Algorithm	**GenerateQNParams(CSM)**
Input	**CSM (*CSMType*)**
Output	**QN parameters**
This algorithm generates the parameters of a QN performance model based on the input CSM.	
Pseudocode	

```
for all ProcRes[i] in CSM.ProcessingResourceList do
   create QNHost for ProcRes[i]

for all Scenario[i] in CSM.ScenarioList do
   if Scenario[i].StartList[0] contains Workload then
      //begin CSM traversal
      currentElement = Scenario[i]→StartList[0]
      //determine the visit ratios for all Steps
      // (the number of times each Step is executed during the scenario)
      if type of currentElement == Step do
         QNHost.totalDemand += currentElement.hostDemand
      //continue ScenarioPathTraversal
      ScenarioPathTraversal( NextScenarioElement ( currentElement ) )
```

The QN models generated by CSM2QN are saved in a tab-delimited spreadsheet format with the sums for each Step and the total demand for each Host.

The parameters can be used in defining a QN model in a suitable solver package.

Automatic production of a portable PMIF specification (Performance Model Interchange Format) [113] appears to be straightforward but was not included in this research.

The QN demands for the GetBuyConfirmPage scenario in Fig 17 are shown below.

Table 29. QN for the GetBuyConfirmPage CSM shown in Fig 18

Step	Task	Host	Service Time	Visits	Demand
Call	EB	ClientProc	0	1	0
getBuyConfirmPage	EB	ClientProc	1	1	1
getShoppingCart	EB	ClientProc	1	1	1
getBuyConfirmImgs	IS	ServerProc	0.5	1	0.5
reply	EB	ClientProc	0	1	0
displayPage	EB	ClientProc	1	1	1
setShippingAddr	EB	ClientProc	0.5	0.05	0.025
matchAddrRecord	DB	DBProc	2	0.05	0.1
insertAddrRecord	DB	DBProc	1	0.025	0.025

checkout	EB	ClientProc	1	1	1
createOrder	EB	ClientProc	5	1	5
insertOrderRecord	DB	DBProc	1	1	1
getAuthorization	EB	ClientProc	1	1	1
buildAuthorizationRequest	EB	ClientProc	5	1	5
requestPGE	PGE	null	30	1	30
decryptAndExtractAuthID	EB	ClientProc	5	1	5
createCreditCardRecord	DB	DBProc	1	1	1
clearShoppingCart	EB	ClientProc	0.5	1	0.5
insertOrderLineRecord	DB	DBProc	1	5	5
dummy	EB	ClientProc	0	5	0
updateItemStock	DB	DBProc	1	5	5
Device	**Demand**				
ServerProc	2.5				
DBProc	15.15				
ClientProc	20.525				

7. CASE STUDIES DEMONSTRATING THE AUTOMATIC GENERATION OF LQN FROM CSM

These case studies demonstrate the application of the end-to-end process to full-scale applications. The first application (BSS) is an academic case study in asynchronous concurrent system design with interesting features related to message types and resource contexts. The second (TPC-W) is a representative web-based system, actually the specification for a benchmark for e-commerce systems. The third (LMP) is based on an actual web-based information system that the author worked on.

7.1. BUILDING SECURITY SYSTEM

7.1.1. System Description

The Building Security System (BSS) has been used as an ongoing case study in the PUMA research [127] [128].

The BSS models a security system that captures time-lapse video frames from different cameras, buffers them, and sends them to a database for storage. The video capture scenario can be paired with a building access scenario that models how doors are opened. The building access scenario was not used for this example.

7.1.2. UML Model

The BSS case study presented in this thesis was specified with UML 2.0 and MARTE 1.1 using MagicDraw 17.0.3. The Activity Diagram for the BSS image capture scenario is shown in Fig 28 and the deployment is shown in Fig 29.

DEPLOYMENT

The BSS is deployed on two separate nodes. The ControlNode hosts the CaptureImage and StoreImage processes which control the video capture and storage functions.

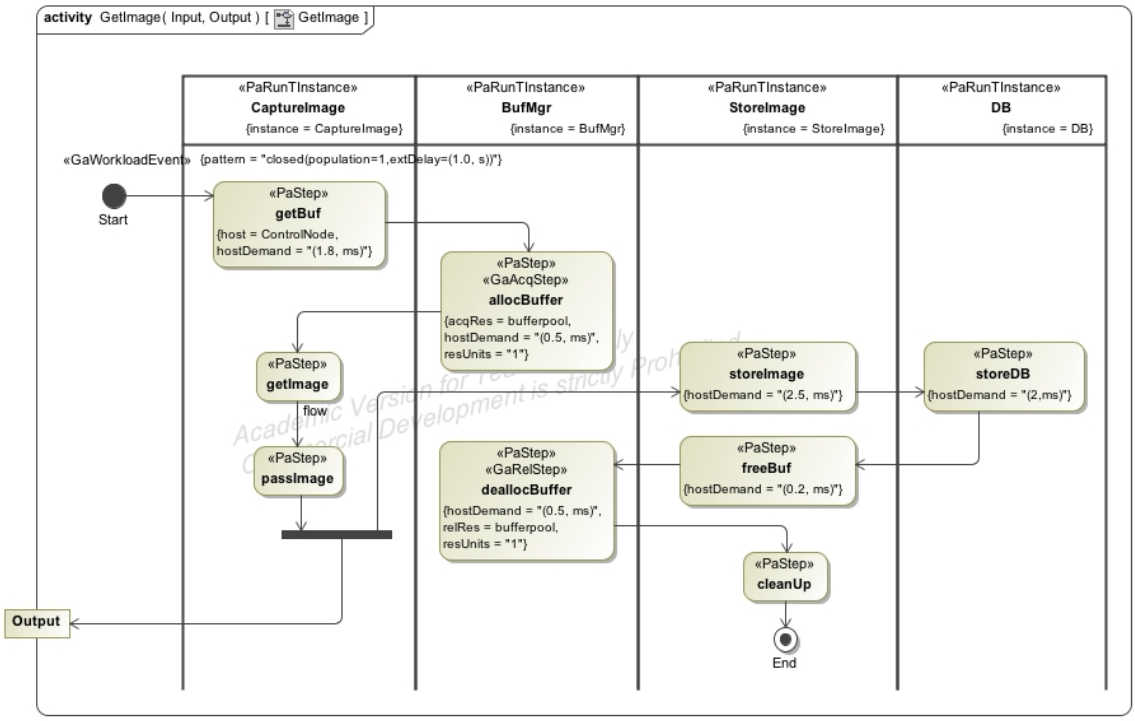


Fig 28 Activity Diagram for the BSS image capture scenario

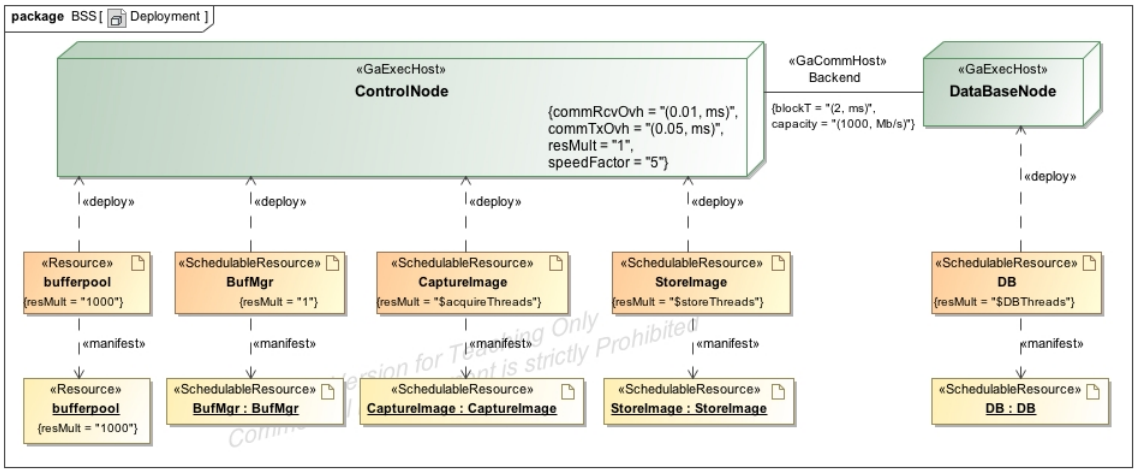


Fig 29 Deployment Diagram for the BSS

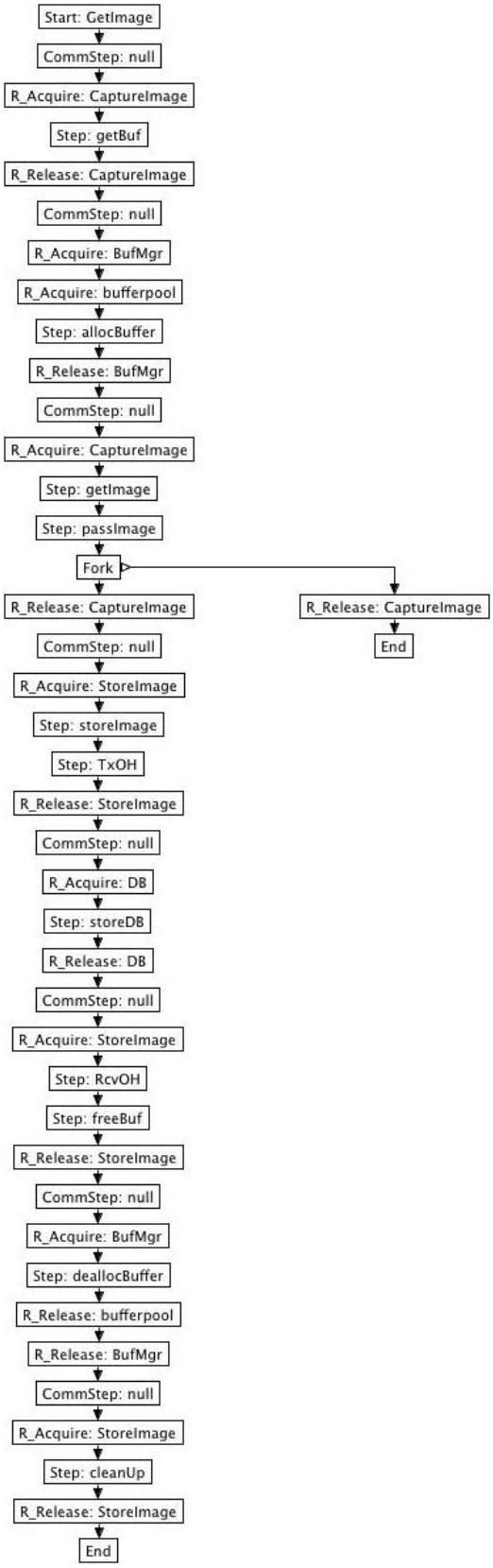


Fig 30 CSM for the image capture scenario of the Building Security System

The video buffers are finite logical resources and access to them is controlled by the BufMgr process which is also hosted on the ControlNode. The DB database is hosted on a separate DataBaseNode.

BEHAVIOUR

The GetImage functionality modeled in Fig 28 begins when CaptureImage requests a buffer from the BufMgr which acquires a buffer from the buffer pool. CaptureImage reads an image from a camera and passes it on to both an external output (for viewing) as well as to StoreImage for saving. StoreImage then sends the image to the DB database for storage, empties the buffer and calls the BufMgr to free the buffer so it can be used again. Finally, StoreImage performs a clean-up operation.

The usage of buffers to store images is a basic functional feature of the BSS. In practice, the BSS stores images from multiple cameras around the building and this leads to overlapping filling and storing of buffers in a non-nested resource context as discussed in chapter 4.

7.1.3. CSM Model

DEPLOYMENT

The CSM resources for the BSS are listed below:

```
<Component id="CM00000" name="CaptureImage" host="PR00002"/>
<Component id="CM00001" name="BufMgr" host="PR00002"/>
<Component id="CM00002" name="StoreImage" host="PR00002"/>
<Component id="CM00003" name="DB" host="PR00001"/>
<ProcessingResource id="PR00001" name="DataBaseNode" multiplicity="1"
speedFactor="1.0"/>
<ProcessingResource id="PR00002" name="ControlNode" multiplicity="1"
commRcvOvh="0.01" commTxOvh="0.05" speedFactor="5.0"/>
<LogicalResource id="LR00000" name=":bufferpool" multiplicity="1000"/>
```

Each swim lane from the Activity Diagram corresponds to a Component in the CSM and the nodes correspond to the CSM ProcessingResources. The buffer pool corresponds to the LogicalResource.

BEHAVIOUR

The CSM scenario for the BSS GetImage Activity Diagram is shown in Fig 30. The generation of CSM from Activity Diagrams results in a ResourceAcquire whenever the scenario enters a swim lane and a ResourceRelease whenever the scenario exits a swim lane. Transitions between swim lanes are interpreted as messages between the components and generate CommSteps in the CSM. Since the swim lane transitions do not have built-in message types in the Activity Diagram, each transition from one swim lane to another generates a "ResourceRelease-CommStep-ResourceAcquire" tuple in the CSM. Any unnecessary ResourceAcquire (of a component that has made a synchronous call when it receives the reply message) or ResourceRelease (of a component making a synchronous call) is removed before generating the LQN by the CleanUpResourceAcquisitionAndRelease algorithm described in section 5.3.9.

The external output after passImage results in an empty outgoing path segment after the Fork in the CSM. This empty segment is removed before generating the LQN by the CleanUpEmptyPathSegments algorithm described in section 5.3.7.

7.1.4. Generated LQN Model

The generated LQN for the BSS image capture scenario is shown in Fig 31.

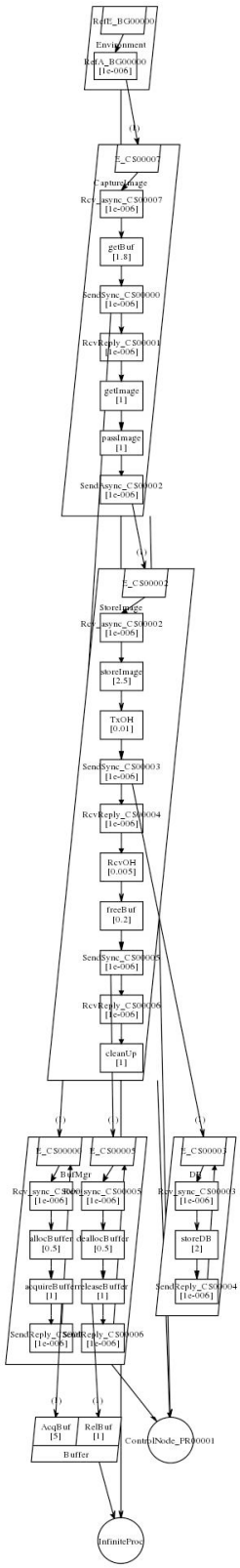


Fig 31 LQN for the image capture scenario of the Building Security System

DEPLOYMENT

The components from the CSM are represented by corresponding tasks in the LQN, and the CSM processing resources are represented by corresponding LQN processors. The logical resource for the buffer pool is represented by the Buffer semaphore task in the LQN – buffer acquisition is accomplished by a synchronous call to the AcqBuf entry and buffer release is accomplished by an asynchronous call to the RelBuf entry. The multiplicity of the buffer is greater than one but is not shown in the Figure.

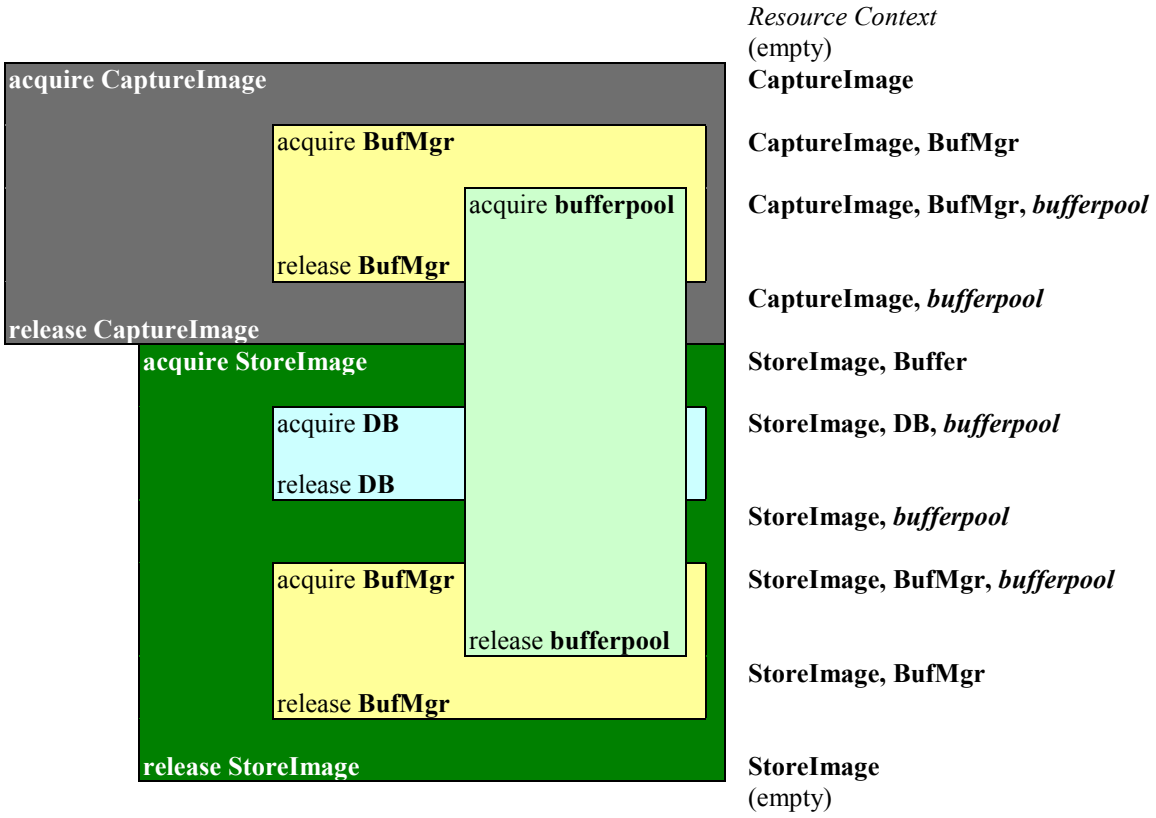


Fig 32 Resource context for the BSS showing the non-nested use of the bufferpool

BEHAVIOUR

The workload from the CSM is represented by the reference task Environment in the LQN.

The swim lane transitions in the original Activity Diagram do not have any specified message types and neither do the CommSteps in the corresponding CSM. The interaction discovery algorithm described in section 5.4 has identified that the calls to the BufMgr are synchronous while the call from CaptureImage to StoreImage is asynchronous. These message types are captured in the generated LQN as follows:

```
Task CaptureImage:
   Activity getBuf is followed by activity SendSync_CS00000 which makes a
   synchronous call to E_CS00000 in task BufMgr
   Activity passImage is followed by activity SendAsync_CS00002 which makes an
   asynchronous call to E_CS00002 in task StoreImage
Task BufMgr:
   Activity acquireBuffer makes a synchronous call to AcqBuf in semaphore task
   Buffer
   Activity acquireBuffer is followed by activity SendReply_CS00001 which
   replies to E_CS00000 in task BufMgr
   Activity releaseBuffer makes an asynchronous call to RelBuf in semaphore task
   Buffer
   Activity releaseBuffer is followed by activity SendReply_CS00006 which
   replies to E_CS00005 in task BufMgr
Task StoreImage:
   Activity storeImage is followed by activity SendSync_CS00003 which makes a
   synchronous call to E_CS00003 in task DB
   Activity freeBuf is followed by activity SendSync_CS00005 which makes a
   synchronous call to E_CS00005 in task BufMgr
Task DB:
   Activity storeDB is followed by activity SendReply_CS00004 which replies to
   E_CS00003 in task DB
```

7.1.5. Features Demonstrated

This example shows:

- The application of CSM transformations to a CSM that was generated from an Activity Diagram
- CSM validation (section 5.1)
- CSM scenario traversal (section 5.2)
- CSM normalization (section 5.3)
    - assign components to all StepType elements (section 5.3.2)
    - check for non-nested component contexts (section 5.3.3)
    - assign path segments to sequential scenario fragments (section 5.3.6)

    - clean up Forks with empty path segments (section 5.3.7)
    - clean up CommSteps (section 5.3.8)
    - clean up resource acquisition and release (section 5.3.9)
- Interaction discovery (section 5.4) of synchronous and asynchronous communication
- Generation of a LQN with a semaphore task to model a non-nested resource architecture of a buffer logical resource that used in an overlapping manner.

