Automatic Generation of Performance Models for SOA Systems
Mohammad Alhaj
Department of Systems and Computer Engineering
Carleton University, Ottawa ON Canada, K1S 5B6
malhaj@sce.carleton.ca

ABSTRACT
The paper proposes a model-driven approach for deriving performance models from UML design models of Service-Oriented Architecture (SOA) enterprise systems, in order to evaluate their run-time performance from the early development phases. This helps to choose between different architecture, design, and configuration alternatives in order to meet the performance requirements. The source model of the proposed transformation is a platform-independent UML model of the service-oriented system. Aspect-oriented modeling techniques are used to produce a platform-dependent model by weaving aspect models of platform operations into the platform-independent model. The platform-dependent model thus obtained represents the source model for the transformation into a performance model. The UML profile MARTE is used to annotate models with performance information. A healthcare example illustrates the proposed approach.

Categories and Subject Descriptors
C.4 [Performance of Systems]: Design Studies, Modeling Techniques; D2 [Software Engineering]: D.2.11 Software Architectures

General Terms
Performance, Design, Languages.

Keywords
SOA, UML, LQN, performance analysis, model transformation.

1. INTRODUCTION
Service-Oriented Architecture (SOA) is a software development paradigm for building and deploying software applications as a set of reusable composable services [14]. SOA is applied to enterprise systems, web-based applications, multimedia, health-care, e-government, etc. Meeting performance requirements (such as response time and throughput) is a major concern for many SOA systems. This paper proposes a model transformation approach for generating automatically a performance model from the design model of an enterprise SOA system. The performance model will allow developers to evaluate the system performance from the early development stages and to assess the performance effects of different design alternatives.

The design model playing the role of source model for the proposed transformation is a UML model used for software development, extended with performance annotations. In the past decade, OMG has standardized two UML profiles for the real-time domain: the “UML Profile for Schedulability, Performance and Time (SPT)” defined for UML 1.X, and the “UML Profile for Modeling and Analysis of Real-Time and Embedded systems (MARTE)” [10] defined for UML 2.X. Both SPT and MARTE include their own performance subprofiles, which allow annotating UML models of real time systems with quantitative resource demands and other information required for performance analysis. The standardization efforts on the one hand and the emergence of the Model-Driven Architecture on the other hand, have triggered a lot of research on the automatic derivation of different types of performance models from annotated UML software models. Our research group contributed significantly to this field, not only by participation in the OMG working groups that have defined SPT and MARTE, but also through work on model transformations for automatic derivation of performance models. An example of such work is the Performance from Unified Model Analysis (PUMA) transformation [18], which represents the foundation for the approach for SOA presented in this paper.

The transformation proposed in this paper extends PUMA in two directions. First, the UML design model reflects now the SOA layers, as discussed in more detail in Section 3: a) workflow model; b) service architecture model; and c) service behavior model. For (b) we use a new OMG profile called the Service-Oriented Architecture Modeling Language (SoaML) [9], which extends UML with the ability to define the service structure and dependencies, to specify service capabilities and classification, and to define service consumers and providers. The mapping of workflows and services to the target performance model is treated differently in the proposed transformation.

Another difference from PUMA is that in our transformation we separate the platform model from the platform-independent model of the SOA system (i.e., workflows and actual services). This allows us to reuse the performance characteristics of different platforms, making the building of the performance model more efficient. To address the problem of composing the platform independent model of a SOA system with the platform model we apply Aspect Oriented Modeling (AOM) [4][7] [19]. An aspect model in its generic form is initially developed to describe a crosscutting concern – in this case the platform operations (e.g., service invocation). The generic aspect model is then bound to the context of the application, becoming a context-specific model and it is composed with the primary model at defined joint points (e.g., in every place where a service invocation take place).

The paper is organized as follows: Section 2 describes the proposed approach called Performance from Unified Model Analysis for SOA (PUMA4SOA). Section 3 describes the
platform independent UML model of the SOA system annotated with performance information. Section 4 presents the aspect model of a platform operation and the AOM binding and composing the aspect model with the primary SOA model. Section 5 presents the transformation principles from the design UML models to the intermediate Core Scenario Model (CSM) and from CSM to LQN. Section 6 discusses some performance results. Section 7 discusses related work and Section 8 presents the conclusions and directions for future work.