The performance of this system for different levels of resources (DB threads, buffer pool size) was studied in [131].

7.2. TPC-W

7.2.1. System Description

TPC is an e-commerce system used as a transactional web benchmark. TPC is specified by the Transaction Processing Performance Council, a non-profit corporation founded to define transaction processing and database benchmarks and to disseminate objective, verifiable TPC performance data to the industry. The current TPC benchmarks can be found at www.tpc.org [120].

The TPC-W model used in this research is based on an older benchmark [117] and was developed as part of the aspect-oriented CSM work introduced in chapter 3.

TPC-W models the activities of an online bookstore website. Users can browse and order products from the website with different scenarios showing the different user interactions. The full TPC-W specification describes 14 different web pages of the website. The user always starts from the homepage and returns there after completing any scenario.

Users can browse pages containing a list of new or best-selling books grouped by subject, or perform searches against all books based upon a title, author or subject. A product page gives the detailed information for the book. Users may order books through order

pages. New customers must go through a registration step while existing users can have their account info pulled from the database. When an order is placed, the system obtains a payment authorization from the Payment Gateway Emulator (PGE) and presents an order confirmation page.

Since the UML diagrams and CSMs used in this case study were produced in earlier work on the SPT-based UML-to-CSM transformation, pre-MARTE, the annotations shown on the diagrams are not precisely in MARTE format. The text explains any differences.

7.2.2. UML Model

The TPC-W case study presented in this thesis was specified using IBM Rational ROSE Architect (RSA 6.0) with UML 1.1 and SPT with customized annotations as described in [127] [139].

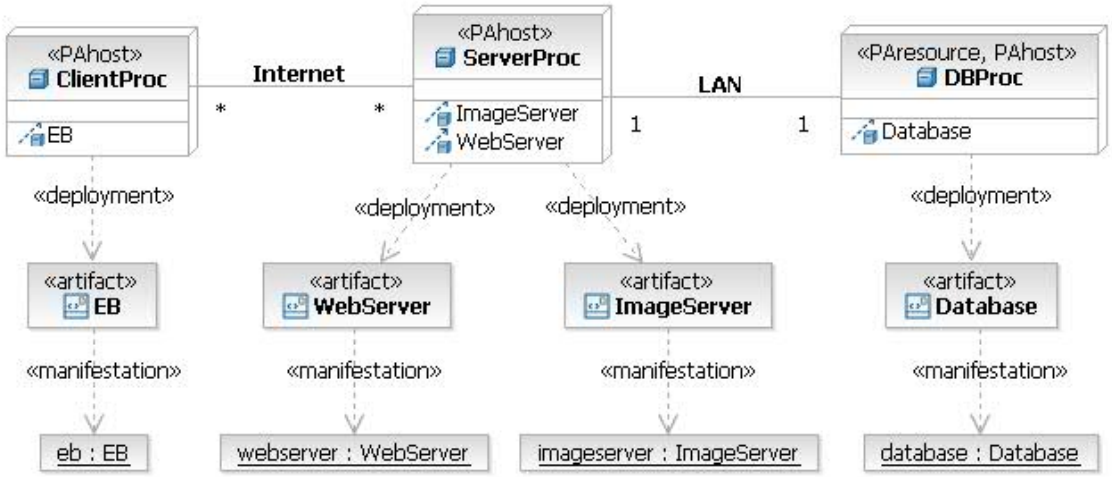


Fig 33 Deployment diagram for TPC-W

DEPLOYMENT

The TPC-W Deployment Diagram is shown in Fig 33. The Deployment Diagram shows the software components, their corresponding artifacts and the deployment of artifacts on

processing nodes. The ClientProc node hosts the user emulated browser EB, the ServerProc hosts the TPC-W WebServer and ImageServer, while the DBProc hosts the system Database. The DBProc node has a multiplicity of 5. If the multiplicity is not specified, it is assumed to be 1.

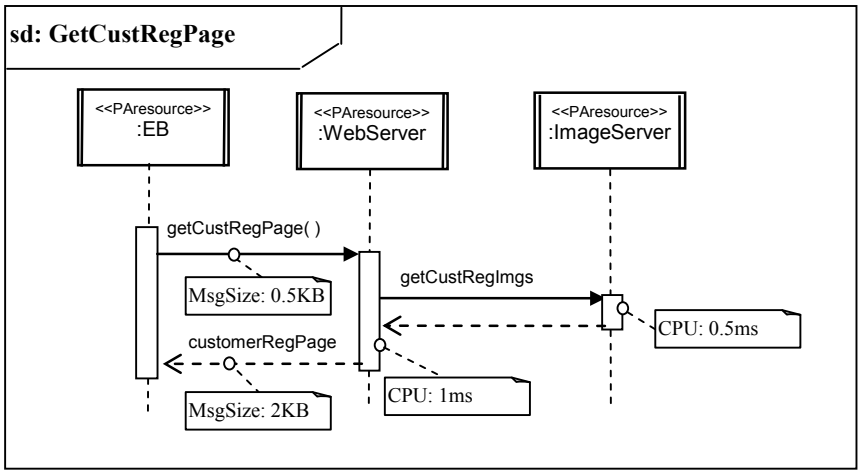
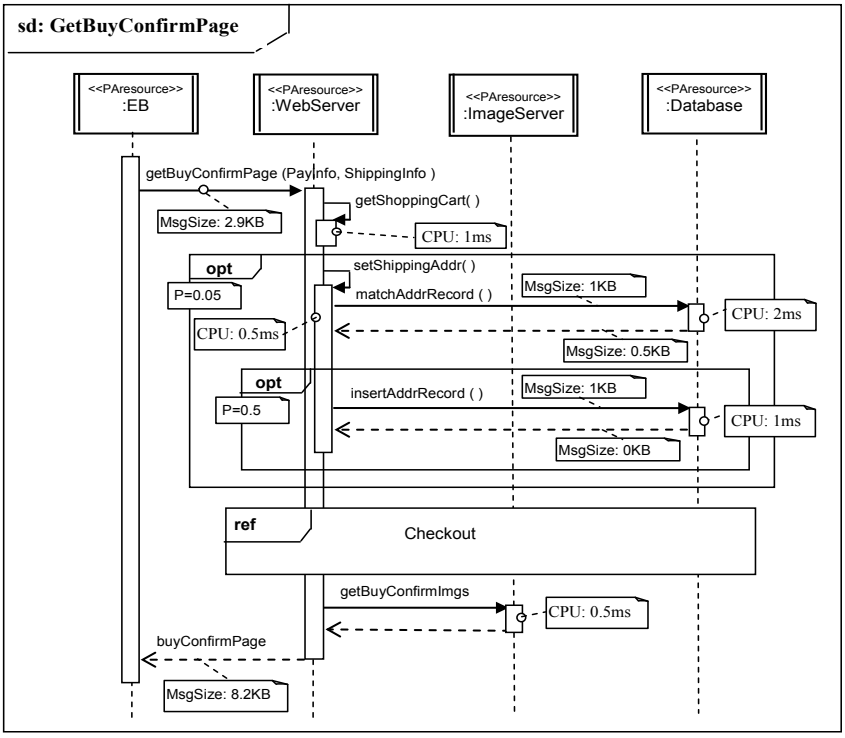


Fig 34 Sequence Diagram for the GetCustRegPage sequence

BEHAVIOUR

The behaviour in this case study consists of a subset of the three scenarios from the full TPC-W specification that have secure communication aspects. The behaviour of the security layer is not shown in the scenarios in this section, but the example was analyzed with and without it, using the techniques of section 3.6 to include it.

The Sequence Diagram for the GetCustRegPage scenario shown in Fig 34 provides the registration web page to the user as modeled by the EB. The scenario starts with a non-secure message between EB and WebServer but ends with a secure reply form. The user can then register using the returned page.



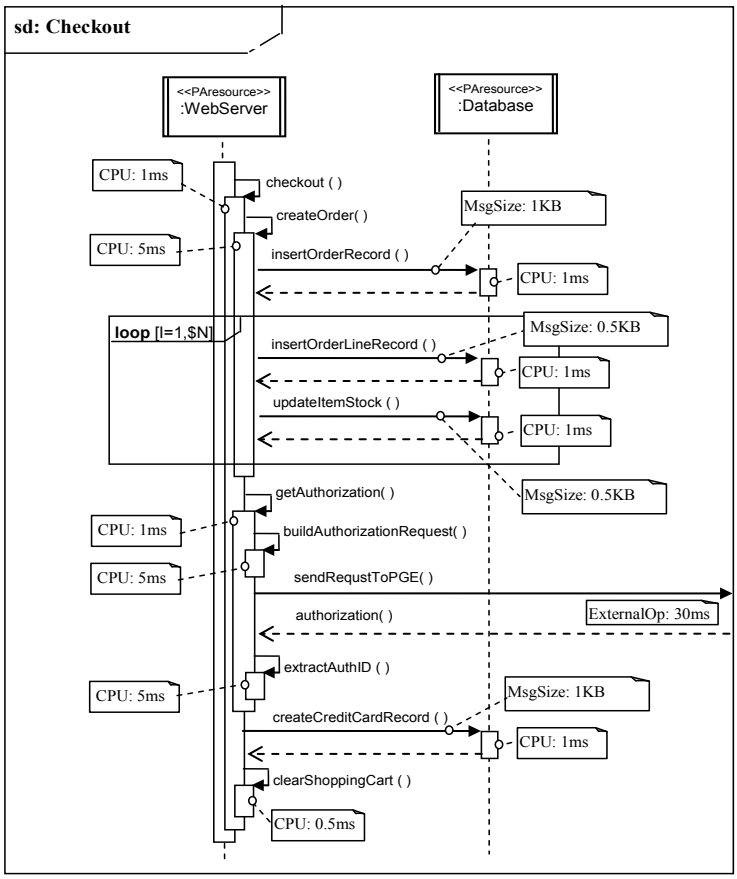


Fig 35 Sequence Diagrams for the GetBuyConfirmPage and Checkout scenarios

The following operations are performed:

- EB issues a request for the customer registration page;
- WebServer gets the necessary images (company logo, button images, etc.) from ImageServer;
- WebServer renders the secure HTML customer registration page and returns it to EB.

The GetBuyConfirmPage scenario is described in the first Sequence Diagram shown in Fig 35. The shopping cart contents are transferred into a newly created order for the registered customer and then payment is authorized. A web page containing the details of the newly created order is returned to the EB. The following operations are performed:

- EB issues a request to WebServer for "buy confirm page";
- WebServer gets the corresponding shopping cart object;
- With 5% probability, a shipping address is passed from EB.
- WebServer tries to match the shipping address with the database
    - If no address record is found then a new address record is inserted
- Webserver invokes the Checkout sub-scenario (as a ref fragment)
- WebServer gets necessary images from ImageServer
- WebServer generates the HTML code for the buy confirm page and returns it to EB.

The Checkout sub-scenario is shown in the second Sequence Diagram in Fig 35. Checkout creates a new order in the database, then obtains an authorization from the external Payment Gateway Emulator (PGE). Once the credit card payment is registered in the database, the cart is cleared.

The operation performed by the PGE external system is represented as an external operation which indicates the name of the operation and the number of visits [86]. The SPT performance attributes used in this model are:

- CPU host demand for operations (sometimes shown as CPU; the MARTE attribute is hostDemand), applied to execution occurrences and messages stereotyped as <<PAstep>> (MARTE: PaStep)
- probability PAprob (MARTE: prob) for alt and opt interaction operands stereotyped as <<PAstep>>
- MsgSize (MARTE: msgSize) for messages passed.
- repetition count rep for loop interaction operands stereotyped as <<PAstep>>
- PArate (MARTE: speedFactor) is the speed factor for host devices, relative to a nominal reference type of host (applied to nodes stereotyped as <<PAhost>>)
- PAcapacity (MARTE: resMult) for host multiplicity

The figures do not show the full SPT syntax for performance annotations, in order to limit clutter. However, the complete stereotypes and corresponding tagged values were defined in the RSA UML tool used to generate the CSM models.

7.2.3. CSM Model

DEPLOYMENT

The CSM resources for TPC-W are listed below:

```
<Component id="id5" name="eb" host="id6"/>
<Component id="id20" name="webserver" host="id21"/>
<Component id="id38" name="database" host="id39"/>
<Component id="id72" name="imageserver" host="id21"/>
<ExternalDemand id="id146" name="PGE"/>
<ProcessingResource id="id6" name="ClientProc" opTime="1.0"/>
<ProcessingResource id="id21" name="ServerProc"/>
<ProcessingResource id="id39" name="DBProc" multiplicity="5" opTime="1.0"/>
```

Each lifeline from the Sequence Diagram corresponds to a Component in the CSM and the nodes in the Deployment Diagram correspond to the CSM ProcessingResources. The external payment gateway corresponds to the ExternalDemand.

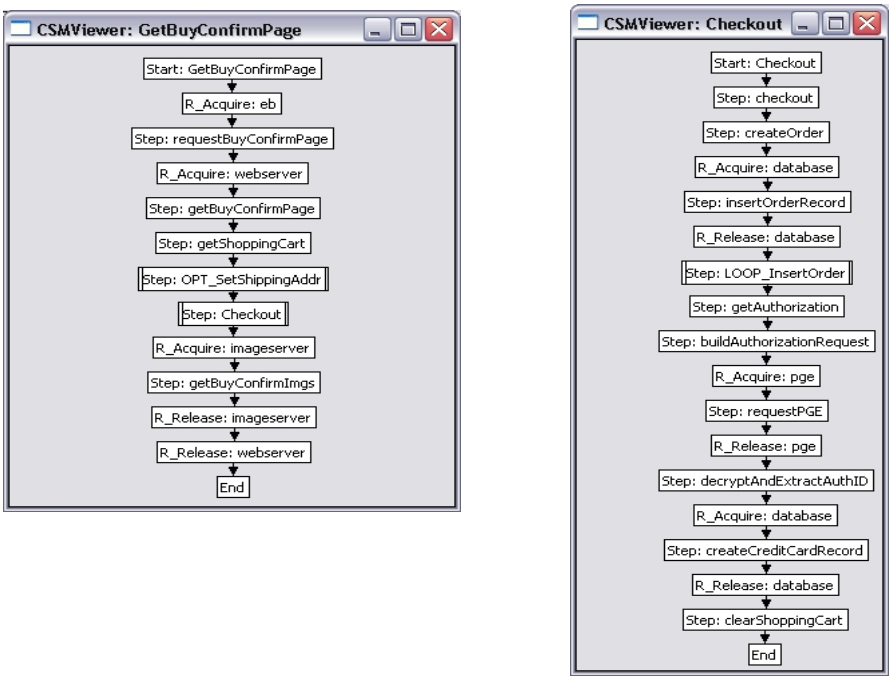


Fig 36 CSM for the GetBuyConfirmPage and Checkout scenarios

BEHAVIOUR

Fig 36 shows the scenarios for GetBuyConfirmPage and Checkout in the first version of the CSM definition.

The generated CSMs can use the explicit message types in Sequence Diagrams in order to determine whether ResourceAcquire and ResourceRelease steps are actually required for a given message. When message types are specified in the design model, the process to generate a CSM can make use of this information to avoid unnecessary ResourceReleases of a component making a synchronous call or unnecessary ResourceAcquires of a component that receives a reply to a previous synchronous call. However if there are and unnecessary ResourceAcquire or ResourceRelease elements in the CSM, these would be removed before generating the LQN model using the Clean Up Resource Acquisition and Release algorithm from section 5.3.9.

The CSM shows steps with sub-scenarios for the optional SetShippingAddr and the repeated InsertOrder interaction fragments from the UML model.

Additional CSMs were also created with SSL communications added for some messages as described in the AO-CSM section 3.6.

7.2.4. Generated LQN Model

The generated LQNs for GetBuyConfirmPage and Checkout, without and with SSL, are shown in Fig 37. Part (a) shows the LQN without the SSL security aspect and part (b) shows the LQN with the SSL security aspect added.

DEPLOYMENT

The components from the CSM are represented by corresponding tasks in the LQN, and the CSM processing resources are represented by corresponding LQN processors. The generated LQN also has additional tasks used to model the repeated and optional interaction fragments from the UML design and the CSM.

BEHAVIOUR

The LQN captures the synchronous interactions between the user EB and WebServer, and between the WebServer and the ImageServer and Database. The addition of SSL introduces forwarding chains between the user EB and the WebServer that are captured in the LQN.

Fig 38 shows the calculated effect on throughput of adding SSL to the GetBuyConfirmPage and Checkout scenarios. The SSL security overhead reduces the capacity (maximum achievable throughput) by a little more than 10%.

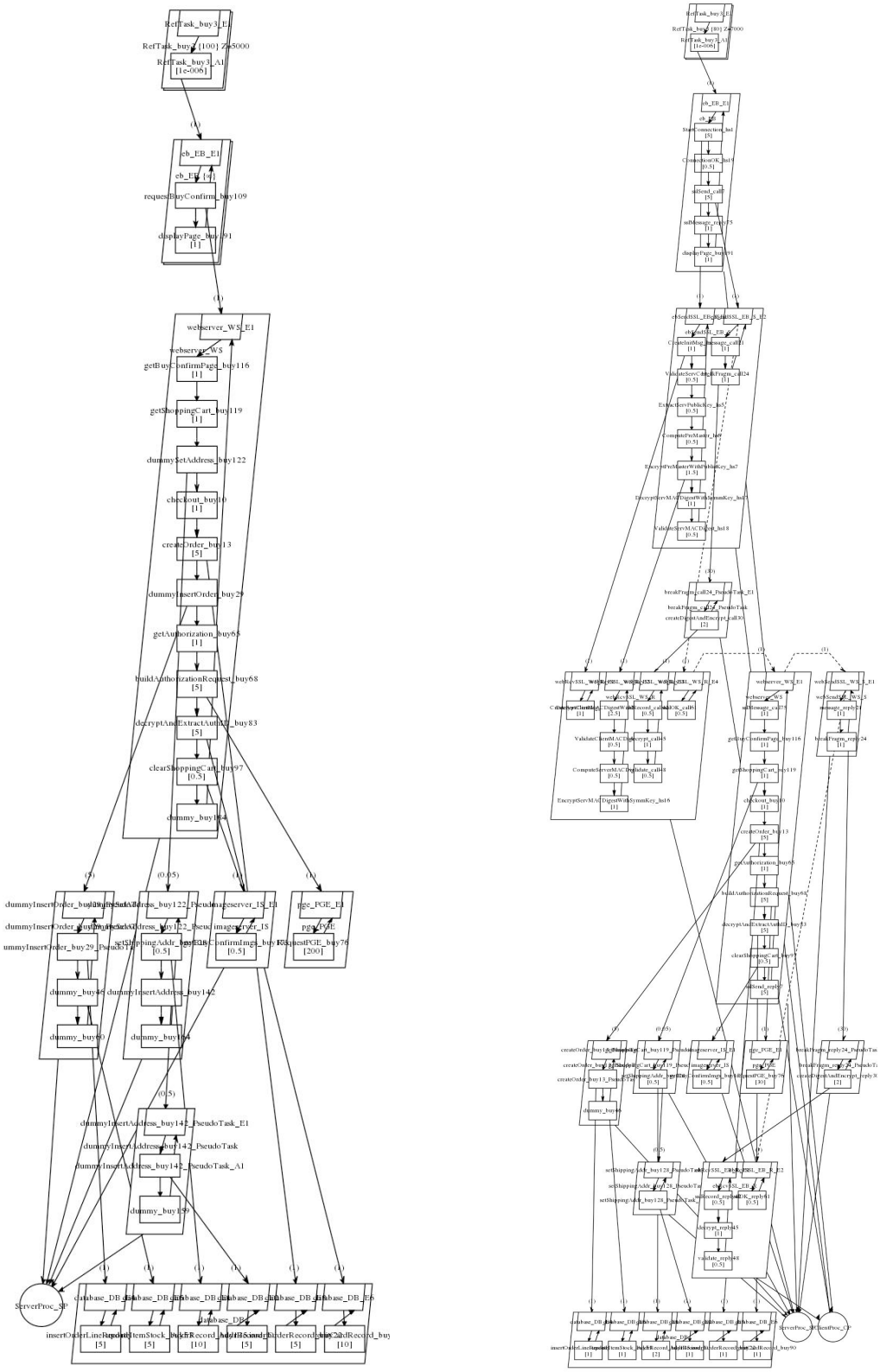


Fig 37 LQNs for GetBuyConfirmPage with Checkout, without and with SSL

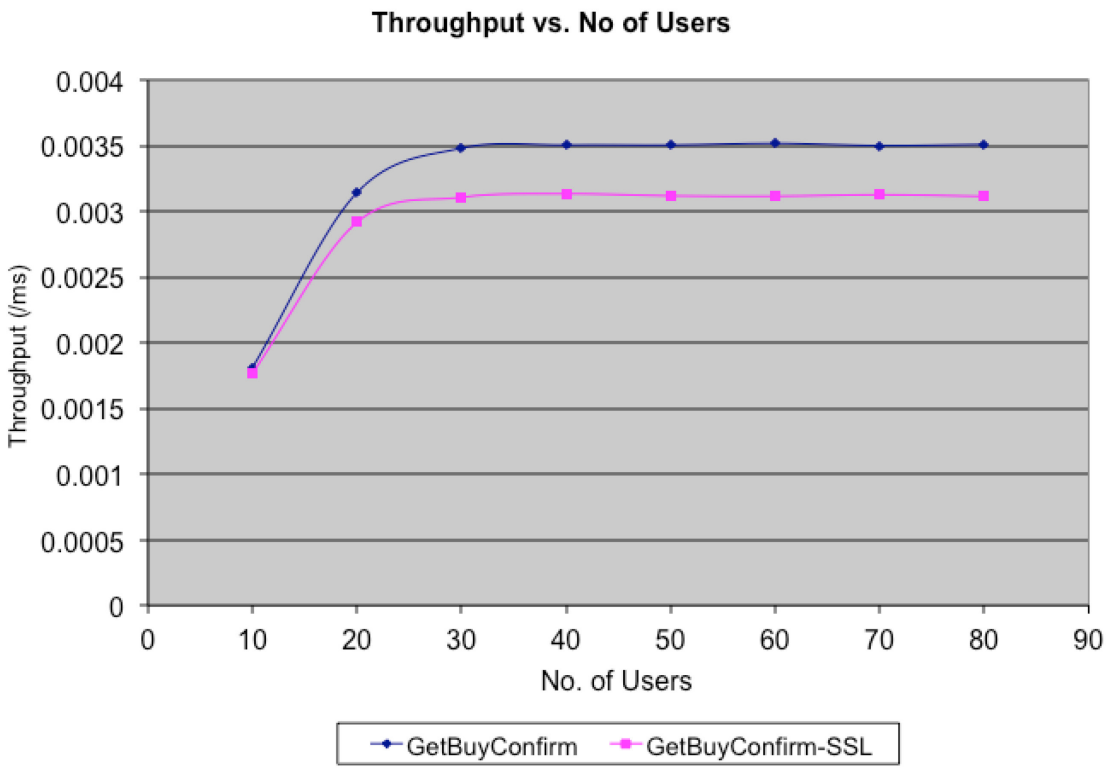


Fig 38 Throughput vs. number of users for the GetBuyConfirmPage with Checkout scenario, without and with SSL

7.2.5. Features Demonstrated

This example shows:

- The application of CSM transformations to a CSM that was generated from a Sequence Diagram
- CSM validation (section 5.1)
- CSM scenario traversal (section 5.2)
- CSM normalization (section 5.3)
  - create unique instances of all sub-scenarios (section 5.3.1)
  - assign components to all StepType elements (section 5.3.2)
  - check for non-nested component contexts (section 5.3.3)
  - remove duplicate ResourceAcquire and ResourceRelease elements in sub-scenarios (section 5.3.4)
  - flatten sub-scenarios (section 5.3.5)

    - assign path segments to sequential scenario fragments (section 5.3.6)
    - clean up CommSteps (section 5.3.8)
    - clean up resource acquisition and release (section 5.3.9)
- Interaction discovery (section 5.4) of synchronous and forwarding communication
- LQN generation (section 6.3)

7.3. LABOUR MARKET PORTAL

7.3.1. System Description

Enterprise-level web-based sites and services are a common form of service delivery to the public. They are also a common source of performance and capacity related problems when they are rolled out or updated. Recent high profile examples include the October 2013 launch of www.HealthCare.gov (the US government website for health care insurance for the Affordable Care Act, commonly referred to as ObamaCare) and the March 2013 launch of Electronic Arts' SimCity game.

The case study models such a system in the form of an online Labour Market Portal (LMP) which allows jobseekers to find job postings as well as relevant labour market information such as prevailing wages, skills required, whether occupations are regulated, and available training. Examples of these systems include www.JobBank.gc.ca in Canada and www.USAJobs.gov in the US for government, as well as www.Monster.com and www.Workopolis.com in the private sector.

Performance is a major concern for systems of this type as they must be able to handle hundreds of thousands of job postings or other records and millions of visitors per month. Employers can also log into the LMP to create and manage job postings. Employers must have authenticated accounts in order to be allowed to post jobs. The LMP system also

aggregates job postings and labour market information from external sources on a daily basis or more frequently if the information is available. Job postings from external sources are validated to ensure they comply with the same conditions as those posted by authenticated employers.

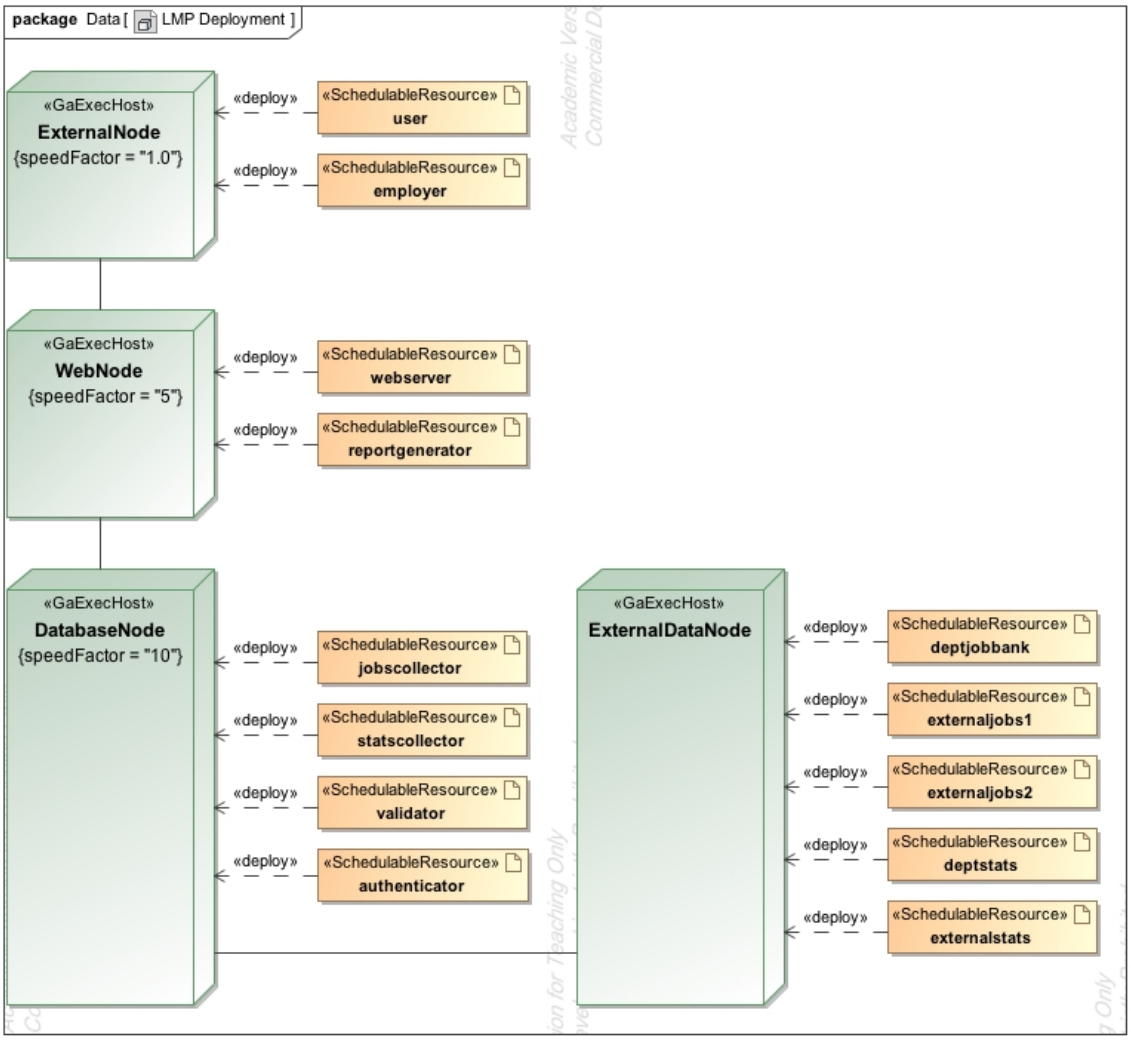


Fig 39 LMP Deployment Diagram

7.3.2. UML Model

DEPLOYMENT

Fig 39 shows the Deployment Diagram for the LMP. Processes for the jobseekers and employers are hosted on an External Node that's independent of the rest of the LMP system.

Users interact with the LMP through a webserver. The webserver calls the report generator when needed in order to create reports on job postings and labour market information.

A separate internal database node caches the job posting and labour market data. To speed things up, a job collector process provides cached summary information for job postings when occupational reports are generated. Similarly, a statistics collector process assembles labour market information. The internal database node also hosts the employer authentication service and the job posting validator.

The original detailed job postings and labour market information is owned and managed by entities external to the LMP. This is modeled through an external database node which hosts the partner job banks for the job posting information and the partner statistical datasets for the labour market information. The LMP accesses these external databases to update its data.

The internal and external network and communications links between the three nodes are not modeled explicitly in this deployment. Estimates for the associated communication overheads are included in the service demands for the operations, and network latency is ignored.

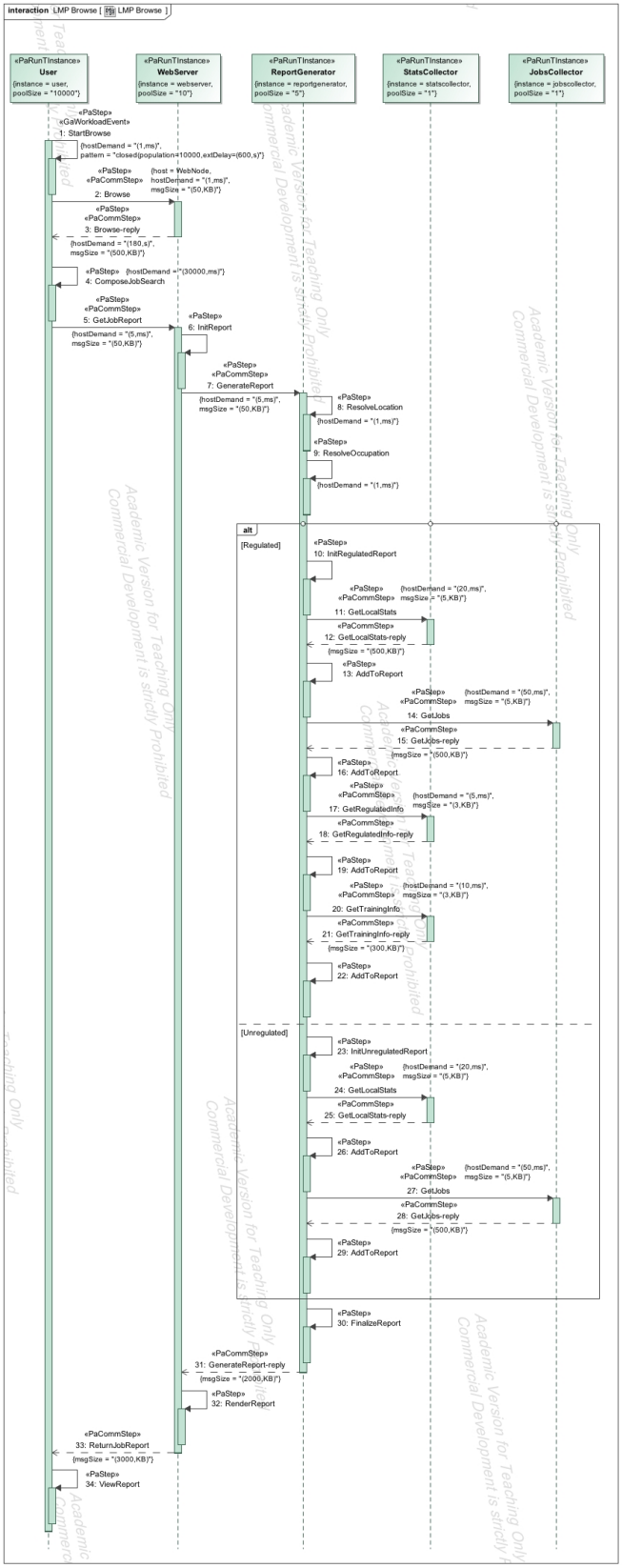


Fig 40 Sequence Diagram for the LMP Browse scenario

BEHAVIOUR

The LMP has four main scenarios. Two scenarios model jobseekers requesting occupational reports and job posting details. The third scenario models employers managing their job postings. The last scenario updates job posting and labour market data.

Alternatives are used to model the different types of information requests based on the user's search parameters. Parallelism in the system is not shown in the scenarios but is handled by multiplicity in the deployment and the processes involved.

Fig 40 shows the Sequence Diagram for the jobseeker occupational browsing scenario. The scenario unfolds as follows:

- The User (jobseeker) requests an occupational report
- The WebServer provides an occupation search form
- The User fills out the search criteria and submits the form
- The WebServer passes on the search request to the ReportGenerator
- The ReportGenerator resolves the location and the occupational coding for the search
    - If the search is for a regulated occupation the report is assembled with local labour market information, regulatory information, and training information from the StatsCollector as well as a list of related job postings from the JobCollector
    - If the search is for an unregulated occupation the report will only include the local labour market information from the StatsCollector and the related job postings from the JobCollector.
- The ReportGenerator returns the report to the WebServer which embeds it into a result page for the User

Fig 41 shows the Sequence Diagram for the jobseeker looking at a detailed job posting scenario. The scenario is as follows:

- The User requests a detailed job posting from the occupational report

- The WebServer passes on the search request to the ReportGenerator
  - If the job is in a regulated occupation then the job details from the JobCollector are augmented with regulatory and training information from the StatsCollector
  - If the job is in an unregulated occupation then the report will only include the job details from the JobCollector and training information from the StatsCollector
- The ReportGenerator returns the report to the WebServer which embeds it into a result page for the User

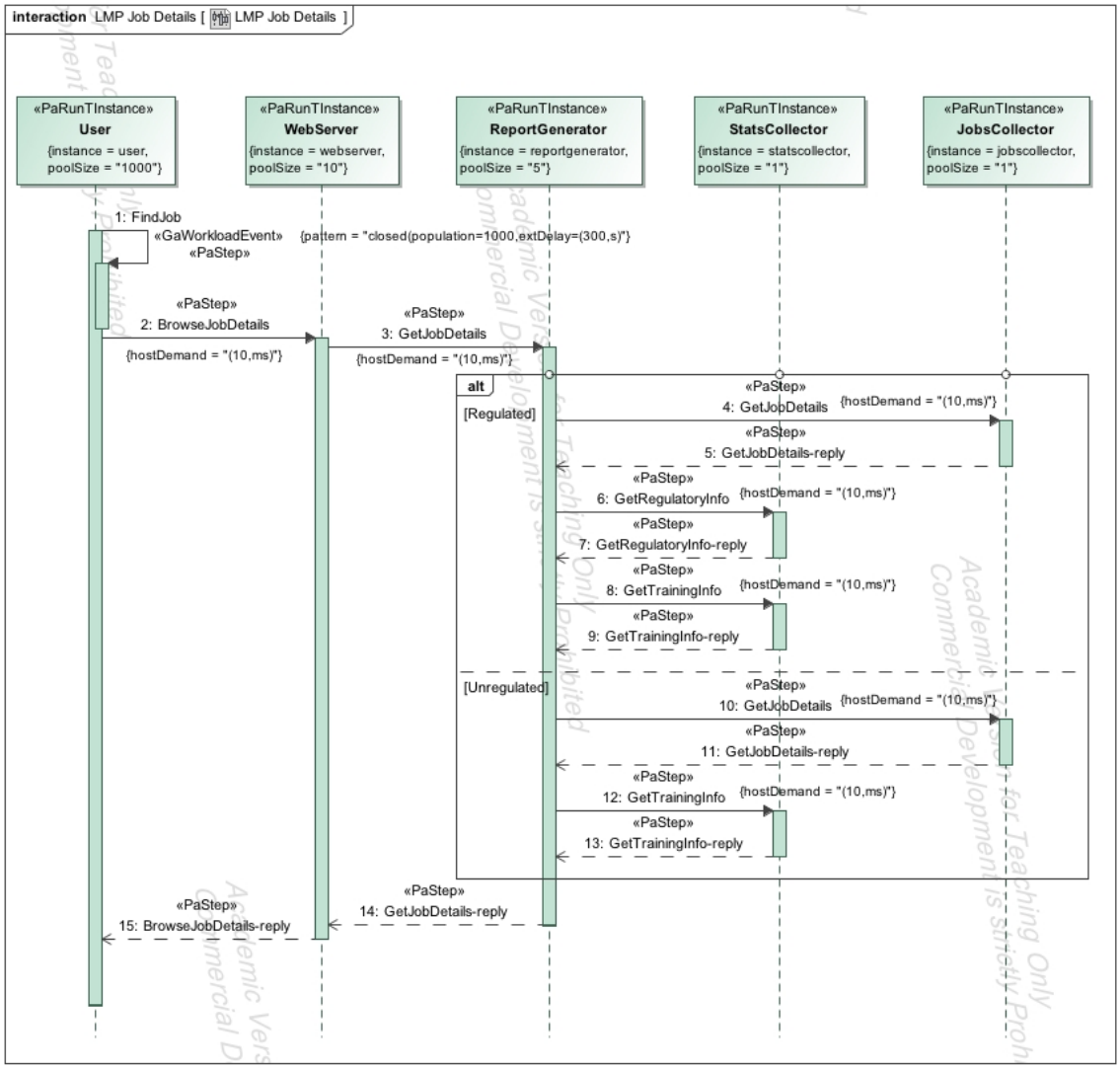


Fig 41 Sequence Diagram for the LMP Job Details scenario.

Fig 42 shows the Sequence Diagram for the employer managing job postings scenario.

The scenario is as follows:

- The Employer logs in to the employer part of the site
- The WebServer authenticates the login information, fetches the Employer's current job postings and provides an account page
- The Employer manages her job postings
  - If editing a job posting, the WebServer fetches the job posting details from the EmployerDB and generates a job posting editing page. The Employer edits the job posting details, saves the changes, and the WebServer sends a request to the EmployerDB to save the new data
  - If adding a job, the WebServer generates a job posting form which the Employer fills out. When ready, the WebServer sends a request to the EmployerDB to save the new data
  - If deleting a job, the WebServer provides a list of active job postings to the Employer. The WebServer then submits a request with the job posting to delete to the EmployerDB.

Fig 43 shows the Sequence Diagram for the LMP data update scenario. The scenario is as follows:

- The JobCollector requests an update of internal job postings from DeptJobs and updates the cached job posting information
- The JobCollector requests an update of external job postings from ExternalJobs1 followed by a validation check request to the JobValidator.
- The JobCollector requests an update of external job postings from ExternalJobs2 followed by a validation check request to the JobValidator.
- The JobCollector updates the cached job postings with only validated jobs.