2. PUMA4SOA

The paper proposes a modeling approach called Performance from Unified Model Analysis for SOA (PUMA4SOA) that aims to derive performance models for enterprise SOA systems. The proposed approach is an extension of the PUMA transformation chain introduced in [18]. PUMA improves the consistency and reduces the complexity of model transformation by providing a unified interface between different software design models that can be accepted as input and different performance model that can be produced as output. The differences between the extended approach PUMA4SOA and PUMA consist in a) the design model used as an input for the former represents the SOA layers; and b) aspect-oriented techniques are used for modeling the platform operations. Fig.1 shows the steps of PUMA4SOA. The top left-most block represents the main difference from PUMA. There are three input models: a) PIM of the SOA system (representing the workflow, service architecture and service behavior models); b) deployment of the service-oriented software to hardware devices in the enterprise system and c) aspect platform models (representing the service middleware platform).

First, an aspect-oriented modeling (AOM) approach is used to generate the Platform Specific Model (PSM). Aspect models of platform operations are woven into different locations of the PIM (the primary model). Several aspects can be introduced for the platform, such as service discovery, service invocation, and service security. In this paper we are using as an example a service invocation aspect. Next, the PSM design model is transformed into a Core Scenario Model (CSM), similar to the original PUMA [18]. CSM uses a light-weight metamodel [12], and is able to capture the performance characteristic from different design models. CSM is then transformed into a performance model. Several types of performance models can be obtained, such as Queueing Networks, Layered Queueing Networks (LQN) and Petrinets [18]. Finally, an existing performance solver is used to perform the analysis, and the results are fed back to improve the design model.

A challenge faced by PUMA4SOA is related to the workflow model included in PIM. Since the workflow model describes the business process model, it can be represented by different graphical modeling notations, such as the Business Process Modeling Notation (BPMN) standardized by OMG, or the UML activity diagram. The advantage of BPMN is that it supports different types of events (such as timer, compensation, escalation, and message), choreography, transactions, tasks (such as send, receive, user, manual). BPMN may seem better suited because of its richer business semantic. However we chose to use UML activity diagrams because of two reasons: a) it is desirable to use a single modeling language for all SOA levels i.e. workflow, structure, and behavior; and b) UML activity diagrams can be extended with the MARTE profile for performance annotations, while BPMN cannot. The limitations of UML activity annotations can be overcome by using a UML profile that extends UML with BPMN concepts. OMG has already issued an RFP for such a profile, but it is not available yet. Until then, we will use our own stereotypes to extend activity diagrams with desired BPMN concepts.

The second challenge faced by PUMA4SOA is related to the representation in the performance model of the steps performed by humans, which may appear in the workflows. The time scale of the activities performed by humans and by computers is very different (by orders of magnitude) so careful consideration is required. Usually in LQN models, human activities are represented by “think times” at the beginning of every scenario or can be modeled by separate tasks. The third challenge regards the level at which aspect composition is performed: UML model level, CSM level or LQN level [18]. (Please note that in Fig.1, the aspect composition takes place in the UML model.) The definition of the platform aspect models is done in UML, but we could transform the aspect models separately from UML to CSM to LQN, and postpone the actual composition until later. Performing aspect composition at UML design model level allows the software developer to view the composed model (PSM). However, composing at this level is a complex process and may lead to complex models that are hard to inspect. On the other hand, performing aspect composition at the CSM level is simple because CSM has a light-weight metamodel, and its elements are more abstract. However, the software developer will not able to view the composed model. At the LQN level, aspect composition is also simple, but may not be always possible because of the difference in level of abstraction (aspect models elements may not map to separate LQN tasks or entries). It seems that composing aspect at the CSM level is the best tradeoff for PUMA4SOA, but more work is required to draw a firm conclusion.