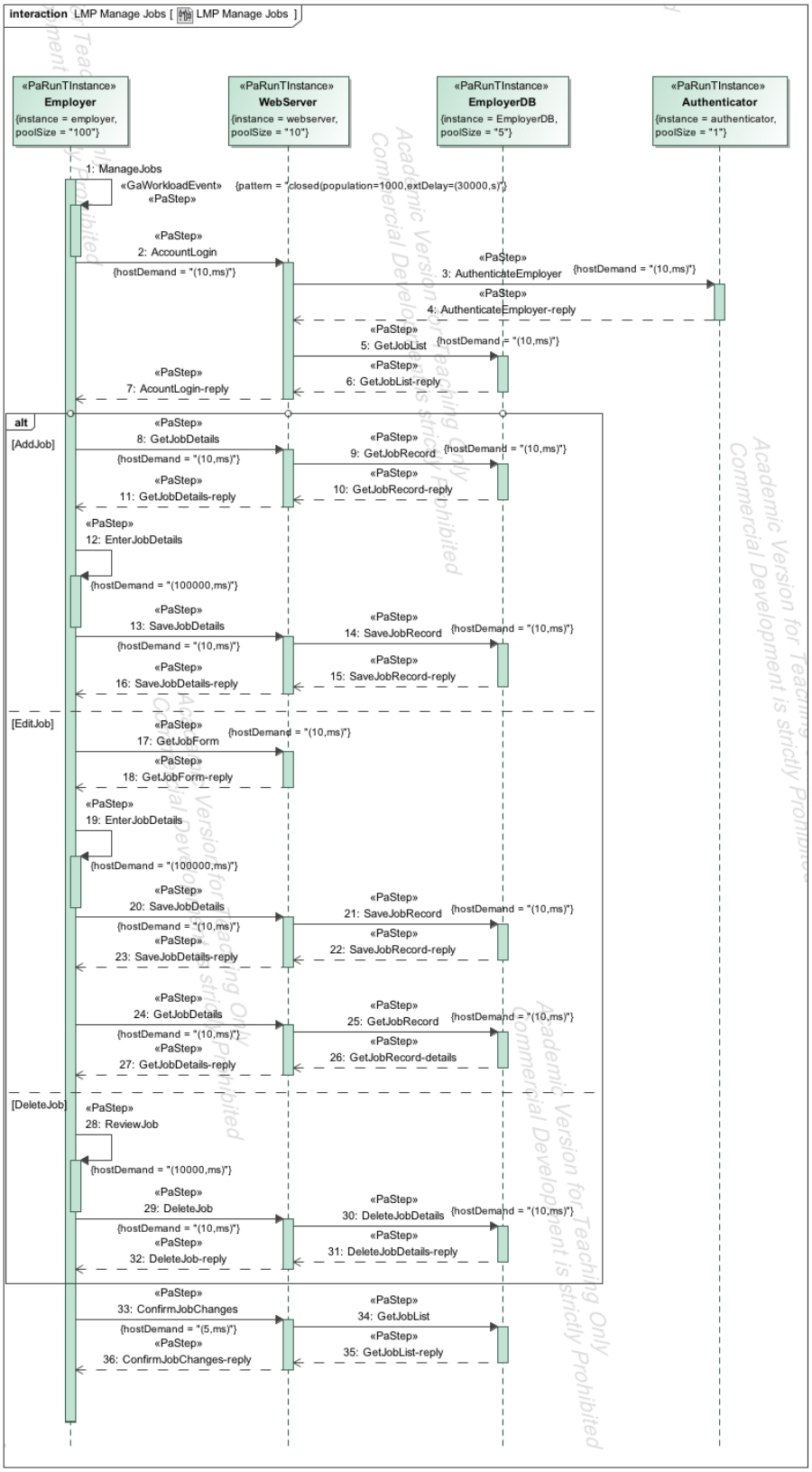


Fig 42 Sequence Diagram for the LMP Manage Jobs scenario

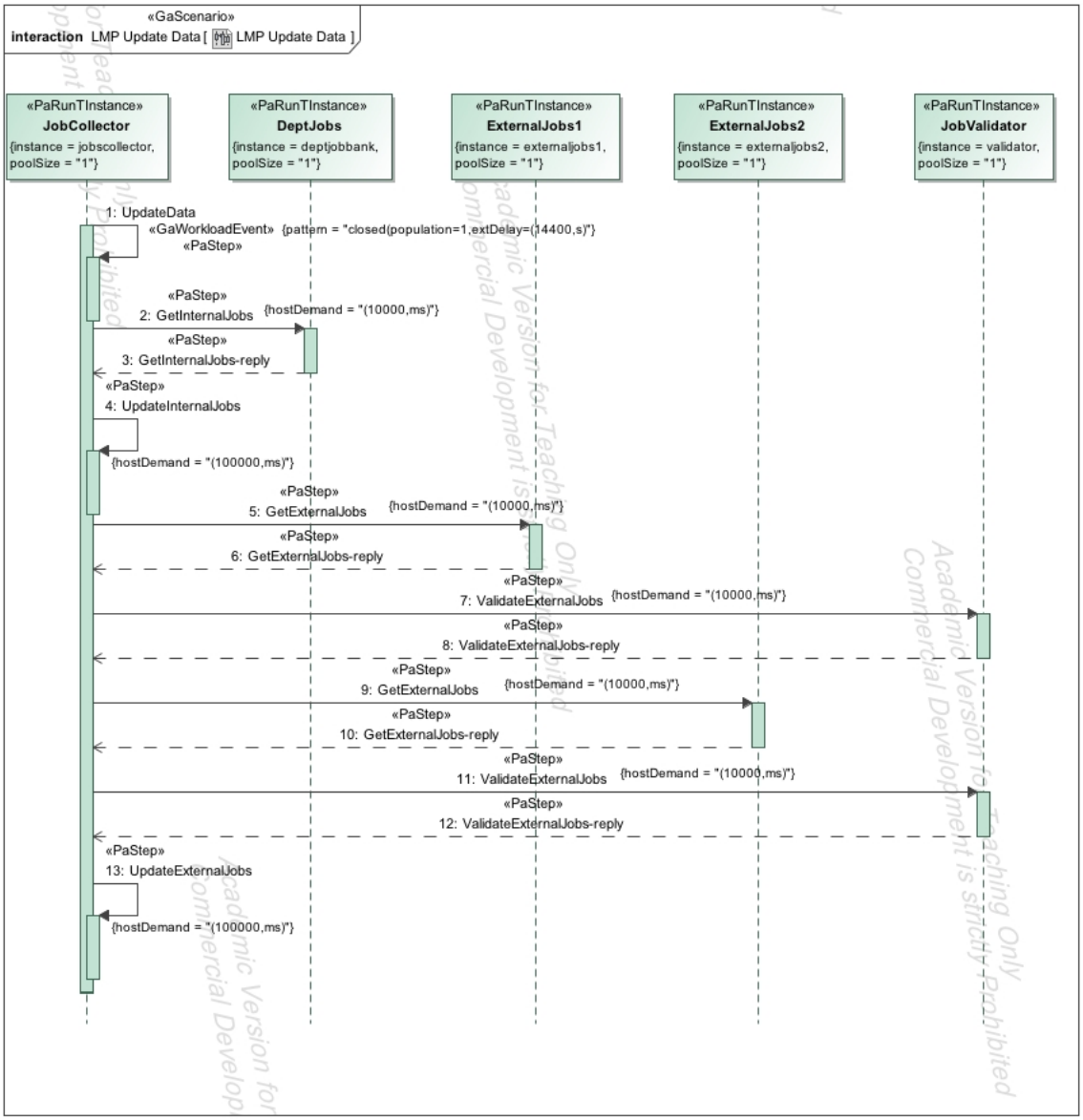


Fig 43 Sequence Diagram for the LMP Update Data scenario

7.3.3. CSM Model

DEPLOYMENT

The generated CSM represents the nodes with the following ProcessingResources:

```
<ProcessingResource id="PR00000" name="ExternalNode" multiplicity="1"
speedFactor="1.0"/>
<ProcessingResource id="PR00001" name="WebNode" multiplicity="1"
speedFactor="5.0"/>
<ProcessingResource id="PR00002" name="DatabaseNode" multiplicity="1"
speedFactor="10.0"/>
```

```
<ProcessingResource id="PR00003" name="ExternalDataNode" multiplicity="1"
speedFactor="1.0"/>
```

The processes are represented by the following components:

```
<Component id="CM00000" name="user" multiplicity="10000" host="PR00000"/>
<Component id="CM00001" name="webserver" multiplicity="10" host="PR00001"/>
<Component id="CM00002" name="reportgenerator" multiplicity="5"
host="PR00001"/>
<Component id="CM00003" name="statscollector" multiplicity="1" host="PR00002"/>
<Component id="CM00004" name="jobscollector" multiplicity="1" host="PR00002"/>
<Component id="CM00005" name="employer" multiplicity="100" host="PR00000"/>
<Component id="CM00006" name="authenticator" multiplicity="1" host="PR00002"/>
<Component id="CM00007" name="deptjobbank" multiplicity="1" host="PR00003"/>
<Component id="CM00008" name="externaljobs1" multiplicity="1" host="PR00003"/>
<Component id="CM00009" name="validator" multiplicity="1" host="PR00002"/>
<Component id="CM00010" name="externaljobs2" multiplicity="1" host="PR00003"/>
```

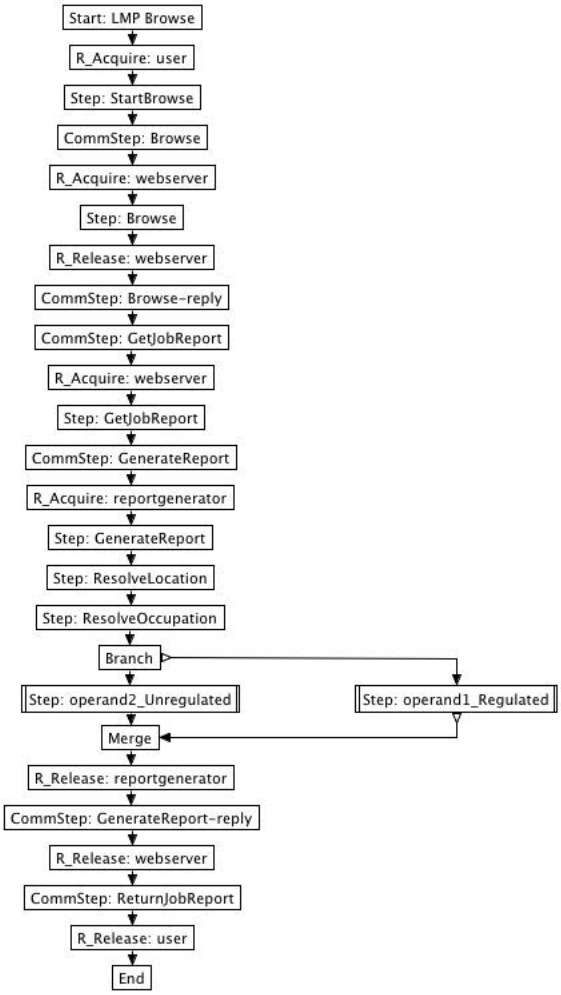


Fig 44 CSM for the main LMP Browse scenario

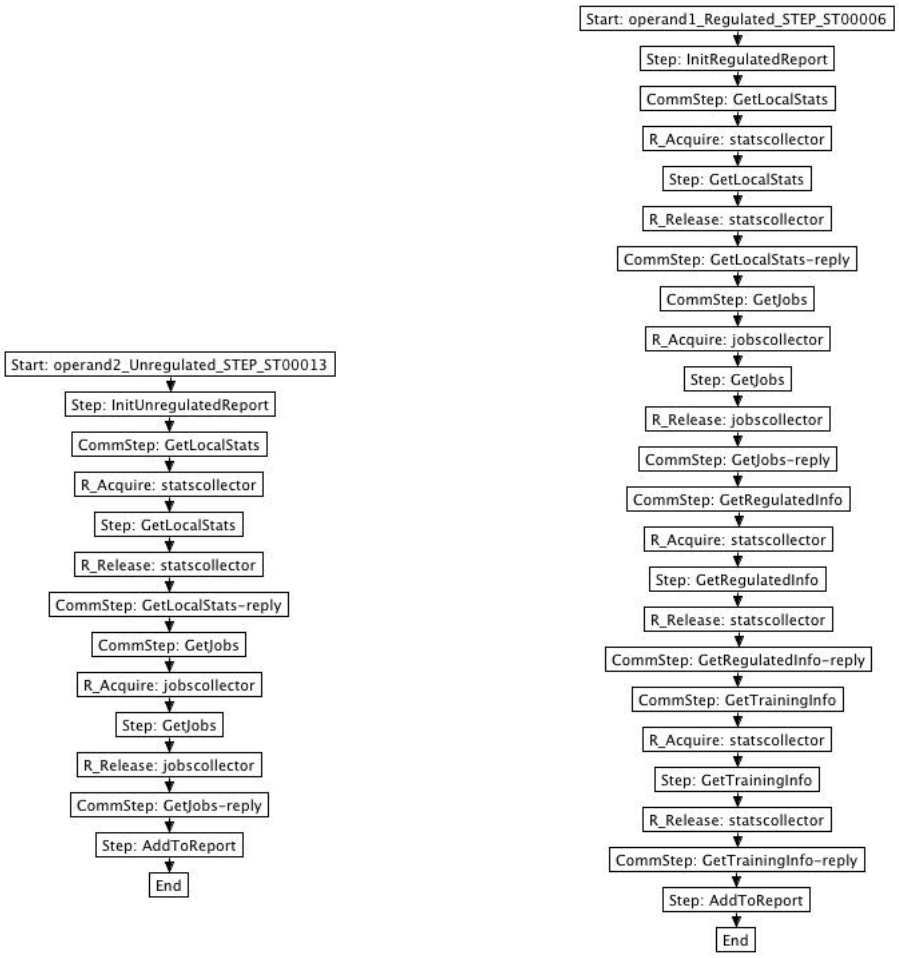


Fig 45 CSMs for the two alternate sub-scenarios for unregulated and regulated occupations in the LMP Browse scenario

BEHAVIOUR

The CSM scenarios for the behaviour are as follows:

- Fig 44 shows the main Browse by occupation scenario for job seekers
- Fig 45 shows the two Browse sub-scenarios corresponding to the "alt" interaction occurrences for regulated and unregulated occupations (UML is shown in Fig 40)
- Fig 46 shows the main Job Details scenario for job seekers
- Fig 47 shows the two sub-scenarios corresponding to the "alt" interaction occurrences for regulated and unregulated occupations in the Browse scenario (UML shown in Fig 41)

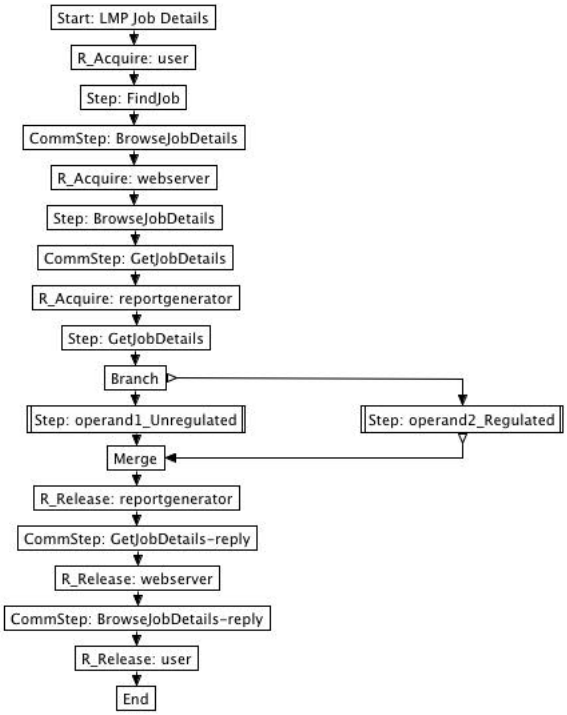


Fig 46 CSM for the main LMP Job Details scenario

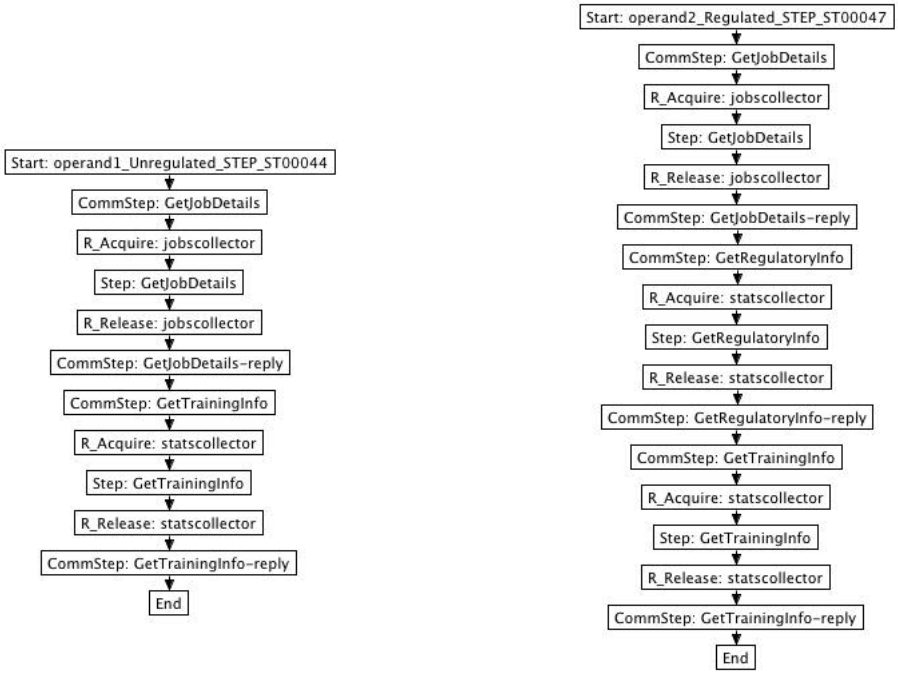


Fig 47 CSMs for the two alternate sub-scenarios for unregulated and regulated occupations in the LMP Job Details scenario

- Fig 48 shows show the main Manage Jobs scenario for employers
- Fig 49 shows the three Manage Jobs sub-scenarios corresponding to the "alt" interaction occurrences for adding a job, editing a job, and deleting a job as (UML shown in Fig 42)
- Fig 50 shows the scenario for the Update Date scenario periodically initiated by the LMP system (UML shown in Fig 43)

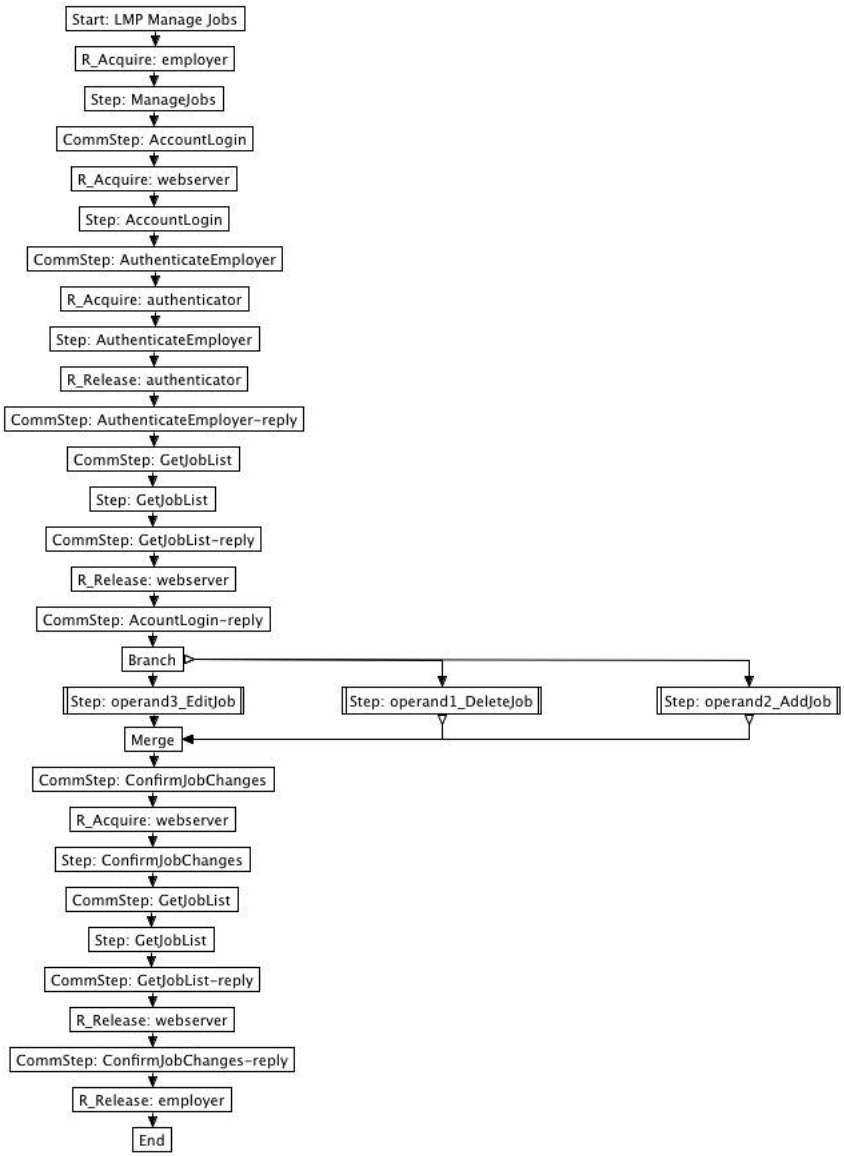


Fig 48 CSM for the main LMP Manage Jobs scenario for employers

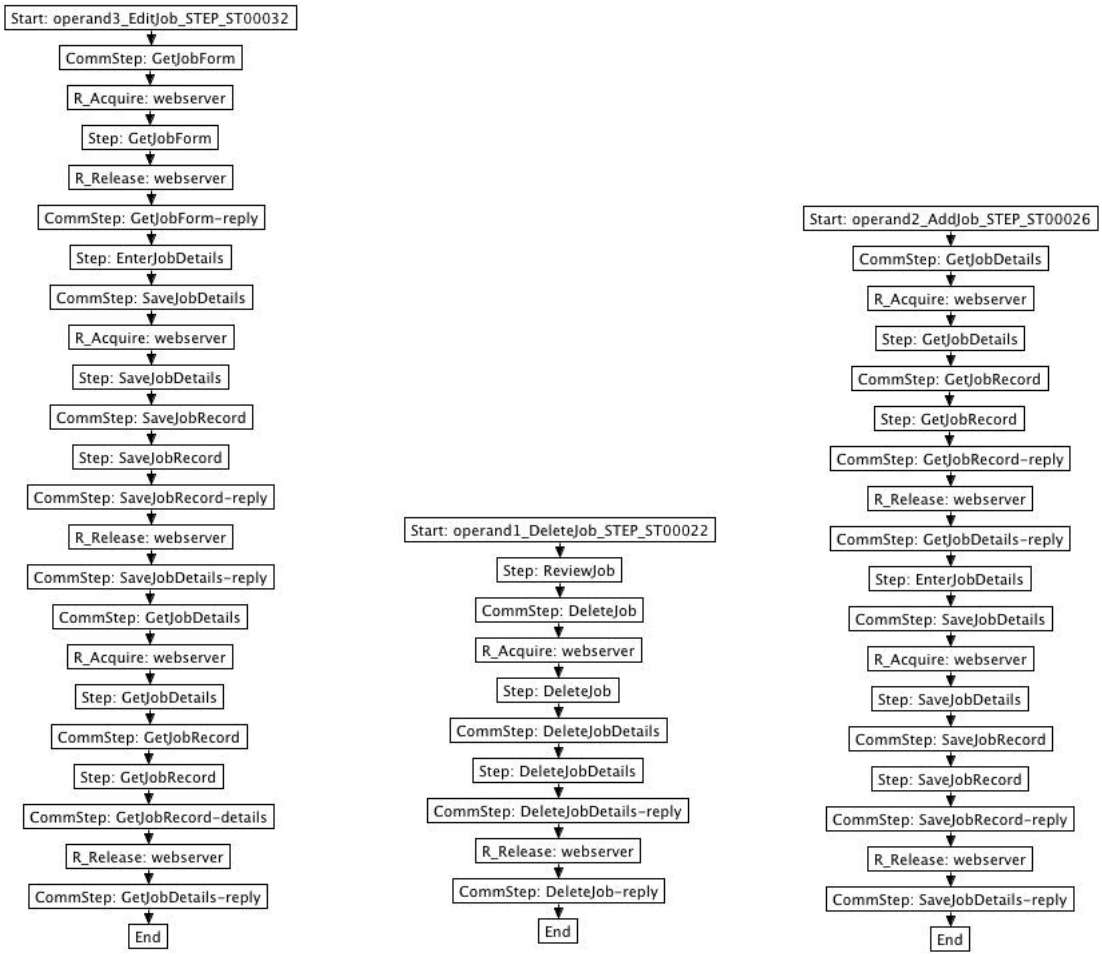


Fig 49 CSMs for the EditJob, DeleteJob and AddJob alternate sub-scenarios in the main Manage Jobs scenario

All separate interaction occurrences in the UML, whether part of an “alt” as used in this example or part of a “par”, “opt” or “loop” are generated as sub-scenarios in the CSM. The algorithms used to flatten the sub-scenarios in the CSM are:

- Create Unique Instances of All Sub- Scenarios (section 5.3.1),
- Assign Components to All Step Type Elements (section 5.3.2)
- Remove Duplicate ResourceAcquire and ResourceRelease Elements from Sub-Scenarios (section 5.3.4)
- Flatten Sub-Scenarios (section 5.3.5).

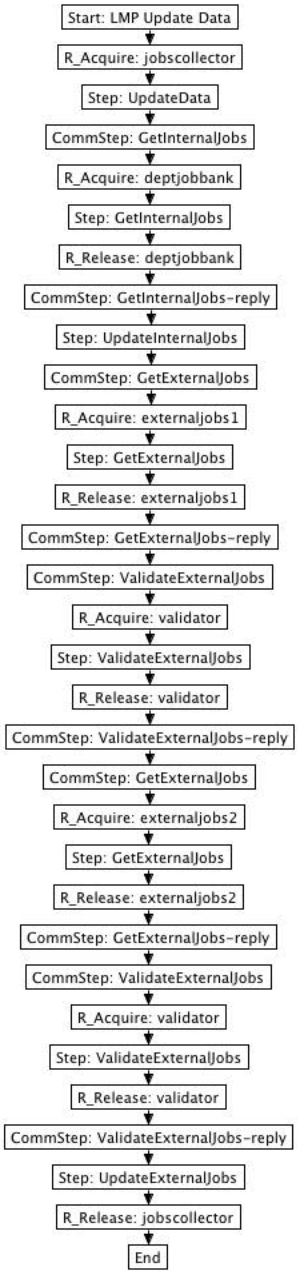


Fig 50 CSM for the LMP Update Data scenario

The CSMs for the LMP system are shown in the updated CSM v2 format which incorporates explicit CommSteps and is consistent with the MARTE profile.

In addition to the explicit CommSteps to model communication between components, the CSMs for the LMP leverage the known typing of messages in the source Sequence

Diagrams to minimize the occurrence of unnecessary ResourceAcquire and ResourceRelease elements (as was done with the TPC-W CSM in section 7.2.3).

7.3.4. Generated LQN Model

The LQN for the Labour Market Portal is shown in Fig 51. Due to the large size of the resulting model, it is hard to make out the individual LQN elements in the figure which was generated automatically using the lqn2ps tool. The figure is expanded into three larger parts for improved readability in the Appendix in section 10.2.

DEPLOYMENT

The LMP LQN model in Fig 51 has the following deployment for processors:

```
<processor name="InfiniteProc" scheduling="fcfs" speed-factor="1">
<processor name="ExternalNode_PR00000" scheduling="fcfs" speed-factor="1">
<processor name="WebNode_PR00001" scheduling="fcfs" speed-factor="5">
<processor name="DatabaseNode_PR00002" scheduling="fcfs" speed-factor="10">
<processor name="ExternalDataNode_PR00003" scheduling="fcfs" speed-factor="1">
```

The LQN tasks for the system are deployed as follows:

```
Deployed on InfiniteProc:
   <task name="browse" scheduling="ref" multiplicity="1000" think-time="1000">
   <task name="details" scheduling="ref" multiplicity="200" think-time="2000">
   <task name="manage" scheduling="ref" multiplicity="100" think-time="3000">
   <task name="update" scheduling="ref" think-time="200000">
Deployed on ExternalNode_PR00000:
   <task name="employer" scheduling="fcfs" multiplicity="100">
   <task name="user" scheduling="fcfs" multiplicity="10000">
Deployed on WebNode_PR00001:
   <task name="reportgenerator" scheduling="fcfs" multiplicity="5">
   <task name="webserver" scheduling="fcfs" multiplicity="10">
Deployed on DatabaseNode_PR00002:
   <task name="authenticator" scheduling="fcfs">
   <task name="jobscollector" scheduling="fcfs">
   <task name="statscollector" scheduling="fcfs">
   <task name="validator" scheduling="fcfs">
Deployed on ExternalDataNode_PR00003:
   <task name="deptjobbank" scheduling="fcfs">
   <task name="externaljobs1" scheduling="fcfs">
   <task name="externaljobs2" scheduling="fcfs">
```

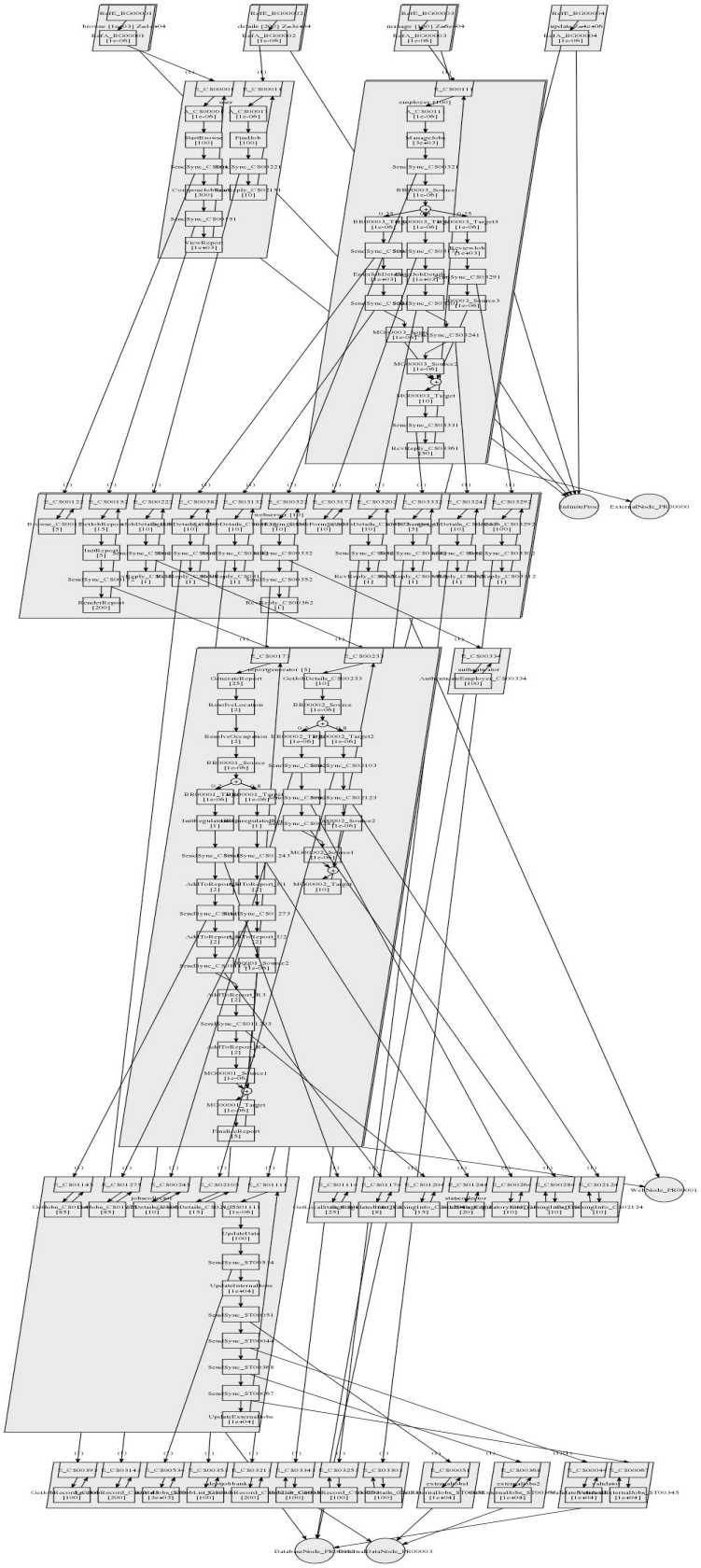


Fig 51 LQN for the Labour Market Portal

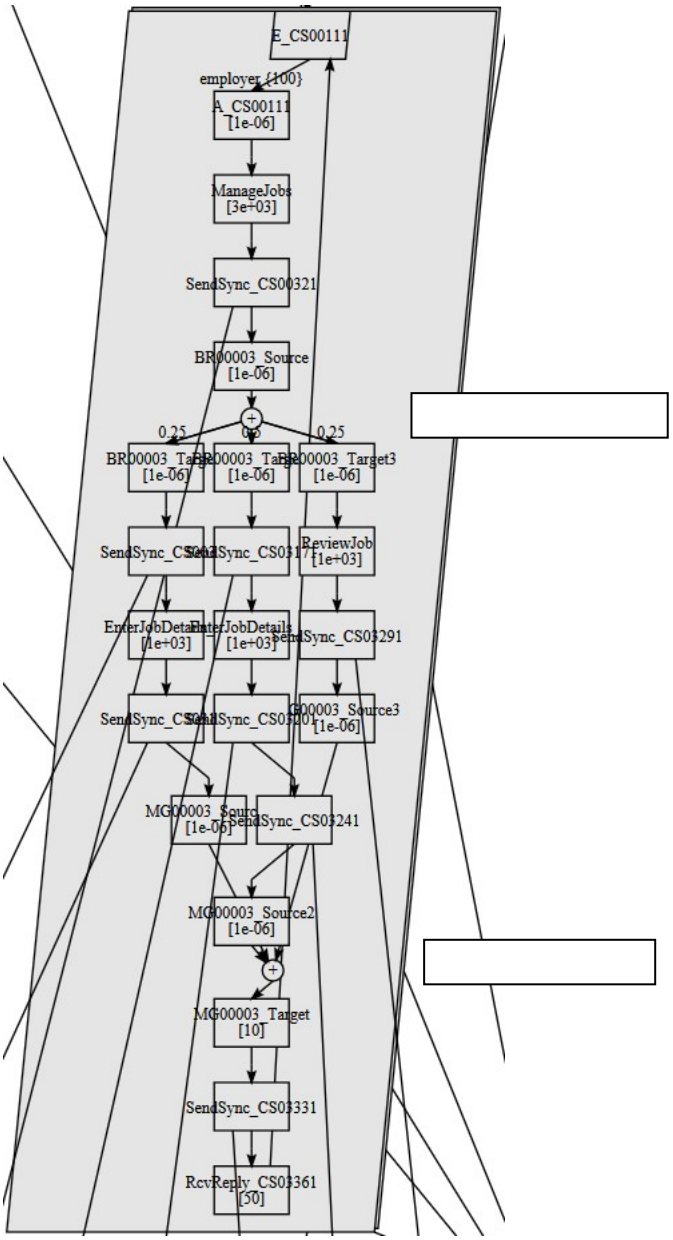


Fig 52 Close-up of the LQN activity graph for the employer task

BEHAVIOUR

The LQN reference tasks generate the workload for the four major LMP scenarios described in the UML and CSM sections. The 'browse' reference task generates the workload for the jobseeker occupational browsing scenario shown in Fig 40 and Fig 44.

The ‘details’ reference task generates the workload for the job posting details scenario shown in Fig 41 and Fig 46. The ‘manage’ reference task generates the workload for the employers managing their job postings scenario shown in Fig 42 and Fig 48. Finally, the ‘update’ reference task generates the workload for the data update scenario shown in Fig 43 and Fig 50.

The result of the CSM sub-scenario flattening algorithm can be clearly seen in the LQN activity graphs for the ‘employer’ and ‘reportgenerator’ tasks. Fig 52 shows a close-up of the ‘employer’ task. The activity graph shows the flattened three-way branching corresponding to the alternatives of whether the employer adds, edits or deletes a job posting. Similarly, the activity graph of the ‘reportgenerator’ task shows the alternate branching for regulated or unregulated occupations for both the occupational a browsing and job details scenarios.

7.3.5. Features Demonstrated

This example shows:

- The application of CSM transformations to a CSM that was generated from a Sequence Diagram
- CSM validation (section 5.1)
- CSM scenario traversal (section 5.2)
- CSM normalization (section 5.3)
    - create unique instances of all sub-scenarios (section 5.3.1)
    - assign components to all StepType elements (section 5.3.2)
    - check for non-nested component contexts (section 5.3.3)
    - remove duplicate ResourceAcquire and ResourceRelease elements in sub-scenarios (section 5.3.4)
    - flatten sub-scenarios (section 5.3.5)

    - clean up Forks with empty path segments (section 5.3.7)
    - clean up CommSteps (section 5.3.8)
    - clean up resource acquisition and release (section 5.3.9)
- Interaction discovery (section 5.4) of synchronous communication
- LQN generation (section 6.3)

8. CONCLUSIONS

This thesis has, first, defined the Core Scenario model (CSM) as an intermediate model to capture performance information from a software behaviour specification. Second, it has examined in detail the problem of transforming CSMs derived from UML annotated with MARTE performance stereotypes, into LQN performance models, and described successful algorithms for the transformation. Unlike previous attempts on this problem, it has resolved implicit synchronous inter-process communication which is a substantial potential source of delays. The transformations can also be applied to CSMs derived from any other software specifications. Also, the thesis has described the process for creating QN models from CSMs. A software tool based on Eclipse has been implemented based on the algorithms described here, and applied by the author to a number of examples, including one based closely on a currently deployed information system.

8.1. CONTRIBUTIONS

DEFINITION OF THE CORE SCENARIO MODEL (CSM)

CSM is an intermediate model that can capture the part of a software specification relevant for performance modeling, from a wide range of software modeling formalisms. CSM simplifies the creation of performance models (from any design model to any performance model) and overcomes the M×N problem inherent in trying to directly transform M source design models into N target performance models.

The CSM definition has three parts:

- A behaviour sub-model that captures the system behaviour in a scenario form.
- A resource sub-model that captures the system resources and their usage.

- A performance sub-model that captures performance requirements and can be used to feed results back from the performance model.

CSM was first designed to be consistent with the UML Profile for Schedulability, Performance and Timing (SPT). CSM was then updated to be consistent with the UML Profile for Modeling and Analysis of Real-Time and Embedded Systems (MARTE) that replaced SPT for UML 2.

The concepts behind CSM are also present in other approaches such as Execution Graphs (which preceded this work), KLAPER (which appeared at about the same time) and TIPM (which was derived from CSM and is used in industry). The strength of CSM is indicated by the fact that, using a small extension to CSM (e.g. adding the scheduling properties of resources), TIPM defines a simulation model that is used directly for evaluation.

Semantic Rules for Validation of CSM

This thesis contributes a set of semantic rules for validating the behaviour and resource handling parts of CSMs. One of the problems with deriving a CSM from a model intended for other purposes (software specification) is that the CSM may contain errors due to design errors, redundant information such as parallel branches that do nothing, or paths in which the resource context is unclear. The concept of a well-formed CSM, defined in chapter 3, provides semantic rules and constraints to define a normal form for CSM behaviour. The handling of resources by a program may also lead to various performance anomalies and is discussed as the system's "resource context" in chapter 4. A concept of "resource neutrality" has been defined to identify behaviours which are free

of these anomalies and is used to generate the appropriate type of task in the LQN target performance model.

Algorithms for Normalizing CSM

Another contribution of this thesis is the definition of and implementation in an automated tool of a set of algorithms for normalizing input CSMs so they are in a form that can be more easily automatically transformed into performance models. The algorithms are explained in detail in chapter 5. The algorithms do the following:

- Create unique instances of all sub-scenarios (section 5.3.1)
- Assign components to all StepType elements (section 5.3.2)
- Check for non-nested component contexts (section 5.3.3)
- Remove duplicate ResourceAcquire and ResourceRelease elements in sub-scenarios (section 5.3.4)
- Flatten sub-scenarios (section 5.3.5)
- Assign path segments to sequential scenario fragments (section 5.3.6)
- Clean up Forks with empty path segments (section 5.3.7)
- Clean up CommSteps (section 5.3.8)
- Clean up resource acquisition and release (section 5.3.9)
- Discover the types of interactions between components (section 5.4)

Additionally, an algorithm to weave in CSM aspect sub-models has also been defined and applied manually to a case study published in [127] and explained in detail in section 3.6. This algorithm has not been automated as part of this research.

Algorithms for Generating Performance Models from CSM

This thesis also contributes two algorithms to generate performance models from CSM. The algorithm to generate Layered Queueing Network (LQN) models from CSM is presented in detail in section 6.3 and implemented in the CSM2LQN automated

transformation tool. A set of test cases for CSM2LQN has also been developed and used to test the automated tool.