3. PLATFORM-INDEPENDENT MODEL

PUMA4SOA approach begins with a PIM software model, which consists of three levels of abstraction: the Workflow Model that represents a set of business processes, the Service Architecture Model that describes the service capabilities arranged in a hierarchy showing anticipated usage dependencies, and the Service Behavior Model that describes the behavior of the invoked services. The MARTE profile in [10] defines the stereotypes which are used to indicate the performance characteristics of model elements. Each level represents a part of the performance details that will be used together with the other parts to build the performance model. To illustrate the PIM views, we use a health

![Figure 1. The steps of PUMA4SOA](image-url)
Figure 2. Workflow model of Electronic Prescription System

Figure 3. Service Architecture model of Electronic Prescription System

care case study for an Electronic Prescribing System [11]. In the workflow model shown in Fig.2, the initiator begins by requesting a prescription. Next, the physician identifies the patient by processing his record, followed by two processes evaluating the patient factors (such as allergies, pregnancy, and current/past medical condition) and medication factors (such as allergy information, warnings, and adverse reactions). Next, a process checks the insurance coverage. Once this is done, the physician decides whether to prescribe or not. If yes, it updates the patient record and notifies the pharmacy. The workflow shows different swimlanes that contain the activities of the different participants. A swimlane is stereotyped as <<PaRunTInstance>> to indicate a concurrent participant that will be executed by a specified process instance. Each activity is stereotyped as <<PaSep>> to indicate a scenario step. The first activity stereotyped as <<GaWorkloadEvent>> to define the stream of events that produces the workload. A control flow arc between two
swimlanes representing the communication between participants is stereotyped as <<PuCommStep>> to indicate the conveyance of a message. Fig. 3 shows a SoaML Service Architecture model which defines the participants, their binding roles in each service contract and the capabilities of each participant. Components are stereotyped as <<participant>> to indicate parties that provide or consume services. Ports are stereotyped with either <<Request>> to indicate the consumption of a service, or <<Service>> to indicate the offered service.

UML Sequence diagrams are used to model the behavior of each activity defined in the workflow model in Fig. 2. Fig. 4 shows as an example the behavior of process patient record activity. UML Deployment diagram in Fig. 5 shows the allocation of software to hardware. Nodes are stereotyped with either <<GaCommHost>> to indicate a physical communication link or <<GaExecHost>> to indicate a processor resource hosting the components deployed to it. Artifacts are stereotyped with <<SchedulableResource>> to indicate a process with a thread pool.

4. PLATFORM ASPECT MODELS
Aspect-Oriented Modeling (AOM) assumes there is a primary model that describes the core design decisions and a set of aspect models, each describing a concern that crosses the primary model [4][18]. PUMA4SOA considers the PIM model as the primary model, and the service middleware operations as aspect models. An aspect model can be seen as a pattern, independent of any primary model it may be composed with. As in [4] it is termed a generic aspect model and may be thought of as a template with formal parameters. For each insertion in the primary model, the template is instantiated and its formal parameters are bound to concrete values using binding rules, to give a context-specific aspect model. There are two kinds of formal parameters: UML model related (such as model element names, as described below) and performance annotations related (which are handled by variables and expressions in the MARTE annotations). The composed model is generated by weaving the context-specific aspect models into the primary model at different locations called join points, which are identified based on a set of conditions called point-cuts.

Fig.7 describes the deployment diagram of the generic Hosts and Artifacts used in the service invocation aspect. A vertical bar ‘|’ at the beginning of an identifier indicates a generic role name, as in [4]. The Service Request Invocation, in Fig. 6, describes the behavior of the service invocation generic aspect. At the Service Request Invocation, the sends RequestService to the |soapClientC|, which in turn request the |xmlParserC| to parse the message. |soapClientC| serialize the message to the |soapClientP| at the provider side. |soapClientP| de-serializes the message, and request the |xmlParserP| to parse it. Then, |soapClientP| requests the ServiceInvocation from the |Provider|. At the Service Response Invocation, similar steps are performed after the |provider| executes the service operation.

The first step for composing an aspect is to identify the join-point in the UML primary model according to the point cut. In our example the point-cut for service invocation is defined as follows: a message in the service behavior model (Fig. 4) such that: a) the message is modeled as a connection in the service structure model (Fig. 3); b) the port of the sender is stereotyped with <<request>> in the service structure model; c) the port of the receiver is stereotyped with <<service>> in the service structure model. Fig. 4 illustrates the join-point of the AccessPatientRecord service. Once the join-point is identified, the context-specific aspect model for this join-point is generated in two phases: binding the resource roles to actual resources, and then assigning context-specific performance values to steps, processing demands, probabilities, etc. The actual resources can be either found in the primary model, or they can be newly defined resources. For instance, the generic role |client| is bound to the sender of the service request and that of |provider| to its receiver. An example of parametric performance annotation is the execution time of the marshaling operation in the generic aspect that can be given as an expression depending on the message size. Once the actual message size of the service request is known, the execution time can be also determined. Using the sequence diagram AccessPatientRecord Service shown in Fig. 4 as a primary model, we obtain the resource bindings as shown in Table 1. Table 2 describes the concrete values of the variable of the stereotypes defined in
Generic models. The values of the service demands and message sizes are similar to those used in [12] [13].

### Table 1. Binding generic to concrete roles

<table>
<thead>
<tr>
<th>Generic Aspect</th>
<th>Context-Specific Aspect</th>
</tr>
</thead>
<tbody>
<tr>
<td>AccessPatientRecord</td>
<td>AccessPatientRecord</td>
</tr>
<tr>
<td>Invocation</td>
<td>Invocation</td>
</tr>
<tr>
<td>XMLParserC</td>
<td>XMLParserC (new)</td>
</tr>
<tr>
<td>soapClientC</td>
<td>soapClientC (new)</td>
</tr>
<tr>
<td>provider</td>
<td>provider</td>
</tr>
<tr>
<td>XMLParserP</td>
<td>XMLParserP (new)</td>
</tr>
<tr>
<td>soapClientP</td>
<td>soapClientP (new)</td>
</tr>
</tbody>
</table>

### Table 2. Binding generic values

<table>
<thead>
<tr>
<th>AccessPatientRecord</th>
<th>Service Demand</th>
</tr>
</thead>
<tbody>
<tr>
<td>hostDemand=25 ms</td>
<td></td>
</tr>
<tr>
<td>hostDemand=23.75 ms</td>
<td></td>
</tr>
<tr>
<td>hostDemand=9.2 ms</td>
<td></td>
</tr>
<tr>
<td>hostDemand=125 ms</td>
<td></td>
</tr>
<tr>
<td>hostDemand=20 ms</td>
<td></td>
</tr>
</tbody>
</table>

Fig. 8 shows the composed model after weaving the context-specific aspect model obtained from the generic aspect in Fig. 6 into the primary model shown in Fig. 4. So far we considered that aspect weaving takes place in the UML model, but in fact it can be performed in the CSM or even LQN model. Postponing the aspect composition has the advantage that CSM and LQN have smaller metamodels, so the composition is easier to perform. More details about aspect weaving at the UML level can be found in [7] and at CSM level in [19].

5. MODEL TRANSFORMATION APPROACH

5.1 From annotated UML to CSM

The next step is to transform the annotated platform-specific model of the SOA system into CSM. The Core Scenario Model (CSM) is a main part of PUMA. It supports the creation of many different kinds of performance models [18]. The CSM metamodel captures the performance concepts from the SPT profile, and recently was updated for MARTE as well. Fig. 9 illustrates two CSM scenarios generated from UML+MARTE design models. Fig 9a is the top level CSM scenario representing the workflow diagram. An atomic step is drawn as a rectangle with a single line, while a composite (or refined step) is a rectangle with a double-line on both sides. A refined step describes a subscenario. Branch-Merge pathconnections are decision nodes. A scenario begins with Start, followed by ResourceAcquire, and finishes with ResourceRelease and End (which are generated for an initial node, respectively a final node in an activity diagram). ResourceAcquire and ResourceRelease are generated when an activity diagram edge is crossing partitions; the resource is the execution thread associated to the partition. Generating CSM from sequence diagrams is quite similar to generating it from activity diagrams. For a sequence diagram, ResourceAcquire is generated when a lifeline first receives a synchronous or asynchronous call message or execution starts first. A ResourceRelease is generated when a lifeline sends an asynchronous message or its execution finishes. A Step is generated when there is an incoming message to the lifeline. Fig 9b describes AccessPatientRecord service generated from the composed model defined in the Fig 8, corresponding to a composite node in Fig. 9a. The model transformation from annotated UML model to CSM is described in more detail in [1].