A second algorithm to generate Queueing Network (QN) models from CSM is presented in section 6.4 but has not implemented as an automated tool.

Automated Tool for Generating LQN Performance Models from CSM (CSM2LQN)

The algorithms for validating and normalizing CSM and for generating LQN performance models from CSM have been implemented as an automated tool called CSM2LQN. The tool is a Java application running on the Eclipse platform. J. Muttulingam provided assistance to the author in coding the algorithms into the CSM2LQN Eclipse application. The author worked closely on the coding and did all the testing.

The author has also collaborated by providing requirements and test cases for the development of other tools to automate the conversion of UML software specifications to CSMs. These tools have been implemented by previous students, and by Tauseef Israr, Andrew Miga, and Mohammad Alhaj, working as research engineers, so they are not directly part of this research.

Case Studies to Demonstrate the Application of the CSM2LQN Tool

The final contribution of this thesis is a set of three substantial case studies of service systems that were defined by the author in UML and annotated with performance data using either the SPT Profile or MARTE. The author then automatically generated CSMs from these UML models using the tools developed by Israr and Miga. Finally, the author automatically generated LQNs from the CSMs using the CSM2LQN tool.

The first case study is based on a building security system (BSS) and demonstrates:

- The automatic generation of a CSM from a UML Activity Diagram using MARTE performance annotations
- The application of the algorithms for CSM validation (section 5.1), CSM normalization (section 5.3), interaction discovery (section 5.4) of synchronous and asynchronous communication, and LQN generation (section 6.3) to the automatically generated CSM
- The handling of a non-nested resource architecture with a semaphore task in the generated LQN

The second case study is based on a transaction processing standard (TPC-W) and demonstrates:

- The automatic generation of a CSM from multiple UML Sequence Diagram using SPT performance annotations
- The application of the algorithms for CSM validation (section 5.1), CSM normalization (section 5.3), interaction discovery (section 5.4) of synchronous and asynchronous communication, and LQN generation (section 6.3) to the automatically generated CSM
- The use of Aspect-oriented CSM and LQN to estimate the effect of adding security aspects to a transaction processing system

The third case study is based on an enterprise-scale labour market information and job posting portal (LMP) and demonstrates:

- The automatic generation of a CSM from multiple UML Sequence Diagram using MARTE performance annotations
- The application of the algorithms for CSM validation (section 5.1), CSM normalization (section 5.3), interaction discovery (section 5.4) of synchronous and asynchronous communication, and LQN generation (section 6.3) to the automatically generated CSM
- The handling of multiple user scenarios in large-scale systems in the generated LQNs

PUBLICATIONS

The following published papers were co-authored by the author based on this thesis (listed in reverse chronological order of publication):

- Murray Woodside, D.C. Petriu, J. Merseguer, Dorin B. Petriu, Mohammad Alhaj, "Transformation challenges: from software models to performance models", Software and Systems Modeling, published online, DOI: 10.1007/s10270-013-0385-x, 2013 [128]
- Murray Woodside, D.C. Petriu, Dorin B. Petriu, J. Xu, T. Israr, G. Georg, R. France, J.M. Bieman, S.H. Houmb, J.Jürjens, "Performance Analysis of Security Aspects by Weaving Scenarios Extracted from UML Models", The Journal of Systems and Software Special Issue WOSP'2007, Vol.82, pp. 56–74, 2009 [125]
- D.C. Petriu, Murray Woodside, Dorin B. Petriu, J. Xu, T. Israr, G. Georg, R. France, J. Jürjens, J.M. Bieman, S.H. Houmb, "Performance Analysis of Security Aspects in UML Models", Proc. of 6th ACM Workshop on Software and Performance, pp. 91-102, Buenos Aires, Argentina, February 5-8, 2007 [100]
- Dorin B. Petriu, Murray Woodside, "An Intermediate Metamodel with Scenarios and Resources, for Generating Performance Models from UML Designs", SoSym Special Issue on UML'04, Software and Systems Modeling (SoSyM), Springer-Verlag, August 2006 [96]
- Dorin B. Petriu, Murray Woodside, "Software Performance Models from System Scenarios", Journal on Performance Evaluation (PEVA), vol. 61, issue 1, pp. 65-89 Elsevier, June 2005 [95]
    - appeared on the ScienceDirect "Top 25 Hottest Articles for Performance Evaluation" list
        - #1 Apr-Jun 2005
        - #7 Jul-Sep 2005
        - #17 Oct-Dec 2005
        - #23 Jan-Mar 2006

- Murray Woodside, D.C. Petriu, Dorin B. Petriu, H. Shen, T. Israr, J. Merseguer, “Performance by Unified Model Analysis (PUMA)”, Proc. WOSP 2005, Palma de Mallorca, pp. 1-12, July 2005 [125]
- Dorin B. Petriu, Daniel Amyot, Murray Woodside, B. Jiang, “Traceability and Evaluation in Scenario Analysis by Use Case Maps”, in Scenarios, LNCS 3466, pp. 134-151, Springer-Verlag, 2005 [94]
- Dorin B. Petriu, Murray Woodside, "A Metamodel for Generating Performance Models from UML Designs, Proc UML 2004, Lisbon, LNCS 3273, pp. 41-53, Springer-Verlag, October 2004 [93]

IMPACT OF THE RESEARCH

CSM and the research presented in this thesis have already made a significant impact on academic research, standards and industrial practice.

The following graduate students have used CSM in their research (listed in chronological order of thesis publication):

- Y.X. Zeng, “Transforming Use Case Maps to the Core Scenario Model Representation”, SITE, University of Ottawa, June 2005. [132]
- Toqeer Israr, "Transformation of UML 2.0 models into Core Scenario Models", M.App.S. thesis, Carleton University, Department of Systems and Computer Engineering, September 2005. [49]
- S. Maqbool, “Transformation of a Core Scenario Model and Activity Diagrams into Petri Nets”, M.Sc. thesis, University of Ottawa, September 2005. [69]
- Hui Liu, "Transformation of UML 2.0 Models Extended with MARTE to Core Scenario Models", M.S. Thesis, Carleton University, Department of Systems and Computer Engineering, June 2008. [63]
- Omar Mahmoud, "Rule-based transformation from UML+MARTE to CSM”, M.A.Sc. thesis, Carleton University, Department of Systems and Computer Engineering, Sept. 2009. [67]

- M. Fritzsche, “Performance related Decision Support for Process Modelling”, PhD Thesis, School of Electronics, Electrical Engineering and Computer Science, Queens University Belfast, 2010. [42]
- Mohammad Alhaj, "Automatic Derivation of performance Models in the Context of Model-Driven SOA", Ph.D. thesis, Carleton University, Department of Systems and Computer Engineering, January 2014. [1]

CSM has been incorporated in the ITU-T Z.150 standardization of the User Requirements Notation (URN) [52] and the associated jUCMNav tool [146]. CSM is included in the roadmap for the continued development of URN [3].

CSM is also included in a commercial Model-Driven Business Impact Analysis (MDBIA) workbench implemented by SAP [104].

Cortellessa includes CSM as one of the main performance meta-models in his survey of the state of the art in the field of software performance engineering in [27].

8.2. FUTURE WORK

The problem of generating LQNs from CSMs is solved in this work. However some additional work could be usefully done to round out the PUMA toolset and to address some of the threats to the validity of the research.

ADDRESSING THREATS TO VALIDITY

There are three principal threats to the validity of the research in this thesis.

The first threat comes from the close alignment of CSM with UML and MARTE. While this alignment enhances CSM’s fit and suitability as an intermediate model for extracting performance information from UML specifications, it may limit the type of information that CSM can express.

The second threat is related to the implementation of software design tools. Such tools are usually complex, integrate a wide range of functionality, and see rapid development. This has a side effect on the stability of their internal data. Their internal object models and the file formats used to save models also tend to change quickly which can interfere with any tools that are built on their platform. This has been a challenge during the development of tools to generate CSMs and poses a concern as to the viability of the automated tools associated with PUMA, including CSM2LQN.

The third threat is related to the tools available, in that the tests and case studies presented in this thesis were all developed by the author and a small number of collaborators in the same research group. This close collaboration reduces the breadth and variety of examples used in the research and leads to a risk that the solutions proposed are less universal than the author believes.

Additional work is needed to better address these threats to the validity of the research and to create a healthy and varied research ecosystem for the continued development of PUMA and CSM.

Topics for Continued Research

In addition to future work to address threats to validity, the following are interesting topics for future research:

- using CSM to facilitate the traceability and feedback of performance results to the design specification
- additional research on resource neutrality and more sophisticated treatments for resolving resource contexts at a Join, indicated in Section 4.2, could also be part of a deeper look into the role of resources in software.
- further extensions to CSM to allow for:

    - additional experimentation with AO-CSM to include aspects that are woven in at multiple points
    - using path segmentIDs to handle “informal” loops in Activity Diagrams where the activities create a repeating cyclical pattern
    - generation of CSM models from other software notations such as Architectural Description Language (ADL)
    - automatic generation of QNs in PMIF format
- additional experiments to validate the obtained models (compared to real-world examples if possible)
- better proofs that the CSM output of all the algorithms is consistent with the original UML models
- additional CSM case studies from scenario specifications other than UML and MARTE.

9. REFERENCES

** All reference URLs were visited and verified to be live on September 17, 2014*

[1] Mohammad Alhaj, "Automatic Derivation of performance Models in the Context of Model-Driven SOA", Ph.D. thesis, Carleton University, Department of Systems and Computer Engineering, January 2014.

[2] D. Amyot, E. Eberlein, “An Evaluation of Scenario Notations and Construction Approaches for Telecommunication Systems Development”, Telecommunication Systems Journal, Volume 24, No. 1, pp. 61 – 94, September 2003.

[3] D. Amyot, G. Mussbacher, “User Requirements Notation: The First Ten Years, The Next Ten Years”, Journal of Software, May 2011, pp. 747-768, DOI:10.4304/jsw.6.5.747-768

[4] J.H. Andrews, “Process-Algebraic Foundations of Aspect-Oriented Programming”, REFLECTION 2001, (A. Yonezawa and S. Matsuoka, Eds.), LNCS 2192, pp.187–209, Springer-Verlag, 2001.

[5] V. Apostolopoulos, V. Peris, D. Saha, “Transport layer security: How much does it really cost?", Conference on Computer Communications (IEEE Infocom), pp. 717-725, New York, March 1999.

[6] S. Balsamo, A. Di Marco, P. Inverardi, M. Simeoni, “Model-based performance prediction in software development: a survey”, IEEE Transactions on Software Engineering, Vol 30, No.5, pp.295-310, May 2004.

[7] S. Balsamo, P. Inverardi, C. Mangano, “An Approach to Performance Evaluation of Software Architectures”, in Proc. of First International Workshop on Software and Performance (WOSP98), October 1998, pp. 178-190.

[8] S. Balsamo, M. Marzolla. “Simulation Modeling of UML Software Architectures”, Proc. ESM'03, Nottingham (UK), June 2003

[9] S. Balsamo, M. Simeoni, “On transforming UML models into performance models”, Workshop on Transformations in the Unified Modeling Language, Genova, Italy, April 2001.

[10] J.P. Barros, L. Gomes, “Towards the Support for Crosscutting Concerns in Activity Diagrams: a Graphical Approach”, Fourth Workshop on Aspect-Oriented Modeling with UML, San Francisco, 2003.

[11] L. Bass, P. Clements, R. Kazman, “Software Architecture in Practice”, Addison-Wesley, 1998.

[12] F. Bause, H. Kabutz, P. Kemper, P.S. Kritzinger, “SDL and Petri net performance analysis of communicating systems”, in *PSTV*, pp. 269-282. 1995.

[13] S. Becker, H. Koziolek, R. Reussner, “Model Based Performance Prediction with the Palladio Component Model”, Proc. 6th ACM Int. Workshop on Software and Performance, pp. 54-65, Buenos Aires, Feb. 2007.

[14] S. Becker, H. Koziolek, R. Reussner, “The Palladio component model for model-driven performance prediction”, Journal of Systems and Software, vol. 82, no. 1, pp. 3-22, Jan 2009.

[15] H. Beilner, J. Mäter, C. Wysocki, “The hierarchical evaluation tool HIT”, in *Short Papers and Tool Descript. of 7th Int. Conf. on Modelling Techniques and Tools for Computer Perf. Evaluation*, 1994.

[16] S. Bernardi, S. Donatelli, and J. Merseguer, “From UML sequence diagrams and statecharts to analyzable Petri net models”, in Proc. 3rd Int. Workshop on Software and Performance (WOSP02), Rome, July 2002, pp. 35-45.

[17] G. Booch, J. Rumbaugh, and I. Jacobson, “The Unified Modeling Language User Guide”, Addison-Wesley, 1998.

[18] R.J.A. Buhr, “Use Case Maps as Architectural Entities for Complex Systems”, IEEE Transactions on Software Engineering, vol. 24, no. 12 pp. 1131 - 1155, 1998.

[19] R.J.A. Buhr, R.S. Casselman, “Use Case Maps for Object-Oriented Systems”, Prentice-Hall, 1995.

[20] R.J.A. Buhr, G.M. Karam, C.J. Hayes, C.M. Woodside, “Software CAD: A Revolutionary Approach”, IEEE Trans. on Software Engineering, Vol. 15, No. 3, pp. 235-249, 1989.

[21] M. Bütow, M. Mestern, C. Schapiro, P.S. Kritzinger, “Performance Modeling with the Formal Specification Language SDL”, in Formal Description Techniques IX, R. Gotzhein and J. Brdereke, eds., IFIP, Chapman & Hall, pp. 213-228, October 1996.

[22] J.M. Carroll, “Making Use: Scenario-based Design of Human-Computer Interactions”, MIT Press, Cambridge, MA, 2000.

[23] J.M. Carroll, M.B. Rosson, G. Chin, J. Koenemann, “Requirements Development in Scenario-Based Design”, IEEE Transactions on Software Engineering, Vol. 24, No. 12, pp. 1156-1170, Dec. 1998

[24] C. Cavenet, S. Gilmore, J. Hillston, L. Kloul, P. Stevens, “Analysing UML 2.0 activity diagrams in the software performance engineering process”, in Proc. 4th Int. Workshop on Software and Performance (WOSP 2004), Redwood City, CA, pp. 74-83, Jan 2004.

[25] L. Chung, B. A. Nixon, E. Yu, J. Mylopoulos, “Non-Functional Requirements in Software Engineering”, Boston, Kluwer, 2000.

[26] V. Cortellessa, R. Mirandola, “Deriving a Queueing Network based Performance Model from UML Diagrams”, in Proc. Second Int. Workshop on Software and Performance (WOSP2000), Ottawa, Canada, pp. 58-70, September 2000.

[27] V. Cortellessa, “How far are we from the definition of a common software performance ontology?”, WOSP’05: Proceedings of the 5th International Workshop on Software and Performance, pp. 195–204, New York, ACM Press, 2005.

[28] V. Cortellessa, A. Di Marco, P. Inverardi, “Model-Based Software Performance Analysis”, Springer, 2011.

[29] L. Dai, K. Cooper, “Modeling and performance analysis for security aspects”, Science of Computer Programming, v 61 pp. 58-71, 2006.

[30] M. Diefenbruch, J. Hintelmann, B. Müller-Clostermann, “QSDL: Language and tools for performance analysis of SDL systems”, Conference Volume for the 5th GI/ITG Specialist Discussion on Formal description techniques for distributed systems, Kaiserslautern, 1995.

[31] S. DiStefano, M. Scarpa, A. Puliafito, "From UML to Petri Nets: The PCM-Based Methodology”, IEEE Trans. on Software Engineering, Vol. 37, Issue 1, pp.65-79, Jan. 2010

[32] S. Donatelli, G. Franceschinis, “PSR Methodology: integrating hardware and software models”, Proc. 17th Int. Conf. on Application and Theory of Petri Nets (Osaka, Japan), LNCS vol. 1091, Springer, June 24-28 1996.

[33] DSTC et al, MOF Query/Views/Transformations: Second Revised Submission, OMG document ad/04-01-06, Jan. 2004.

[34] W. Dulz, S. Gruhl, L. Lambert, M. Sollner, “Early performance prediction of SDL/MSC specified systems by automated synthetic code generation”, Proc. of the Ninth SDL Forum (SDL'99), Montreal, Canada, Elsevier, 1999.

[35] H. Espinoza, H. Dubois, S. Gerard, J. Medina, D.C. Petriu, M. Woodside, “Annotating UML Models with Non-Functional Properties for Quantitative Analysis”, in MoDELS 2005 Workshops (Jean-Michel Bruel, Ed.), LNCS 3844, pp. 79-90, Springer-Verlag, 2006.

[36] R. France, I. Ray, G. Georg, S. Ghosh, “An Aspect-Oriented Approach to Early Design Modeling”, IEEE Proceedings - Software, Special Issue on Early Aspects: Aspect-Oriented Requirements Engineering and Architecture Design, 151(4):173-185, August 2004.

[37] G. Franks “Performance Analysis of Distributed Server Systems”, Ph.D. Thesis, Carleton University, Systems and Computer Engineering, Report OCIEE-00-01, Jan. 2000.

[38] G. Franks, S. Majumdar, J. Neilson, D. Petriu, J. Rolia, M. Woodside, “Performance Analysis of Distributed Server Systems”, Proc. Sixth International Conference on Software Quality (6ICSQ), Ottawa, Ontario, pp. 15-26, 1996.

[39] G. Franks, M. Woodside, “Effectiveness of early replies in client-server systems”, Proceedings of Performance 99, Performance Evaluation, vol. 36--37, pp. 165-183, August 1999.

[40] G. Franks, M. Woodside, “Performance of Multi-level Client-Server Systems with Parallel Service Operations”, Proceedings of Workshop on Software Performance (WOSP98), Santa Fe, October 1998, pp. 120-130.

[41] M. Fritzsche, J. Johannes, S. Zschaler, A. Zherebtsov, A. Terekhov, “Application of Tracing Techniques in Model-Driven Performance Engineering”, 4th ECMDA Traceability Workshop (2008), pp. 1-10, 2008

[42] M. Fritzsche, “Performance related Decision Support for Process Modelling”, PhD Thesis, School of Electronics, Electrical Engineering and Computer Science, Queens University Belfast, 2010.

[43] V. Grassi, R. Mirandola, A. Sabetta, “From Design to Analysis Models: a Kernel Language for Performance and Reliability Analysis of Component-based Systems”, Proc. of the 5th International Workshop on Software and Performance (WOSP 2005), pp. 25–36, Palma, Illes Balears, Spain, 2005.

[44] W.M. Ho, J-M. Jézéquel, F. Pennaneac'h, N. Plouzeau, “A Toolkit for Weaving Aspect Oriented UML Designs”, Proc. of the 1st Int. Conference on Aspect-Oriented Software Development AOSD’2002, pp.99-105, Enschede, The Netherlands, 2002.

[45] S.H. Houmb, G. Georg, “The Aspect-Oriented Risk-Driven Development (AORDD) Framework”, In O. Benediktsson et al., editor, Proc. of the International Conference on Software Development (SWDC.REX), pp. 81-91, Reykjavik, Iceland, 2005. Gutenberg.

[46] S.H. Houmb, J. Jürjens, G. Georg, R. France, “An integrated security verification and security solution trade-off analysis”, In Integrating Security and Software Engineering: Advances and Future Vision. Mouratidis, H. and Giorgini, P. (eds). Idea Group Inc. To be published in 2006.

[47] ISO/IEC 15909-1:2004, “Software and system engineering – High-level Petri nets – Part 1: Concepts, definitions and graphical notation”, 2004.

[48] Tauseef Israr, “A Lightweight Technique for Extracting Software Architecture and Performance Models from Traces”, MASc thesis, April 2001.

[49] Toqeer Israr, "Transformation of UML 2.0 models into Core Scenario Models", M.App.S. thesis, Carleton University, Department of Systems and Computer Engineering, September 2005.

[50] ITU-T, Recommendation Z.100, Specification and Description Language (SDL), Geneva, 2000.

[51] ITU-T, Recommendation Z. 120, Message Sequence Charts (MSC), Geneva 2001.

[52] ITU-T, Recommendation Z.150, User Requirements Notation (URN), Geneva, 2003. http://www.UseCaseMaps.org/urn/

[53] ITU-T, URN Focus Group, Draft Rec. Z.152 - UCM: Use Case Map Notation (UCM), Geneva, 2002.