5.2 LQN: The target model

The LQN model [18] is an extended queueing network model able to represent nested services (see Fig.10). An LQN model is a set “tasks” that offer services represented by “entries”. The entries of a task may send requests to entries of other tasks. LQN is used to model several types of system behavior and inter-process communication style. Tasks represent both software and hardware devices, and allow for multi-threading and nested services. Graphically, the software tasks are drawn with thick rectangles and the entries with attached thin rectangles. The hardware devices are represented as ellipses. The LQN solver is used to produce task service times (including waiting for nested services), queueing delays, processor utilization, response time, and throughput. The results are used to identify the performance hotspots in the system.

5.3 From CSM to LQN

We mentioned that our PUMA4SOA is an extension of PUMA, which is a transformation approach developed by our group [18]. The first difference between the two approaches is in the design model of the source model. PUMA takes as input an annotated UML model containing the structural and a behavioral view represented by one or more scenarios, while PUMA4SOA targets SOA systems which require additional levels of abstraction, as explained in the previous sections. Other differences between the two approaches are due to a higher emphasis on service contracts and the fact that we use aspect-oriented models in PUMA4SOA to capture the performance overheads of the middleware platform. The transformation approach from the software design to the performance models is performed in both approaches using an intermediate CSM model described in the previous section. The structure of the LQN model is generated from resources identified in the high-level service architecture and deployment. In principle, active software component instances and hardware devices (which are all resources) are mapped to LQN tasks. In some cases, LQN tasks are also generated from passive instances representing logical resources shared by active instances. Task entries correspond to service access points. The request arcs between task entries are obtained from the two-level behavioral description (workflow level and service level). LQN activities correspond to workflow activities executed in the same swimlane. The resource demands (such as execution times) are obtained from the attributes of different steps.

6. PERFORMANCE ANALYSIS

In this section, we define two design alternatives for analyzing the performance of the Electronic Prescribing System: a) finer service granularity corresponding to the service architecture from Fig. 3, and b) coarser service granularity, where the invocations of low-level services for accessing the Prescription server are replaced with regular calls. The second solution practically integrates the functionality of the lower level services in the higher-level services provided by the component Physician server. The LQN solver produces a complete performance characteristics of the system (such as response time, throughput, utilization), however due to the paper limitation, we will show only the response time. The LQN results will show the response time reduction obtained by making fewer expensive service invocations using SOAP and XML in the same scenario. Fig. 10 shows the LQN model generated for the Electronic Prescribing System for the first alternative. The main purpose of our performance analysis is to find the performance bottleneck (i.e., software or hardware components that saturate first and throttle the system). After identifying the bottleneck, we apply a series of hardware and/or...
software modifications to mitigate the bottleneck and to improve the response time and the throughput of the overall system. The results of the performance analysis, shown in Fig. 11 give the response time of the system versus the number of users ranging from 1 to 100.

The results for the system with finer service granularity are given in Fig. 11.a. *Initial:* The multiplicity of all tasks and hardware devices is 1, except for the number of users. The physician Server is the software bottleneck. A: The bottleneck is resolved by allowing physician server multithreading. The response time is reduced slightly, and moves the bottleneck to the clinicHost. B: The bottleneck is resolved by increasing the multiplicity of clinicHost to 5. The response time is reduced by 38% (with respect to A), and moves the bottleneck to the MW_physician. C: The bottleneck is resolved by allowing MW_physician multithreading. The response time is reduced by 40% (with respect to A), and moves the bottleneck to the MW_prescription. D: The bottleneck is resolved by increasing the

The results for the case with coarser service granularity are shown in Fig. 11.b. *Initial:* The multiplicity of all tasks and hardware devices is 1, except for the number of users. The physician Server is the software bottleneck. A: The bottleneck is resolved by allowing physician server multithreading. The response time is reduced by 45% (with respect to the initial), and moves the bottleneck to the clinicHost. B: The bottleneck is resolved by increasing the multiplicity of clinicHost to 5. The response time is reduced by 36% (with respect to A), and moves the bottleneck to the MW_physician. C: The bottleneck is resolved by allowing MW_prescription multithreading. The response time is reduced by 38% (with respect to B), and moves the bottleneck to the InsuranceHost. D: The bottleneck is resolved by increasing the

![Figure 8. Model for Process Patient Record after composition of aspects for service invocation and reply](image)

![Figure 9. CSM for Electronic Prescribing System](image)
multiplicity of InsuranceHost to 2. The response time is reduced by 49% (with respect to C).