[54] R. Jain, “The Art of Computer Systems Performance Analysis”, John Wiley & Sons Inc., 1991.

[55] D. Jansen, H. Hermanns, “QoS Modeling with UML StateCharts”, Proc ACM Sigmetrics, 32 (2005), pp. 28–33.

[56] J. Jürjens, “Secure systems development with UML”, Springer-Verlag, Berlin Heidelberg, 2004.

[57] J. Jürjens, “Sound Methods and Effective Tools for Model-based Security Engineering with UML”, 27th International Conference on Software Engineering (ICSE 2005), St. Louis, Missouri, USA, pp. 322-331, 2005.

[58] P. Kahkipuro, “UML-Based Performance Modeling Framework for Component-Based Systems”, in Performance Engineering, R. Dumke, C. Rautenstrauch, A. Schmietendorf, and A. Scholz, Eds. Berlin, Springer, pp. 167-184, 2001.

[59] L. Kerber, “Scenario-Based Performance Evaluation of SDL/MSC-Specified Systems”, in Performance Engineering, ed. R. Dumke, C. Rautenstrauch, A. Schmietendorf, and A. Scholz, Springer, 2001, pp. 185 – 201.

[60] R.H. Khan, P.E. Heegaard, “Translation from UML to Markov model: A performance modeling framework for managing behavior of multiple collaborative sessions and instances”, 3rd IEEE Int. Conf. on Computer Science and Information Technology (ICCSIT), pp. 677 - 686, 2010.

[61] J. Klein, F. Fleurey, J-M. Jézéquel, “Weaving Multiple Aspects in Sequence Diagrams”, accepted for publication in Transactions on Aspect-Oriented Software Development, 2007.

[62] O. Lassila, R.R. Swick (eds), Resource Description Framework (RDF) Model and Syntax Specification, W3C Recommendation 22, World Wide Web Consortium, Feb. 1999.

[63] Hui Liu, "Transformation of UML 2.0 Models Extended with MARTE to Core Scenario Models", M.S. Thesis, Carleton University, Department of Systems and Computer Engineering, June 2008.

[64] J. Liu, Real-Time Systems, Prentice Hall, 2000.

[65] J. P. López -Grao, J. Merseguer, and J. Campos, “From UML Activity Diagrams To Stochastic Petri Nets: Application To Software Performance Engineering”, in Fourth Int. Workshop on Software and Performance (WOSP 2004), Redwood City, CA, Jan. 2004, pp. 25-36.

[66] M. Mahoney, A. Bader, T. Elrad, O. Aldawud “Using Aspects to Abstract and Modularize Statecharts”, Proc. 5th Wsh. Aspect-Oriented Modeling, Lisboa, 2004.

[67] Omar Mahmoud, "Rule-based transformation from UML+MARTE to CSM”, M.A.Sc. thesis, Carleton University, Department of Systems and Computer Engineering, Sept. 2009.

[68] P. Maly, “Description of the LQN XML Schema”, http://www.sce.carleton.ca/rads/lqns/lqn-documentation/schema/

[69] S. Maqbool, “Transformation of a Core Scenario Model and Activity Diagrams into Petri Nets”, M.Sc. thesis, University of Ottawa, September 2005.

[70] M. Marzolla, S. Balsamo, “UML-PSI: The UML Performance Simulator”, in Proc. First Int. Conf. on The Quantitative Evaluation of Systems (QEST'04), Enschede, pp. 340-341, 2004.

[71] D. Menasce, H. Gomaa, “A Method for Design and Performance Modeling of Client/Server Systems”, IEEE Transactions on Software Engineering, vol. 26, no. 11 pp. 1066-1085, 2000.

[72] D. Menascé, “Security Performance”, IEEE Internet Computing, vol. 7, nb. 3, pp. 84-87, May/June 2003.

[73] J. Merseguer, “Software performance engineering based on UML and Petri nets”, Ph.D. thesis, University of Zaragoza, Spain, March 2003.

[74] A. Miga, Application of Use Case Maps to System Design with Tool Support, M.Eng. thesis, Dept. of Systems and Computer Engineering, Carleton University, Ottawa, Canada, 1998.

[75] A. Miga, D. Amyot, F. Bordeleau, D. Cameron, M. Woodside, "Deriving Message Sequence Charts from Use Case Maps Scenario Specifications", in Proc. SDL Forum, Copenhagen, June 2001.

[76] A. Mitschele-Thiel, Systems Engineering with SDL: Developing Performance-Critical Communication Systems, Wiley, 2001.

[77] G. Mussbacher, D. Amyot, M. Weiss, "Visualizing Aspect-Oriented Requirements Scenarios with Use Case Maps", International Workshop on Requirements Engineering Visualization (REV 2006), IEEE CS, 4th paper. DOI: 10.1109/REV.2006.9, Minneapolis, Minnesota, USA, September 2006.

[78] G. Mussbacher, D. Amyot, M. Weiss, "Visualizing Early Aspects with Use Case Maps", Rashid, A. and Aksit, M. (Eds.), Transactions on Aspect-Oriented Software Development (TAOSD) III, Springer, LNCS 4620:105-143. DOI: 10.1007/978-3-540-75162-5_5, 2007.

[79] G. Mussbacher, D. Amyot, "Goal and scenario modeling, analysis, and transformation with jUCMNav", 31st International Conference on Software Engineering – Companion Volume, pp. 431-432, IEEE, 2009.

[80] G. Mussbacher, "Aspect-oriented User Requirements Notation", PhD thesis, University of Ottawa, Ottawa, ON, Canada, 2010.

[81] J.E. Neilson, C.M. Woodside, D.C. Petriu and S. Majumdar, "Software Bottlenecking in Client-Server Systems and Rendez-vous Networks", IEEE Transactions On Software Engineering, Vol. 21, No. 9, pp. 776-782, September 1995

[82] B.A. Nixon, "Management of Performance Requirements for Information Systems", IEEE Trans. on Software Eng., vol. 26, no. 12 pp. 1122-1146, 2000.

[83] Object Management Group, Unified Modeling Language Superstructure, Version 2.4.1, OMG Document Number: formal/2011-08-06, 2011.

[84] Object Management Group, Meta Object Facility (MOF) 2.0 Core Specification, OMG Document Number: formal/2006-01-01, 2006.

[85] Object Management Group, Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification, Version 1, OMG Document Number: formal/2008-04-03, 2008.

[86] Object Management Group, UML Profile for Schedulability, Performance, and Time Specification, Version 1.1, OMG Document Number: formal/05-01-02), January, 2005.

[87] Object Management Group, UML Profile for Modeling and Analysis of Real-Time and Embedded systems (MARTE), Version 1.1, OMG Document Number: formal/2011-06-02, Jun. 2011.

[88] Dorin B. Petriu, "Layered Software Performance Models Constructed from Use Case Map Specifications", M.Eng. Thesis, Carleton University, May 2001.

[89] Dorin B. Petriu, CSM v1 schema, http://www.sce.carleton.ca/rads/puma/csm/csm-1-schema/

[90] Dorin B. Petriu, A. Miga, CSM v2 schema, http://www.sce.carleton.ca/rads/puma/csm/csm-2-schema/

[91] Dorin B. Petriu, M. Woodside, "Incorporating Performance Analysis in the Early Stages of Software Development Using Generative Programming Principles", in Third Workshop on Generative Programming (ECOOP 2001 Workshop), Budapest, June 2001.

[92] Dorin B. Petriu, M. Woodside, "Software Performance Models from System Scenarios in Use Case Maps", in Proc. 12th Int. Conf. on Modeling Tools and Techniques (TOOLS 2002), London, England, April 2002.

[93] Dorin B. Petriu, M. Woodside, “A Metamodel for Generating Performance Models from UML Designs”, in Proc. UML 2004, Lisbon, v. 3273 of Lecture Notes in Computer Science (LNCS 3273), pp. 41-53, Oct 2004.

[94] Dorin B. Petriu, Daniel Amyot, Murray Woodside, B. Jiang, “Traceability and Evaluation in Scenario Analysis by Use Case Maps”, in Scenarios, LNCS 3466, pp. 134-151, Springer-Verlag, 2005

[95] Dorin B. Petriu, M. Woodside, “Software Performance Models from System Scenarios”, in Performance Evaluation, vol. 61, issue 1, pp. 65-89, Elsevier, Amsterdam, June 2005.

[96] Dorin B. Petriu, M. Woodside, “An Intermediate Metamodel with Scenarios and Resources, for Generating Performance Models from UML Designs”, SoSym Special Issue on UML'04, Software and Systems Modeling (SoSyM), Springer-Verlag, August 2006

[97] D.C. Petriu, H. Shen, “Applying the UML Performance Profile: Graph Grammar-based derivation of LQN models from UML specifications”, in Proc. 12th Int. Conf. on Modeling Tools and Techniques for Computer and Communication System Performance Evaluation, London, England, 2002.

[98] D.C. Petriu, H. Shen, A. Sabetta, “Performance Analysis of Aspect-Oriented UML Models", Software and Systems Modeling (SoSyM), Vol. 6, No. 4, pp. 453-471, Dec. 2007.

[99] D.C. Petriu, M. Woodside, “Performance Analysis with UML”, in UML for Real, ed. B. Selic, L. Lavagno, and G. Martin, Eds., Kluwer, pp. 221-240, 2003.

[100] D.C. Petriu, M. Woodside, Dorin B. Petriu, J. Xu, T. Israr, G. Georg, R. France, J.M. Bieman, S.H. Houmb, J. Jürjens, “Performance Analysis of Security Aspects in UML Models”, Proc. 6th Int. Workshop on Software and Performance WOSP2007, Buenos Aires, Argentina, 2007.

[101] R. Pooley, “Software Engineering and Performance: a Roadmap. In: The Future of Software Engineering”, ICSE'2000, Limerick, Ireland, pp. 189-200, 2000.

[102] H. Ratine, F. Mitri, Y. Coenradie, S. Cavassila, D. van Ormondt, D. Graveron-Demilly, “QUEST: time-domain quantitation with advanced prior knowledge”, MAGMA 15 (2002), 373.

[103] Y.R. Reddy, S. Ghosh, R. France, G. Straw, J.M. Bieman, N. McEachen, E. Song, G. Georg, “Directives for Composing Aspect-Oriented Design Class Models”, in A. Rashid, and M. Aksit (eds). Transactions on Aspect-Oriented Software Development I, LNCS 3880, pp. 75-105, Springer, 2006.

[104] D. Redlich, U. Winkler, T. Molka, W. Gilani, “Model-driven engineering in practice: integrated performance decision support for process-centric business impact analysis”, in ICPE '14: Proceedings of the 5th ACM/SPEC International Conference on Performance Engineering, pp. 247-258, March 2014.

[105] J.A. Rolia, K.C. Sevcik, “The Method of Layers”, IEEE Transactions on Software Engineering, Vol. 21, No. 8, 1995, pp. 682-688

[106] P. P. Sancho, C. Juiz, R. Puigjaner, L. Chung, N. Subramanian, “An Approach to Ontology-aided Performance Engineering through NFR Framework”, in Proc. Sixth Int. Workshop on Software and Performance (WOSP’07), Buenos Aires, Argentina, 2007

[107] Chr. Schaffer, R.J. Raschhofer, A. Simma, “EaSy-Sim: A tool environment for the design of complex, real-time systems”, Computer Aided Systems Theory—EUROCAST'95, pp. 358-374, Springer Berlin Heidelberg, 1996.

[108] A. Schmietendorf, E. Dimitrov, “Possibilities of Performance Modeling with UML”, in Performance Engineering, R. Dumke, C. Rautenstrauch, A. Schmietendorf, and A. Scholz, Eds., Springer-Verlag, 2002, pp. 78-95.

[109] W.C. Scratchley, M. Woodside, “Evaluating Concurrency Options in Software Specifications”, in International Conference on Modeling, Analysis and Simulation of Computer and Telecommunications Systems (MASCOTS), College Park, MD, Oct. 1999, pp. 330-338.

[110] H. Shen, D.C. Petriu, “Performance Analysis of UML Models using Aspect Oriented Modeling Techniques", in Model Driven Engineering Languages and Systems, (L.Briand and C. Williams, Eds). LNCS Vol. 3713, pp.156-170, Springer, 2005.

[111] K.H. Siddiqui, M. Woodside, “Performance Aware Software Development (PASD) Using Resource Demand Budgets", Proc Third Int. Workshop on Software and Performance, Rome, July 2002, pp. 275 - 285.

[112] C.U. Smith, Performance Engineering of Software Systems. Addison-Wesley, 1990.

[113] C.U. Smith, C.M. Lladó, “Performance Model Interchange Format (PMIF 2.0): XML Definition and Implementation”, Proc QEST 2004 (First Int. Conf on Quantitative Evaluation of Systems), Enschede, Sept. 2004

[114] C.U. Smith, L. G. Williams, Performance Solutions. Addison-Wesley, 2002.

[115] C.U. Smith, Introduction to software performance engineering: origins and outstanding problems”, in *Int. summer school on formal methods for performance evaluation,* Bertinoro, Springer-Verlag, 2007, pp. 395-428.

[116] C.U. Smith, M. Woodside, “Performance Validation at Early Stages of Development”, Position paper, Performance 99, Istanbul, Turkey, October 1999.

[117] W.D. Smith, “TPC-W*: Benchmarking An Ecommerce Solution”, Intel Corporation

[118] M. Steppler, M. Lott, “SPEET SDL performance evaluation tool”, in *SDL Forum*, pp. 53-68. 1997.

[119] G. Straw, G. Georg, E. Song, S. Ghosh, R. France, J.M. Bieman, "Model Composition Directives", Proc. UML 2004 - Modeling Languages and Applications, 7th Int. Conference, Lisbon, Portugal, LNCS 3273, pp. 84-97, Springer 2004

[120] Transaction Processing Performance Council, www.tpc.org

[121] M.Tribastone, S. Gilmore, "Automatic Translation of UML Sequence Diagrams into PEPA Models", Proc. of 5th Int. Conference on Quantitative Evaluation of SysTems (QEST 2008), pp. 205-214, St. Malo, France, 2008.

[122] M. Woodside, "Software Resource Architecture", Int. Journal on Software Engineering and Knowledge Engineering (IJSEKE), v. 11, no 4, pp. 407-429, 2001.

[123] M. Woodside, C. Hrischuk, B. Selic, S. Bayarov, "Automated Performance Modeling of Software Generated by a Design Environment". Performance Evaluation, vol. 45, pp. 107-124, July 2001.

[124] M. Woodside, Dorin B. Petriu, K. Siddiqui, "Performance-related Completions for Software Specifications", to appear in Proc Int. Conf. on Software Engineering (ICSE 2002), May 2002. Report SCE-01-09, Dept. of Systems and Computer Engineering, Carleton University.

[125] M. Woodside, D.C. Petriu, Dorin B. Petriu, H. Shen, T. Israr, J. Merseguer, "Performance by Unified Model Analysis (PUMA)", In Proc. 5th Int. Workshop on Software and Performance WOSP2005, pp. 1-12, Palma, Spain, 2005.

[126] M. Woodside, G. Franks, D.C. Petriu, "The Future of Software Performance Engineering", Proc Future of Software Engineering 2007, IEEE Computer Society, May 2007, pp. 171-187.

[127] M. Woodside, D.C. Petriu, Dorin B. Petriu, J. Xu, T. Israr, G. Georg, R. France, J.M. Bieman, S.H. Houmb, J.Jürjens, "Performance Analysis of Security Aspects by Weaving Scenarios Extracted from UML Models", The Journal of Systems and Software Special Issue WOSP'2007, Vol.82, pp. 56–74, 2009.

[128] M.Woodside, D.C. Petriu, J.Merseguer, Dorin B. Petriu, M.Alhaj, “Transformation challenges: from software models to performance models", Software and Systems Modeling, DOI: 10.1007/s10270-013-0385-x, published online, 2013.

[129] M. Woodside, J.E. Neilson, D.C. Petriu and S. Majumdar, “The Stochastic Rendezvous Network Model for Performance of Synchronous Client-Server-Like Distributed Software”, IEEE Transactions on Computers, Vol. 44, No. 1, Jan 1995, pp. 20-34

[130] X. Wu, M. Woodside, “Performance Modeling from Software Components”, in Proc. 4th Int. Workshop on Software and Performance (WOSP 2004), Redwood Shores, Calif., Jan 2004, pp. 290-301

[131] J. Xu, M. Woodside, D.C. Petriu “Performance Analysis of a Software Design using the UML Profile for Schedulability, Performance and Time", Proc. 13th Int. Conf. on Computer Performance Evaluation, Modeling Techniques and Tools (TOOLS 2003), Urbana, Illinois, USA, Sept 2003, pp. 291-310.

[132] Y.X. Zeng, “Transforming Use Case Maps to the Core Scenario Model Representation”, SITE, University of Ottawa, June 2005.

[133] ACM Proceedings of the First International Workshop on Software and Performance (WOSP 98), Santa Fe, USA, Oct. 1998.

[134] ACM Proceedings of the Second International Workshop on Software and Performance (WOSP2000), Ottawa, Canada, Sept. 2000.

[135] ACM Proceedings of the Third International Workshop on Software and Performance (WOSP2002), Rome, Italy, July 2002.

[136] ACM Proceedings of the Fourth International Workshop on Software and Performance (WOSP2004), Redwood Shores, California, USA, January 2004

[137] ACM Proceedings of the Fifth International Workshop on Software and Performance (WOSP2005), Palma, Illes Balear, Spain, July 2005.

[138] ACM Proceedings of the Sixth International Workshop on Software and Performance (WOSP2007), Buenos Aires, Argentina, February 2007.

[139] ACM Proceedings of the Seventh International Workshop on Software and Performance (WOSP2008), Princeton, NJ, USA, June 2008.

[140] Proceedings of First Joint WOSP/SIPEW International Conference on Performance Engineering San Jose, CA, USA, January 2010.

[141] Proceedings of Second Joint WOSP/SIPEW International Conference on Performance Engineering, Karlsruhe, Germany, March, 2011.

[142] Proceedings of Third Joint WOSP/SIPEW International Conference on Performance Engineering, Boston, MA, USA, April 2012.

[143] Proceedings of ACM/SPEC International Conference on Performance Engineering, Prague, Czech Republic, April 2013.

[144] Proceedings of ACM/SPEC International Conference on Performance Engineering, Dublin, Ireland, March 22 - 26, 2014.

[145] GreatSPN tool, http://www.di.unito.it/~greatspn/index.html.

[146] jUCMNav CSM export, http://jucmnav.softwareengineering.ca/ucm/bin/view/ProjetSEG/CsmExport.

[147] LQN Online Documentation, http://www.sce.carleton.ca/rads/lqns/lqn-documentation/

[148] SteelCentral NetModeler Suite (formerly OPNET Modeler Suite), http://www.riverbed.com/products/performance-management-control/network-performance-management/network-simulation.html

10. APPENDIX

10.1. EXPANDED CLASS DIAGRAM FOR THE CSM V2 SCHEMA

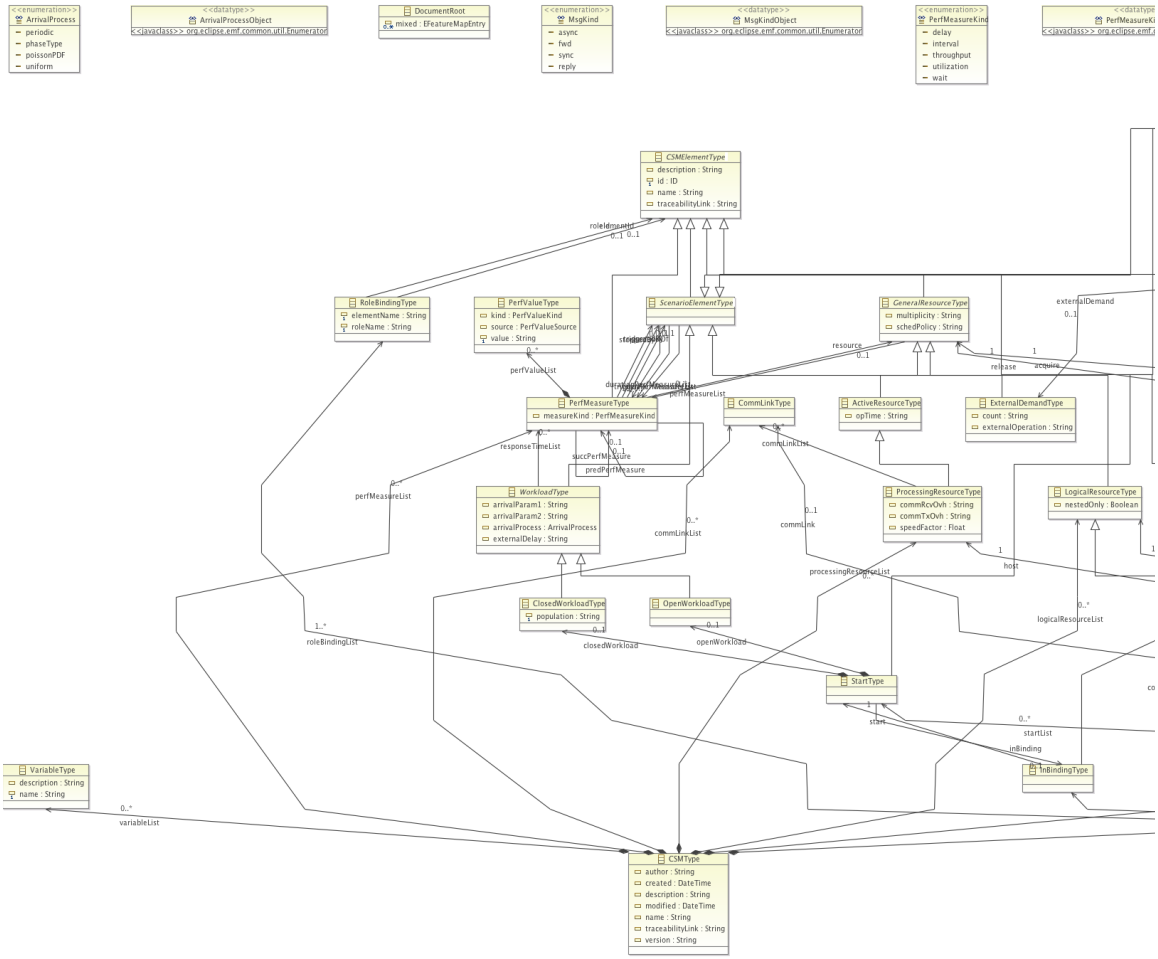


Fig 53 Left third of the class diagram for the CSM v2 schema implementation

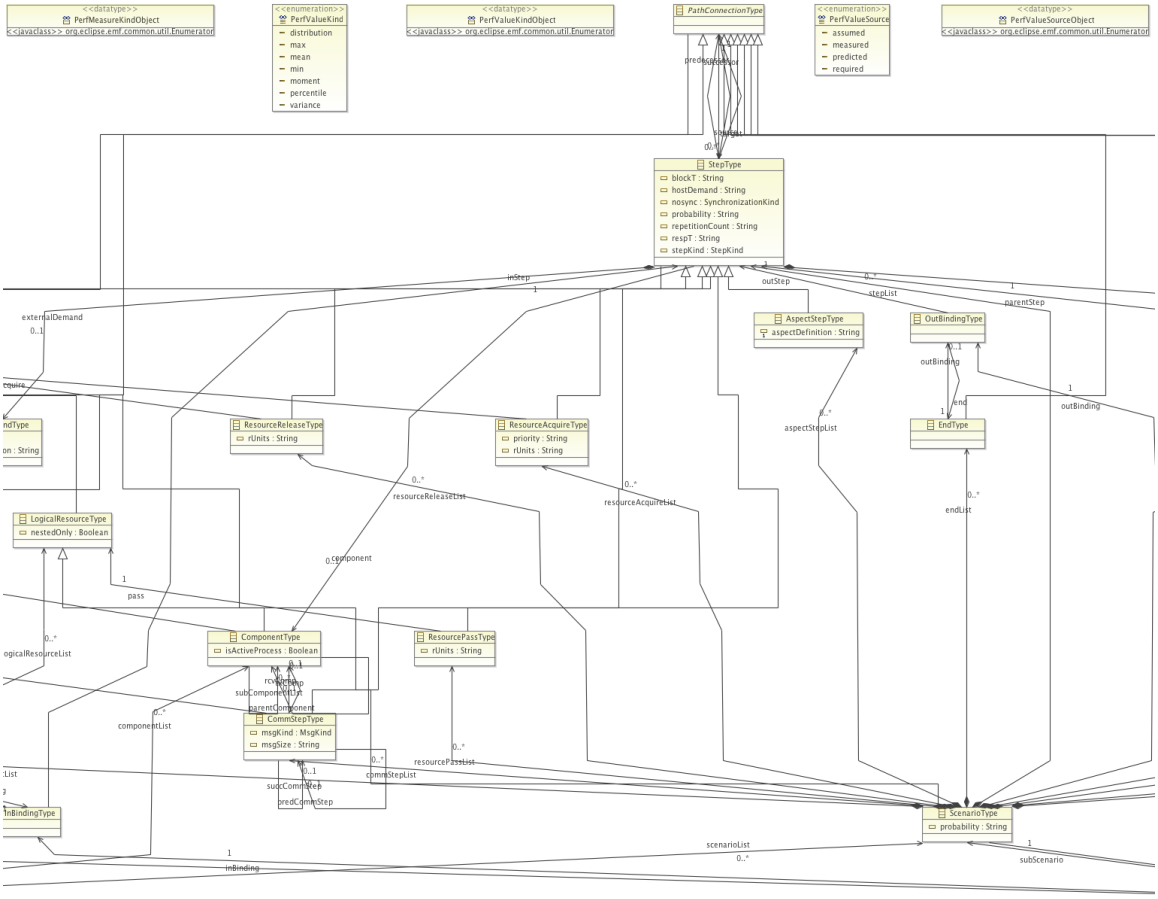


Fig 54 Middle third of the class diagram for the CSM v2 schema implementation

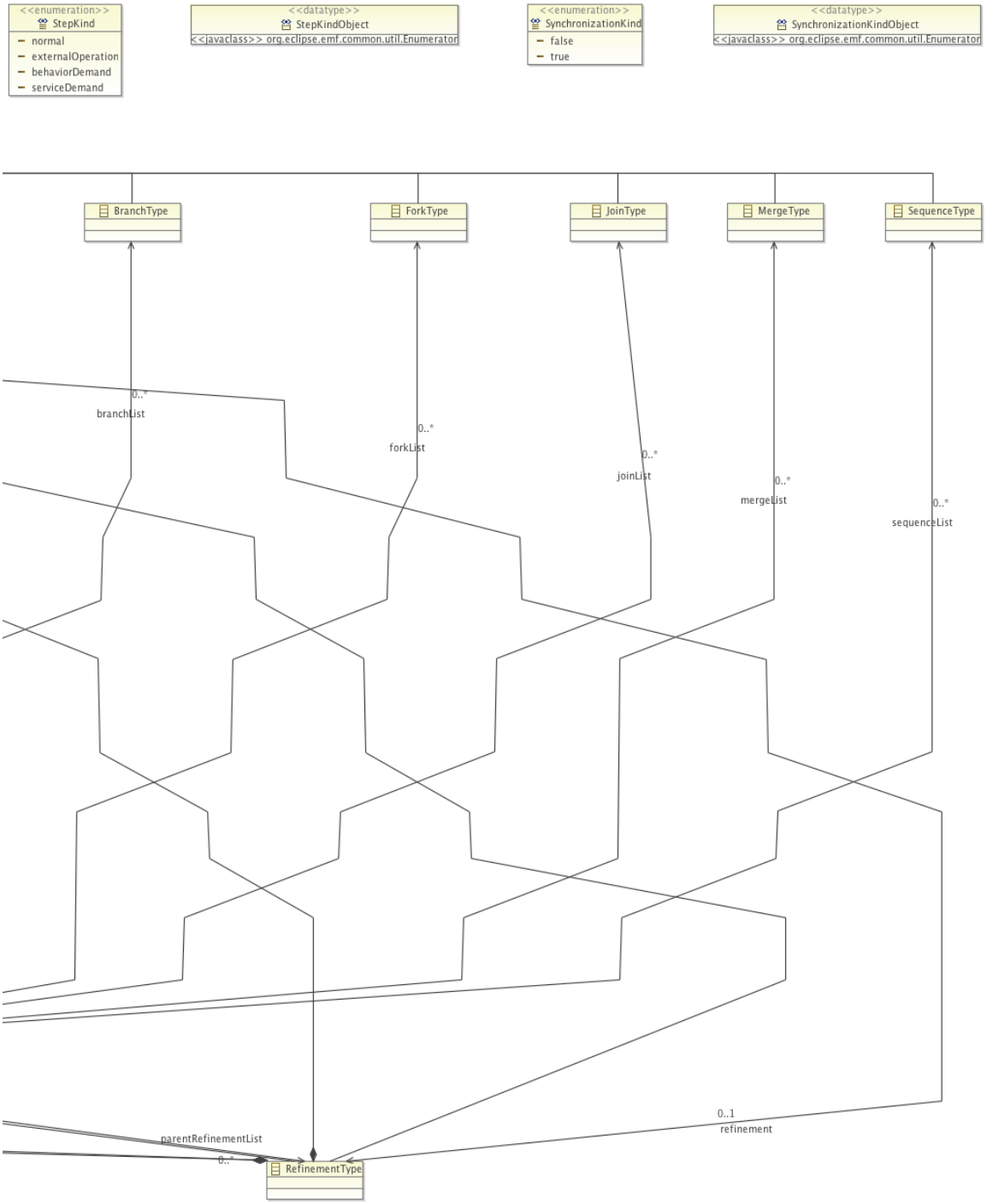


Fig 55 Right third of the class diagram for the CSM v2 schema implementation

10.2. EXPANDED LQN FIGURES FOR THE LABOUR MARKET PORTAL (LMP)

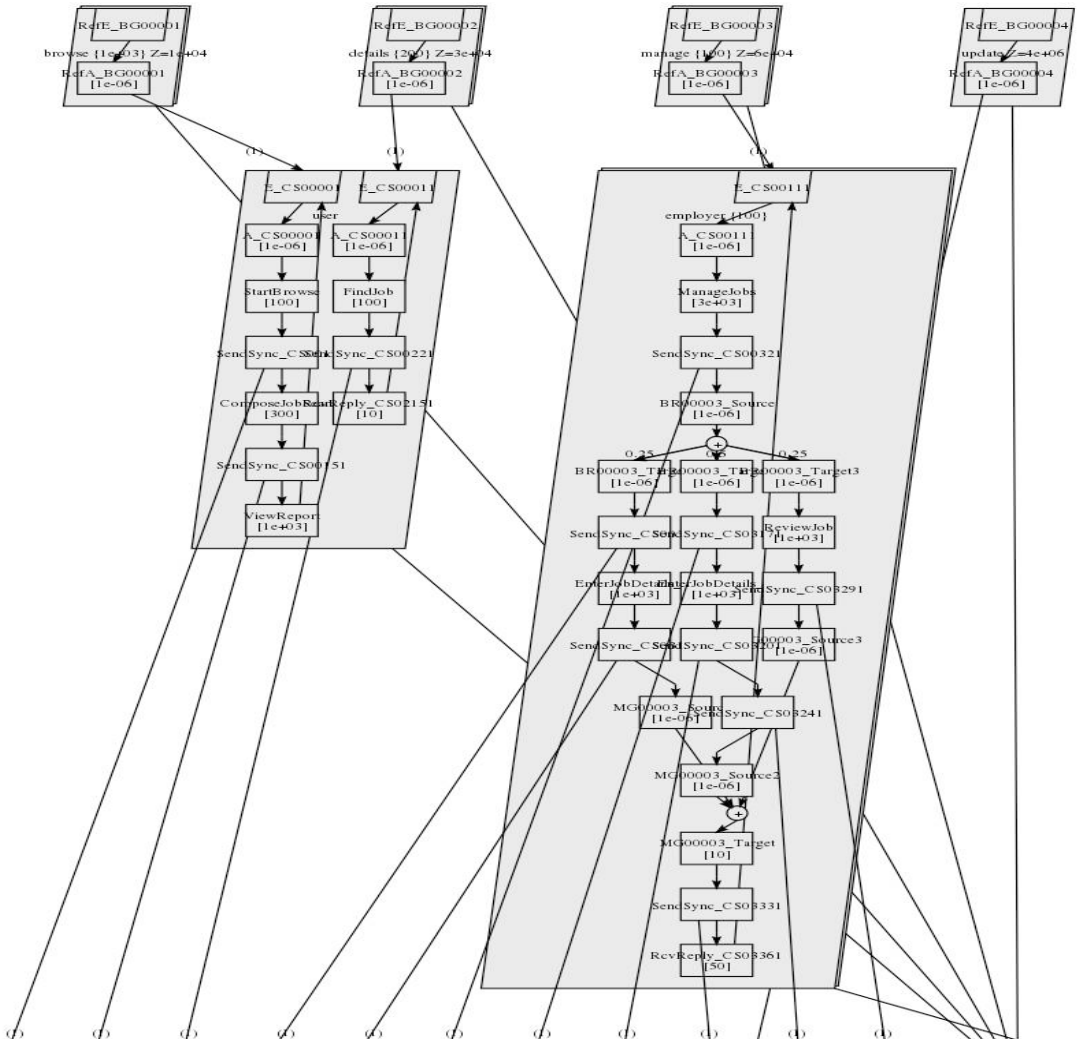


Fig 56 Top third of the LQN model for LMP

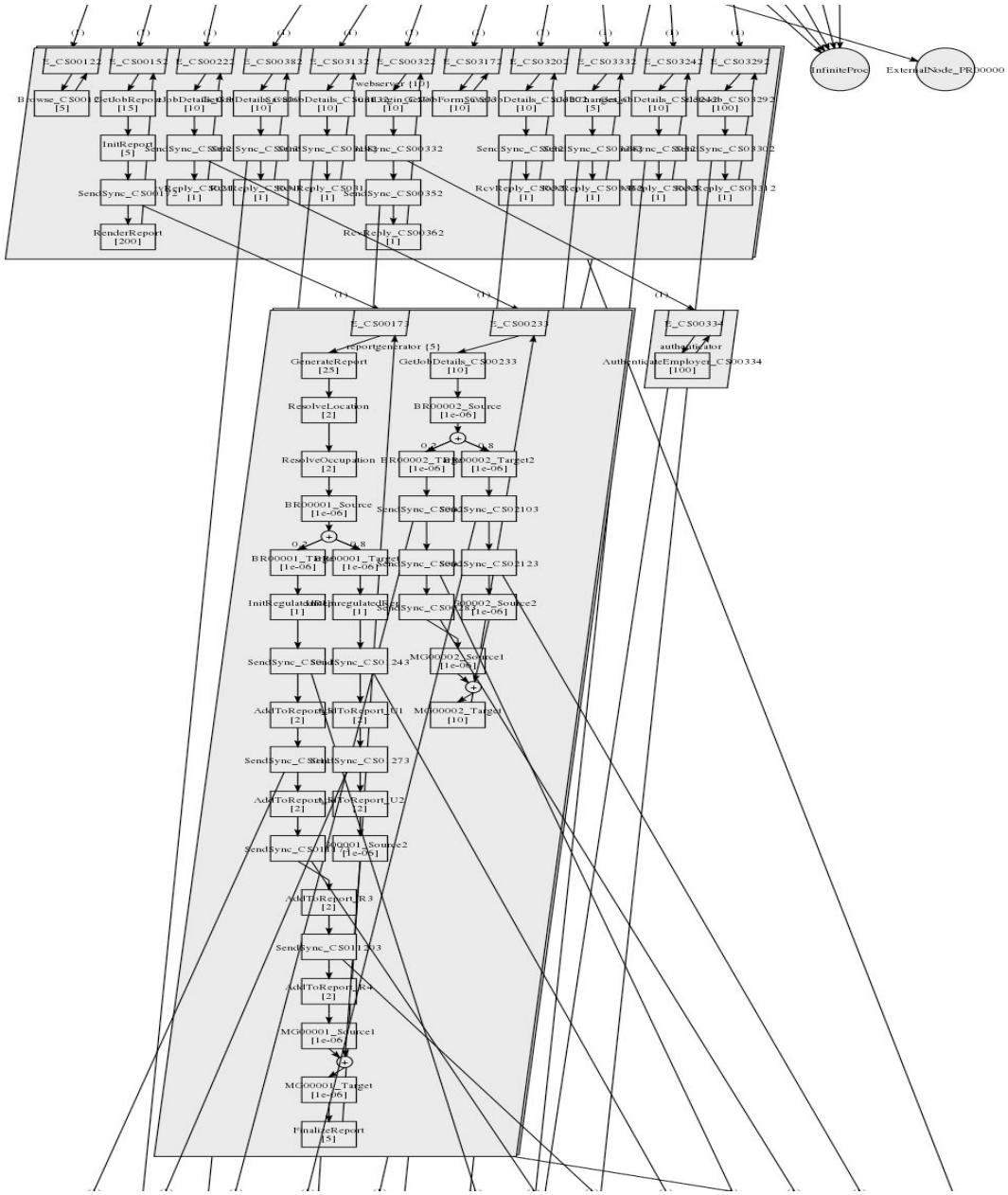


Fig 57 Middle third of the LQN model for LMP

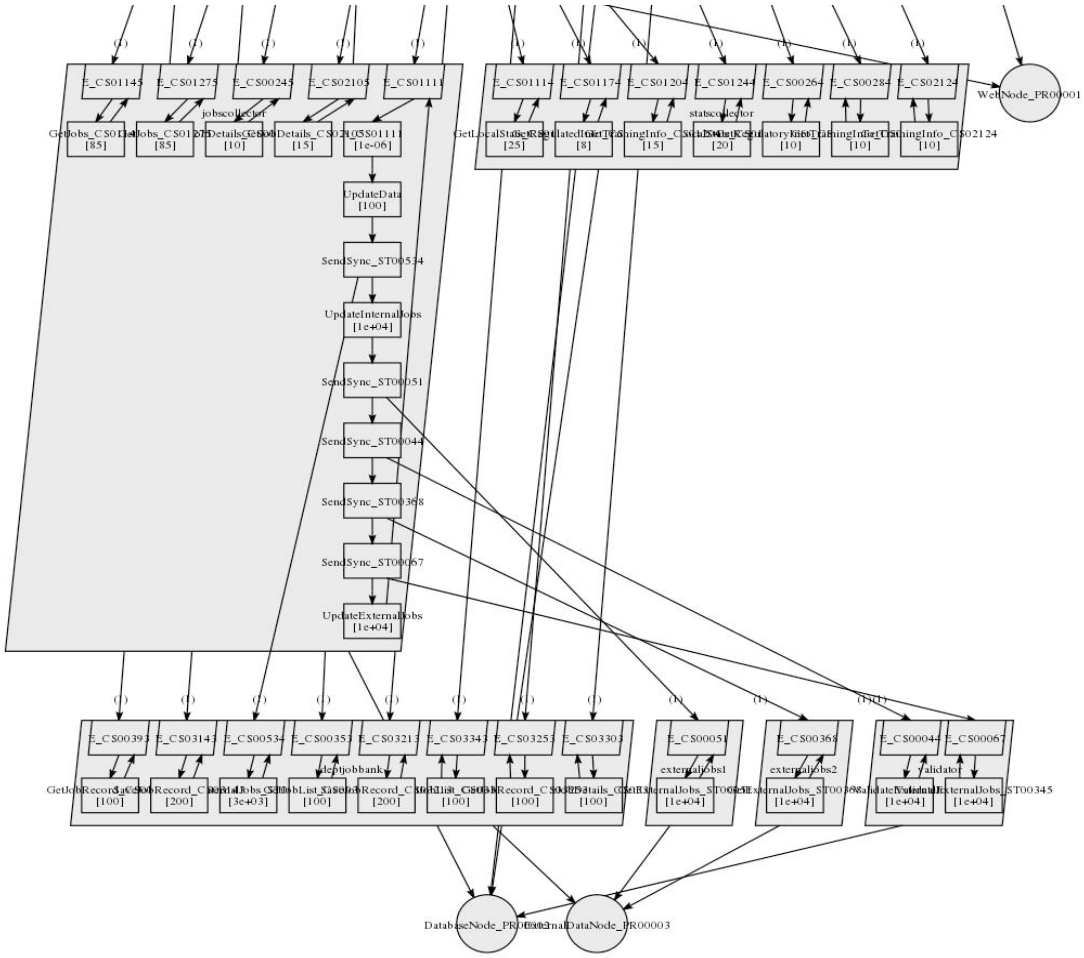


Fig 58 Bottom third of the LQN model for LMP