Fig. 11.c. presents the difference between the two alternatives at the bases (C). Both alternatives are similar in the number of hardware and the software components, except that the coarser service granularity option performs a fewer service invocation through the web service middleware. The reduction in response time is high, which is about 60% for the large number of users. The reduction can be justified since 7 service invocations are executed by MW_Prescription in the finer service granularity option are replaced with regular messages in the coarser service granularity option. The effect of service granularity on system performance should be evaluated at an early design stage, when the service boundaries are defined. The proposed analysis helps the modeler to decide on the right granularity level, making a trade-off between system performance and the level of coupling between the system services.

7. RELATED WORK
With the emergence of model-driven SOA, the evaluation of non-functional properties of service systems based on models has attracted a lot of interest in the research community. In [15] a model transformation framework is proposed to automatically include the architectural impact and the performance overhead of the middleware layer in distributed systems. Using this framework, architects can model the system in a middleware-independent way and then obtain a platform-specific model by composition. A LQN model build by hand is used for performance evaluation. Another modeling framework is proposed in [16] based on an intermediate language called KLAPER, which is used to bridge the gap between the design-oriented and analysis-oriented notations. KLAPER captures the information that is relevant for the analysis of non-functional characteristics of component-based systems, such as performance. KLAPER can be used to derive QN performance model. In [1] a model-driven approach is proposed, which automatically selects the set of available services, transforms a design model of service composition into an analysis model, and then feeds a probabilistic model checker for quality prediction. Another model-driven approach for development and evaluation of non-functional properties such as performance and reliability is based on the Palladio Component Model (PCM), which allows specifying component-based software architectures in a parametric way [6].

A parametric performance completion for message-oriented middleware was proposed for PCM in [6], which plays a role similar with the middleware aspect models in our approach. The project SENSORIA [17] aims at developing a novel approach to the engineering of service-oriented software systems, which
includes service-oriented extensions to the UML, a mathematical basis formed by a family of process calculi, a language for expressing context-dependent soft constraints and preferences, qualitative and quantitative analysis methods, and model transformations from UML to process calculi. One of the best known technologies for realizing SOA systems are Web Services, so it is not surprising that there are many efforts to model and analyze the performance of such systems. As Web Services use the SOAP protocol, which is XML-based and introduces performance overhead to parse and serialize XML documents, there was a lot of interest in studying the performance impact of different SOAP implementations, as for instance in [5][3]. Other research on building performance models for web services takes a two-layered user/provider approach in [3][8]: the user is a set of workflows and the provider a set of services deployed on a physical system. In [3] a LQN model is generated for the system. In [8] the queueing network formalism is used to derive performance bounds.

8. CONCLUSIONS AND FUTURE WORK
The paper proposes an approach to derive automatically LQN performance model from UML software models of SOA systems with MARTE performance annotations. We are currently implementing the proposed transformation approach as an Eclipse plugin that accepts as an input a UML design model exported from a UML editor, such as IBM Rational Software Architecture. Our implementation is based on the Eclipse UML2 Plugin that supports the UML2.X metamodel. We are planning in future work to extend the UML activity diagram with more business modeling concepts. We will use the full capacity of the SoaML metamodel in order to investigate the performance effects of allocating service capabilities to services and participants in different ways. We will investigate aspect compositions at the three levels (UML system model, CSM, LQN) to find the best tradeoff. We will implement traceability between the source, intermediate and target model, especially related to workflows and services. We will also introduce other aspects of middleware services, such as service security, service discovery, and service description. We intend to validate the generated performance model against measurements of a real system running on different platforms. We will apply our approach to different systems in the health care domain.

9. ACKNOWLEDGMENTS
The author would like to thank his PhD supervisor, Prof. Dorina C. Petriu for her continuous guidance and support. This research was partially supported by the Natural Sciences and Engineering Research Council (NSERC) and industrial and government partners, through the hSITE Strategic Research Network.

10. REFERENCES